# Contents

## Part 1  Getting Started with SuiteScript

**Chapter 1**  SuitScript - The Basics  14

**Chapter 2**  What is SuiteScript?  16

*What can I do with the SuiteScript API?*  16
*Using the SuiteScript API with NetSuite Records*  18

**Chapter 3**  Setting Up Your SuiteScript Environment  20

*Environment Setup Overview*  20
*Configuring NetSuite for SuiteScript*  20
*Setting Up Your SuiteScript Development Environment (SuiteCloud IDE)*  26
*Working with Your SuiteScript Development Environment (SuiteCloud IDE)*  42
*Working with IDEs Other Than SuiteCloud IDE*  70

## Part 2  Running a Script in NetSuite

**Chapter 4**  Step 1: Create Your Script  74

**Chapter 5**  Step 2: Add Script to NetSuite File Cabinet  75

**Chapter 6**  Step 3: Attach Script to Form  76

**Chapter 7**  Step 4: Create Script Record  79

*Steps for Creating a Script Record*  81
*Creating a Custom Script Record ID*  85

**Chapter 8**  Step 5: Define Script Deployment  87

*Steps for Defining a Script Deployment*  88
*Creating a Custom Script Deployment ID*  89

**Chapter 9**  Viewing Script Deployments  90

**Chapter 10**  Using the Scripted Records Page  91

**Chapter 11**  Using the Script Execution Log  94

## Part 3  Scripting Records, Fields, Forms, and Sublists
Chapter 23  Scheduled Scripts  259

Understanding Scheduled Script Execution .......................... 260
Deploying Scheduled Scripts ........................................... 264
Creating Multiple Deployments for a Scheduled Script .............. 268
Running Scheduled Scripts Using nlapiScheduleScript ............. 272
Setting the Scheduled Script type Argument ......................... 272
Setting Recovery Points in Scheduled Scripts ......................... 273
Understanding Memory Usage in Scheduled Scripts ................. 274
Monitoring Scheduled Scripts ........................................ 276
Scheduled Script Samples ............................................. 277
Using Multiple Script Queues to Run Scheduled Scripts ............ 283

Chapter 24  Portlet Scripts  285

What Are Portlet Scripts? .............................................. 285
Portlet Script Execution .............................................. 286
Assigning the Portlet Preference to a Script Parameter ............... 287
Running a Portlet Script in NetSuite ................................ 287
Displaying Portlet Scripts on the Dashboard ......................... 288
Portlet Scripts Samples ............................................. 288

Chapter 25  Mass Update Scripts  296

What Are Mass Update Scripts? ....................................... 296
Mass Update Script Execution ....................................... 298
Running a Mass Update Script in NetSuite .......................... 299
Mass Update Scripts Samples ....................................... 300

Chapter 26  Bundle Installation Scripts  305

What are Bundle Installation Scripts? ............................... 305
Setting Up a Bundle Installation Script ............................. 307
Sample Bundle Installation Script .................................. 312

Part 5  Setting Runtime Options

Chapter 27  Setting Runtime Options Overview  317

Chapter 28  Setting Script Execution Event Type from the UI  318

Chapter 29  Setting Script Execution Log Levels  320

Chapter 30  Executing Scripts as Admin  322

Chapter 31  Setting Available Without Login  324

Chapter 32  Setting Script Deployment Status  326

Chapter 33  Defining Script Audience  328
Part 6  Creating Script Parameters (Custom Fields)

Chapter 34  Creating Script Parameters Overview  331
Chapter 35  Why Create Script Parameters?  332
Chapter 36  Creating Script Parameters  333
Chapter 37  Referencing Script Parameters  336
Chapter 38  Setting Script Parameter Preferences  337

Part 7  Searching with SuiteScript

Chapter 39  Searching Overview  342
Chapter 40  Understanding SuiteScript Search Objects  343

Chapter 41  Search Samples  346

Chapter 42  Supported Search Operators, Summary Types, and Date Filters  361

Chapter 43  UI Objects Overview  367
Chapter 44  Creating Custom NetSuite Pages with UI Objects  369
Chapter 45  InlineHTML UI Objects  371

Chapter 46  Building a NetSuite Assistant with UI Objects  373

NetSuite UI Object Assistant Overview  373
Understanding NetSuite Assistants  373
Using UI Objects to Build an Assistant  377

Part 9  Debugging SuiteScript

Chapter 47  Debugging SuiteScript  393

Debugging Overview  393

Chapter 48  Before Using the Debugger  395

Chapter 49  Ad-hoc Debugging  397

Chapter 50  Deployed Debugging  403

Chapter 51  Debugger Metering and Permissions  410

Chapter 52  SuiteScript Debugger Interface  412

Chapter 53  Debugger Keyboard Shortcuts  418

Chapter 54  Debugger Glossary  419

Part 10  SuiteScript API

Chapter 55  SuiteScript API Overview  421
Chapter 71  Scriptable Sublists  873

Scriptable Sublists Overview  873
Access Sublist (contact roles)  875
Address Sublist  875
Adjustments Sublist  875
Apply Sublist  876
Assignees Sublist  876
Attendees Sublist  876
Billable Expenses Sublist  877
Billable Items Sublist  877
Billable Time Sublist  877
Bin Numbers Sublist  877
Company Contributions Sublist  878
Company Taxes Sublist  878
Competitors Sublist  878
Credits Sublist  878
Currencies Sublist  879
Custom Child Record Sublists  879
Deductions Sublist  892
Demand Plan Detail Sublist  892
Deposits Sublist  898
Direct Mail Sublist  899
Download Sublist  899
Earnings Sublist  899
E-mail Sublist  899
Employee Taxes Sublist  900
Escalate To Sublist  900
Expenses Sublist  900
Group Pricing Sublist  900
Item Fulfillment/Receipt Sublist  901
Items Sublist  902
Item Pricing Sublist  902
Lead Nurturing Sublist  903
Line Sublist  903
Members Sublist  903
Orders Sublist  903
Other Events Sublist  904
Partners Sublist  904
Pricing Sublist  905
Predecessors Sublist  921
Related Solutions Sublist  921
Resources Sublist  921
Sales Team Sublist  921
Shipping Sublist  922
Site Category  922
Time Tracking Sublist  922
Topics Sublist  923
Units  923
Vendors  923

Chapter 72  Record Initialization Defaults  924

Chapter 73  Transaction Type IDs  934
Part 1  Getting Started with SuiteScript
Chapter 1  SuiteScript - The Basics

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 end up referring to the SuiteBuilder Guide 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 Help, see SuiteBuilder Overview to learn more about SuiteBuilder.

Additional Topics

Once 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 and Subrecords 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.

- **Debugging Overview** - Describes how to use the SuiteScript Debugger to debug server-side SuiteScripts.

- **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, the Developer Portal, and the SuiteSource repository.
Chapter 2  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?

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 just 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.

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

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 and Sublists
5. Body fields
6. Sublist fields

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.
Chapter 3  Setting Up Your SuiteScript Environment

Environment Setup Overview

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 Your SuiteScript Development Environment (SuiteCloud IDE)

Configuring NetSuite for SuiteScript

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 Setting Up Your SuiteScript Development Environment (SuiteCloud IDE).

Enabling SuiteScript

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 check 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 Custom Forms in the SuiteBuilder (Customization) Guide.

Related Topics

- Showing Record and Field IDs in Your Account
- Setting Up Your SuiteScript Development Environment (SuiteCloud IDE)
- SuiteScript API Overview
Showing Record and Field IDs in Your Account

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. (Note that if you have enabled the Web Services you will also want to enable the Show Internal IDs preference.)

When referencing a record or field in SuiteScript, you will generally always 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 can also obtain record and field internal IDs by going to SuiteScript Supported Records in the NetSuite Help Center. This section lists all NetSuite records that currently support SuiteScript. Click on the record you are scripting against to see the record’s internal ID, as well as all of the field IDs associated with the record.

**To show internal NetSuite IDs:**

1. Go to Home > Set Preferences.
2. Click the General tab and then click the Show Internal IDs checkbox (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?**

Record IDs are unique IDs associated with a record at the time the record is created. Once 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).
If the *Show Internal IDs* preference is NOT enabled, or if the internal IDs are not displayed on a given 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 Adina Fitzpatrick customer record, the internal record ID (149) appears in the browser status bar.
How do I find a field’s internal ID?

Internal field IDs must be used when calling a field from a SuiteScript API. When the Show Internal IDs preference is enabled, internal IDs for each 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 Setup > Customization > CRM Fields. The ID column appears in the list of custom fields, as shown in the figure.

With Show Internal IDs enabled, you can also view internal field IDs 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 Name label. The internal ID for the Company Name field is `companyname`, which appears in the bottom-right corner of the Field Level Help window.

**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

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.

<table>
<thead>
<tr>
<th>Role</th>
<th>SuiteScript Access Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator</td>
<td>FULL</td>
</tr>
<tr>
<td>Full Access</td>
<td>FULL</td>
</tr>
<tr>
<td>Marketing Manager</td>
<td>FULL</td>
</tr>
<tr>
<td>Marketing Administrator</td>
<td>FULL</td>
</tr>
<tr>
<td>Sales Manager</td>
<td>FULL</td>
</tr>
<tr>
<td>Sales Administrator</td>
<td>FULL</td>
</tr>
<tr>
<td>Support Administrator</td>
<td>FULL</td>
</tr>
</tbody>
</table>

Important:

- 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.
Setting Up Your SuiteScript Development Environment (SuiteCloud IDE)

Before setting up your SuiteScript development environment, be sure you have completed the NetSuite configuration tasks described in Configuring NetSuite for SuiteScript.

Once your account has been configured accordingly, NetSuite recommends you read the SuiteCloud IDE Overview and follow the steps (in order) to set up your SuiteScript development environment:

1. Installing SuiteCloud IDE
2. Launching SuiteCloud IDE
3. Selecting a Workspace
4. Setting Up a Master Password
5. (Optional) Setting Up an Environment
6. Setting Up an Account
7. Importing Existing NetSuite Projects into SuiteCloud IDE
8. Setting SuiteCloud IDE Preferences

After setting up your SuiteCloud IDE, see Working with Your SuiteScript Development Environment (SuiteCloud IDE). If you choose to set up a SuiteScript development environment other than SuiteCloud IDE, see Working with IDEs Other Than SuiteCloud IDE.

Related Topics
- Environment Setup Overview
- Configuring NetSuite for SuiteScript
- Working with Your SuiteScript Development Environment (SuiteCloud IDE)
- Working with IDEs Other Than SuiteCloud IDE
- Setting Up Your SuiteScript Environment

SuiteCloud IDE Overview

SuiteCloud IDE is an Eclipse-based IDE that is packaged for NetSuite platform development. SuiteCloud IDE creates a development environment that offers:

- Code Completion for SuiteScript API and Internal IDs
- Upload and Download of Files from NetSuite File Cabinet
- Comparison of Files with NetSuite File Cabinet Version
• Validation of Internal IDs
• Integration with the SuiteScript Records Browser
• Management of Multiple NetSuite Accounts
• Support for JSDoc

For more information about how these features work for SuiteCloud IDE, see Usage Examples.

Related Topics
• Installing SuiteCloud IDE
• Launching SuiteCloud IDE
• Selecting a Workspace
• Setting Up a Master Password
• Setting Up an Environment
• Setting Up an Account
• Importing Existing NetSuite Projects into SuiteCloud IDE
• Setting SuiteCloud IDE Preferences
• Setting Up Your SuiteScript Development Environment (SuiteCloud IDE)

Installing SuiteCloud IDE

NetSuite recommends that you install and use the SuiteCloud IDE to write SuiteScript.

If you are not currently using SuiteCloud IDE, and you are installing it for the sole purpose of writing SuiteScript, NetSuite recommends that you install the current version of SuiteCloud IDE.

Important: Make sure that you have the latest Java software (Java Runtime Environment or JRE) installed, version 1.5 and above to be specific.

In order to properly run 64-bit versions of SuiteCloud IDE, make sure your operating system and Java versions are both 64-bit versions.

The current version of SuiteCloud IDE works only in accounts that do not have 2-factor authentication. Additionally, web services must be enabled for NetSuite accounts in order to use the SuiteCloud IDE.

ANT is not enabled in the current version of SuiteCloud IDE. For more information about adding ANT, see SuiteCloud IDE FAQ.

To install the SuiteCloud IDE, click the link below that is specific to your operating system.

• Windows 32-bit
• Windows 64-bit
• Linux 32-bit
• Linux 64-bit
• Mac OS 64 bit

Unpack the archive file you downloaded, and place it anywhere on your hard drive. You will see a directory named `eclipse`. Within that directory you will see an executable named `eclipse`.

For more information about launching SuiteCloud IDE, see Launching SuiteCloud IDE.

### Launching SuiteCloud IDE

When you launch your SuiteCloud IDE for the first time after installation, you will be asked to select a workspace. For more information, see Selecting a Workspace. Once you have a workspace selected, you will then be asked to accept the terms of the license agreement.

**Note:** The Usage Data Collector (UDC) is a default Eclipse feature that may appear from time to time (every five days by default) when you launch the SuiteCloud IDE. UDC gathers data on how you use the Eclipse platform. The gathered data is not stored in any NetSuite server at all. You may disable this feature by selecting **Turn UDC feature off** the first time it appears. For more information about how it works and its terms of use, see Usage Data Collector in the Eclipse documentation.

**To launch SuiteCloud IDE:**

1. Navigate to the folder location of your SuiteCloud IDE.
2. Locate the eclipse application.
3. Double-click eclipse.

   The NetSuite splash screen with progress bar appears and the SuiteCloud IDE window opens immediately after completion.
Selecting a Workspace

Upon launching SuiteCloud IDE after installation, you are prompted to select a workspace for your projects. Navigate to your desired workspace location and then click OK. You will then be asked to accept the terms of the license agreement.

Important: NetSuite recommends that you create a new workspace for SuiteCloud IDE instead of reusing an existing workspace. This is to avoid the possibility of carrying over incompatible settings from an old workspace.

You may check the Use this as the default and do not ask again box if you want your selected workspace location to become the default location.

Setting Up a Master Password

SuiteCloud IDE stores all of your account login information every time you add an account. With this, any succeeding upload or download operations will no longer prompt you to enter your account login information. To prevent unauthorized running of these operations, you need to set up a master password. Once you have it set up, you are only required to enter it once per session before any account-driven operations can be done.

Important: You need to set up a master password once for every workspace.

Your master password does not only protect all of your account login information, but it also saves you from entering different passwords for different accounts when you have multiple NetSuite accounts (different email addresses and passwords, to be specific).

To set up a master password, see the following procedures:

- Setting a Master Password
- Authenticating a Master Password
- Revoking a Master Password
- Changing a Master Password
### Setting a Master Password

After installing the SuiteCloud IDE for the first time, you need to set a master password for your workspace to protect all of your NetSuite account login information.

**To set a master password:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, go to NetSuite > Master Password > Set Master Password. The Set Master Password window opens.
3. Enter the following information: New Master Password and Re-enter New Master Password.
4. Click OK.

### Authenticating a Master Password

In order to upload a project, download a project, or interact with your NetSuite account, you need to authenticate your master password once per session until you close your SuiteCloud IDE.

**To authenticate a master password:**

1. Launch SuiteCloud IDE.
3. Enter the master password.
4. Click OK.

**Note:** You can also authenticate your master password from the NS Explorer pane or the editor area. Right-click on the pane or the editor area, and go to NetSuite > Authenticate Master Password.

You can also use the shortcut, Ctrl + Alt + A. For more information, see Shortcuts in SuiteCloud IDE Tips.
Revoking a Master Password

You can revoke your master password to avoid any mistaken or unauthorized upload, download, and interaction with your NetSuite account. For example, someone else needs to look at your SuiteCloud IDE editor to do a peer review of your code. You need to revoke your master password so that your NetSuite accounts are protected.

To revoke a master password in SuiteCloud IDE, go to NetSuite > Master Password > Revoke Master Password, and click OK when the Revoke Master Password window opens.

**Note:** Revoking a master password only logs you out. It does not delete the master password.

You can also revoke your master password from the NS Explorer pane or the editor area. Right-click on the pane or the editor area, and go to NetSuite > Revoke Master Password.

Changing a Master Password

You can change a master password.

**To change a master password:**

1. Launch SuiteCloud IDE.
3. Enter the following information: Old Master Password, New Master Password, and Re-enter New Master Password.
4. Click OK.
Setting Up an Environment

After setting up your master password, you can opt to set up a NetSuite environment where you want to run your scripts. The three predefined NetSuite environments in SuiteCloud IDE are:

- Production
- Release Preview
- Sandbox

To set up an environment, see the following procedures:

- Adding an Environment
- Modifying an Environment
- Removing an Environment

Adding an Environment

Use the following steps to add a NetSuite environment to the SuiteCloud IDE.

**To add an environment:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, go to NetSuite > Environments. The Environments window opens.
4. Enter the following information: Name and URL.
5. Click OK.
6. Click Close.
Modifying an Environment

Use the following steps to modify an existing NetSuite environment. You can change its name or point to its updated URL.

**To modify an environment:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, go to NetSuite > Environments. The Environments window opens.
3. In the Environment list, select an environment that you want to edit.
4. Click Edit. The Edit Environment window opens.
5. Modify the following information: Name and URL.
6. Click OK.
7. Click Close.

Removing an Environment

Use the following steps to remove a NetSuite environment from the SuiteCloud IDE. You can remove unused or unnecessary environment to have the environment options you only need at a given period of time.

**To remove an environment:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, go to NetSuite > Environments. The Environments window opens.
3. In the Environment list, select an environment that you want to remove.
4. Click Remove. The Remove Environment dialog opens.
5. Click OK.
6. Click Close.

**Setting Up an Account**

After setting up your master password, you need to set up the NetSuite accounts you will be writing scripts for. If you have set up environments in addition to the three predefined NetSuite environments, you need to configure account-specific details for these additional environments.

To set up an account, see the following procedures:

- Adding an Account
- Removing an Account

**Adding an Account**

Use the following steps to add an account. SuiteCloud IDE passes your account information to NetSuite during the IDE authentication process.

**Important:** Make sure to add NetSuite accounts that comply with NetSuite’s password requirements, especially in terms of the recommended special characters you can use. For more information, see NetSuite Password Requirements.

Whenever you change your NetSuite password, you will need to re-add your accounts using your latest NetSuite login credentials. However, SuiteCloud IDE will automatically prompt you to enter your latest NetSuite login credentials if you have updated your NetSuite login credentials but have not re-added your accounts in the IDE.
To add an account:

1. Launch SuiteCloud IDE.
3. Click Add. The Add Account(s) window opens.
4. Select an environment.
5. Enter the following information: Email and Password. **Important:** The password referred to here is the NetSuite password associated with your email.
6. Click Next. The Select Account(s) window opens.
7. Select the account you want to add. You may click Select All; or click Deselect All and then select what you need.
8. Click Finish. The Accounts window opens with your added accounts.
   Existing accounts are maintained. Newly-found accounts are added when you select and add them.
9. Click Close.

### Related Topics
- Removing an Account
- Setting Up an Account
- Launching SuiteCloud IDE

### Removing an Account

Use the following steps to remove an account. You can remove inactive or unused accounts that you no longer need.

**To remove an account:**

1. Launch SuiteCloud IDE.
3. In the Accounts list, select an account that you want to remove.
4. Click Remove. The Remove Account dialog opens.
5. Click OK.
6. Click Close.
Importing Existing NetSuite Projects into SuiteCloud IDE

If you already have existing projects in the NetSuite file cabinet, synchronize your file cabinet with SuiteCloud IDE by importing your projects into SuiteCloud IDE. Before doing so, ensure that all your project files are up-to-date in the NetSuite file cabinet.

To import existing NetSuite projects into SuiteCloud IDE:

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, authenticate your master password.
3. Right-click in the NS Explorer pane. The context menu appears.
4. In the context menu, go to NetSuite > Download Project. The Download Project window opens.
5. Select an account.
6. Select a role.
7. Click Get File List.
   
   Note: Hidden bundles, inactive files, and empty folders are excluded from the file list.
8. Wait for the progress bar to complete.
9. Select the projects that you want to import.
10. Select Use project name in file cabinet.
11. Select Use this project name.
12. Enter the project name that you want to use.
13. Click OK.
14. Check your project folder to verify that all files were imported.
Setting SuiteCloud IDE Preferences

To set SuiteCloud IDE preferences, see the following procedures:

- Setting SuiteCloud IDE General Preferences
- Setting SuiteCloud IDE Code Template Preferences
- Setting SuiteCloud IDE Validation Preferences
- Restoring SuiteCloud IDE Default Preferences

Related Topics

- Setting Up an Account
- Setting SuiteCloud IDE Preferences
- Setting Up Your SuiteScript Development Environment (SuiteCloud IDE)
- Launching SuiteCloud IDE
- Authenticating a Master Password

Setting SuiteCloud IDE General Preferences

You can set SuiteCloud IDE general preferences such as the display setting for the Start Page, the file backup mechanism, and the Internal ID Quote Character option. The preferred quote character will be automatically used when an internal ID is selected as a function parameter after pressing the shortcut, Ctrl + Space.

For example, if you set single quotes as the Internal ID Quote Character, typing `nlapiLoadRecord()` followed by the shortcut, Ctrl + Space, and then selecting an internal ID such as `salesorder` automatically results to `nlapiLoadRecord(‘salesorder’).

To set SuiteCloud IDE general preferences:

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, go to Window > Preferences. The Preferences window opens. The Preferences window defaults to the NetSuite Preferences view.
3. Under General Preferences, if you want to display the Start Page on startup, check Show Start Page on startup.
4. If you want to enable backing up of existing files during download, check Enable file backup (*.bak) for project or file downloads.
5. Select the Internal ID Quote Character that you want.
6. Click Apply.
7. Click OK.

**Related Topics**
- Setting SuiteCloud IDE Code Template Preferences
- Setting SuiteCloud IDE Validation Preferences
- Restoring SuiteCloud IDE Default Preferences
- Setting SuiteCloud IDE Preferences
-Launching SuiteCloud IDE

---

**Setting SuiteCloud IDE Code Template Preferences**

You can set SuiteCloud IDE code template preferences. You can create your own custom headers and function templates with the appropriate content and file name; and specify a folder to where these custom code templates for your team are located. If you do not specify a folder location, the default templates are used.

The following are the files you can customize code templates for:

- bundle_install_after_install.js
- bundle_install_after_update.js
- bundle_install_before_install.js
- bundle_install_before_update.js
- client_field_changed.js
- client_line_init.js
- client_page_init.js
- client_post_sourcing.js
- client_recalc.js
- client_save_record.js
- client_validate_delete.js
- client_validate_field.js
- client_validate_insert.js
- client_validate_line.js
- header.js
- mass_update.js
•portlet_form.js
•portlet_html.js
•portlet_links.js
•portlet_list.js
•RESTlet_delete.js
•RESTlet_get.js
•RESTlet_post.js
•RESTlet_put.js
•ssp.ss
•ssp.ssp
•scheduled.js
•suitelet.js
•templates.xml
•user_event_after_submit.js
•user_event_before_load.js
•user_event_before_submit.js
•workflow_action.js

Moreover, the table below shows the tokens you can use in creating your customized code template for the header.js file.

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>${filename}</td>
<td>Current filename</td>
</tr>
<tr>
<td>${project}</td>
<td>Current project</td>
</tr>
<tr>
<td>${year}</td>
<td>Current year</td>
</tr>
<tr>
<td>${date}</td>
<td>Current date</td>
</tr>
<tr>
<td>Note: The preferred date format can be specified in the Date Format field.</td>
<td></td>
</tr>
<tr>
<td>${author}</td>
<td>Current user name</td>
</tr>
<tr>
<td>Note: The current user name defaults to the system user name. The preferred user name can be specified in the User Name field.</td>
<td></td>
</tr>
</tbody>
</table>

For example, you can have the following text as code template for your header.js file:

```javascript
/**
 * Module Description
 *
 * Version Date Author Remarks
 * 1.00 ${date} ${author}
 * 
 */
```
To set SuiteCloud IDE code template preferences:

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, go to Window > Preferences. The Preferences window opens. The Preferences window defaults to the NetSuite Preferences view.
3. In the left-hand pane, navigate to NetSuite > Code Templates. The Code Templates Preferences view is shown.
4. Select the Code Templates Folder location.
5. Enter the following information: User Name and Date Format.
6. Click Apply.
7. Click OK.

Setting SuiteCloud IDE Validation Preferences

You can set SuiteCloud validation preferences and specify a text file as Ignore list. Validation for internal IDs is done whenever you save a file or you build or rebuild a project. The table below shows the three validation types you can use.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>This switches off the validation feature and the system performance is not impacted.</td>
</tr>
<tr>
<td>Fast</td>
<td>This determines context within the current file and the system performance is average.</td>
</tr>
<tr>
<td>Full</td>
<td>This determines context across different files but the system performance is slow.</td>
</tr>
</tbody>
</table>

**Note:** By default, the validation type is set to Fast.

To set SuiteCloud IDE validation preferences:

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, go to Window > Preferences. The Preferences window opens. The Preferences window defaults to the NetSuite Preferences view.
3. In the left-hand pane, navigate to NetSuite > Validation. The Validation Preferences view is shown.

4. Under Validation Preferences, select the Validation Type that you want.

5. Select the Ignore List text file.

   **Important:** In your text file, make sure that the items to be ignored are entered per line.

6. Click Apply.

7. Click OK.

### Related Topics

- Setting SuiteCloud IDE General Preferences
- Setting SuiteCloud IDE Code Template Preferences
- Restoring SuiteCloud IDE Default Preferences
- Setting SuiteCloud IDE Preferences
- Launching SuiteCloud IDE

### Restoring SuiteCloud IDE Default Preferences

Use the following steps to restore IDE default preferences.

**To restore SuiteCloud IDE default preferences:**

1. Launch SuiteCloud IDE.

2. In SuiteCloud IDE, go to Window > Preferences. The Preferences window opens. The Preferences window defaults to the NetSuite Preferences view.

3. In the left-hand pane, navigate to the specific preferences for NetSuite (such as general, code templates, and validation) that you want to restore default settings to.

4. Click Restore Defaults.

5. Click Apply.

6. Click OK.

### Related Topics

- Setting SuiteCloud IDE General Preferences
- Setting SuiteCloud IDE Code Template Preferences
- Setting SuiteCloud IDE Validation Preferences
- Setting SuiteCloud IDE Preferences
- Launching SuiteCloud IDE
Working with Your SuiteScript Development Environment (SuiteCloud IDE)

Before you can start working with SuiteCloud IDE, you need to familiarize yourself with the following:

- NetSuite Perspective Overview
- SuiteCloud IDE Tips
- Usage Examples

With your SuiteCloud IDE, you can do the following procedures:

- Working with SuiteScript Projects
- Working with SuiteScript Files
- Working with SSP Application Projects
- Changing Project Settings
- Converting a Project to a SuiteScript Project
- Converting a Project to an SSP Application Project
- Logging in to a Project Account
- Debugging a Project Account
- Launching the SuiteScript Records Browser
- Viewing Error Logs
- Resetting Your Master Password and Account Info

NetSuite Perspective Overview

The NetSuite Perspective is an added perspective in the Eclipse workbench for SuiteCloud IDE. It provides a set of functionality for manipulating your NetSuite projects and resources that are accessible through certain menus and toolbars.

The NetSuite perspective consists of an editor area and one or more views that you will use to accomplish your NetSuite projects.
Editor Area

The Editor area displays the source file editors you can use for your code. The NetSuite Perspective mainly uses the JavaScript and HTML editor for NetSuite projects, such as SuiteScript and SSP application projects.

For more information, see Editors in the Eclipse documentation.

NS Explorer View

The NetSuite Explorer view, referred to as the NS Explorer view, displays the hierarchical view of your NetSuite projects (their folders and files) and resources. It works similar to the Project Explorer view of Eclipse.

The table below shows the additional icons that can appear in the NS Explorer view.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Icon 1]</td>
<td>SuiteScript Project (open)</td>
</tr>
<tr>
<td>![Icon 2]</td>
<td>SSP Application Project (open)</td>
</tr>
</tbody>
</table>

In NS Explorer view, you can hover over the project folders to see the following details when available:
• Project Type
• NetSuite Account ID
• Company
• Email
• Environment
• File Cabinet Folder

For more information, see Project Explorer view in the Eclipse documentation.

**Outline View**

The Outline view displays the organizational structure of your source file that is active in the editor area. In general, it lists the structural elements of your source file by function.

For more information, see Outline view in the Eclipse documentation.

**Problems View**

The Problems view displays the errors and warnings found in your source file that is active in the editor area. The information found in this view corresponds to the problem markers shown in the marker bar of the editor area.

For more information, see Problems view in the Eclipse documentation.

**Documentation View**

The Documentation view displays the corresponding documentation or comment details you have for your code that is active in the editor area. This view displays only the documentation details for your functions. The same details appear in the popup when you click or hover over a function name.

**Progress View**

The Progress view displays the progress bar of all your upload or download actions.
Related Topics

- SuiteCloud IDE Tips
- Usage Examples
- Working with SuiteScript Projects
- Working with SuiteScript Files
- Working with SSP Application Projects
- Changing Project Settings
- Converting a Project to a SuiteScript Project
- Converting a Project to an SSP Application Project
- Logging in to a Project Account
- Debugging a Project Account
- Launching the SuiteScript Records Browser
- Viewing Error Logs
- Working with Your SuiteScript Development Environment (SuiteCloud IDE)

**SuiteCloud IDE Tips**

When working with SuiteCloud IDE, the following information will make your development experience easier:

- Shortcuts
- Validation Markers
- Guidelines

**Shortcuts**

The following table shows the different shortcuts you can use within the SuiteCloud IDE.
<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
</table>
| Ctrl + Space | This is an existing Eclipse shortcut. This is used to get a dropdown of all possible code completion options available in a particular context. **Note:**  
- In cases wherein a number of record types are involved, you can precede this shortcut with the shortcut, `<x>` + Space + `<y>`, to specify the record context you need.  
- In cases wherein a number of sublists of a record are involved, you can precede this shortcut with the shortcut, `<x>` + Space + `<y>` + Space + `<z>`, to specify the record-sublist context you need. |
| `<x>` + Space | This is used to get all possible code completion options for a record; where `<x>` is your filter against record types. For example, you type ‘s’ + Space if you want to get all possible code completion options for salesorder fields. This is used in conjunction with the shortcut, Ctrl + Space. For more information, see Usage Examples. |
| `<x>` + Space + `<y>` | This is used to get all possible code completion options for a field in a particular record context; where `<x>` is your filter against record types and `<y>` is your filter against fields of a record type. For example, you type ‘s’ + Space + ‘a’ if you want to get all possible code completion options for salesorder fields starting with the letter ‘a’. **Note:** For internal ID code completion options, append an exclamation point ‘!’ to override and ignore the context when “No Proposal” appears for your given prefix filter. This is used in conjunction with the shortcut, Ctrl + Space. For more information, see Usage Examples. |
| `<x>` + Space + `<y>` + Space + `<z>` | This is used to get all possible code completion options for a sublist field in a particular record-sublist context; where `<x>` is your filter against record types, `<y>` is your filter against sublists of a record type, and `<z>` is your filter against fields of a sublist. For example, you type ‘s’ + Space + ‘i’ + Space + ‘a’ if you want to get all possible code completion options for salesorder-item sublist fields starting with the letter ‘a’. This is used in conjunction with the shortcut, Ctrl + Space. For more information, see Usage Examples. |
### Validation Markers

The table below shows the Editor area validation markers that were augmented for SuiteCloud IDE.

<table>
<thead>
<tr>
<th>Validation Marker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warning</td>
<td>Augmented to indicate the source location of NetSuite-related code completion warnings</td>
</tr>
<tr>
<td>Error</td>
<td>Augmented to indicate the source location of NetSuite-related syntax or compilation errors</td>
</tr>
</tbody>
</table>

### Shortcut Description

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;XYZ&gt;</td>
<td>This is an existing Eclipse shortcut. This is used to get code completion options using camel case patterns/filters; where xYZ is your camel case filter against methods and variables. For example, you type 'nLR' if you want to get code completion options for methods using the camel case pattern such as nlapiLoadRecord. This is used in conjunction with the shortcut, Ctrl + Space. For more information, see Usage Examples.</td>
</tr>
<tr>
<td>Ctrl + Alt + A</td>
<td>This is used to authenticate your master password.</td>
</tr>
<tr>
<td>Ctrl + U</td>
<td>This is used to upload file(s) from the editor area.</td>
</tr>
<tr>
<td>Ctrl + B</td>
<td>This is used to log in to a project account from the editor area.</td>
</tr>
<tr>
<td>Tab</td>
<td>This is used to globally change function names, record names, variables, and parameters. <strong>Note:</strong> This is applicable only for newly created SuiteScript files.</td>
</tr>
<tr>
<td>Esc</td>
<td>This is used to exit the mode set in conjunction with the shortcut, Tab.</td>
</tr>
<tr>
<td>Ctrl + Click &lt;internal ID&gt;</td>
<td>This is used to launch the Record Browser specific to the internal ID from the editor.</td>
</tr>
<tr>
<td>Type /** and press Enter</td>
<td>This is an existing Eclipse shortcut. This is used to enter a standard JSDoc comment.</td>
</tr>
<tr>
<td>Alt + Up or Alt + Down</td>
<td>This is an existing Eclipse shortcut. This is used to move the current line/selection up or down.</td>
</tr>
<tr>
<td>Ctrl + Alt + Up or Ctrl + Alt + Down</td>
<td>This is an existing Eclipse shortcut. This is used to duplicate the current line/selection, and place the clone above (Up) or below (Down) the current line/selection.</td>
</tr>
<tr>
<td>Alt + Shift + A</td>
<td>This is an existing Eclipse shortcut. This is used to toggle block selection mode (formerly column mode). Block selection mode allows you to select a rectangle of text and modify the highlighted text altogether instead of selecting text and modifying them a line at a time.</td>
</tr>
</tbody>
</table>
For more information, see *Markers* in the Eclipse documentation.

**Guidelines**

When working with SuiteCloud IDE, you need to take note of the following guidelines:

- **Use top-down coding method.** Make sure to write your code from top to bottom. The code preceding the cursor should always be completely valid or syntactically correct. For code completion to work, there should be no undeclared variables. For more information about the different ways to maximize code completion, see [SuiteCloud IDE FAQ](https://www.netsuite.com/what-is-suitecloud-ide).

- **Use camel case filters.** You can use filters in camel case format inherent to Eclipse to maximize code completion. If you want to display `nlapiLoadRecord` as an option, enter ‘nLR’ followed by the code completion shortcut (Ctrl + Space). For more information about other ways to maximize code completion, see [SuiteCloud IDE FAQ](https://www.netsuite.com/what-is-suitecloud-ide).

  **Note:** Make sure that the Show camel case matches preference is enabled. To verify if it is enabled, go to Window > Preferences > JavaScript > Editor > Content Assist in SuiteCloud IDE.

- **Pass variables to functions.** When variables (simple variables and not object properties) are passed to a SuiteScript function, code completion will work. This approach is applicable to all parameters that have code completion. For more information about other ways to maximize code completion, see [SuiteCloud IDE FAQ](https://www.netsuite.com/what-is-suitecloud-ide).

- **Use prefix filters.** When working with multiple records, use prefix filters to specify the record context you need for code completion. The prefix filter shortcuts are: `<x> + Space`, `<x> + Space + <y>`, and `<x> + Space + <y> + Space + <z>`. Use them in conjunction with the code completion shortcut (Ctrl + Space). For more information about prefix filter shortcuts, see [Shortcuts](https://www.netsuite.com/what-is-suitecloud-ide). For more information about the different ways to maximize code completion, see [SuiteCloud IDE FAQ](https://www.netsuite.com/what-is-suitecloud-ide).

- **Use standard JSDoc.** Based on JavaDoc, you can use the standard format of a JSDoc comment/tags for a function (including variable type declaration) for documenting the parameter (@param) and return (@returns) types to maximize code completion. For more information about the shortcut for a standard JSDoc comment, see [Shortcuts](https://www.netsuite.com/what-is-suitecloud-ide). Additionally, you can specify the variable type (@type) for variable declarations. For more information about the different ways to maximize code completion, see [SuiteCloud IDE FAQ](https://www.netsuite.com/what-is-suitecloud-ide).

- **Use custom NetSuite JSDoc.** You can use the custom NetSuite JSDoc tags, `@appliedtorecord` and `@record`, to augment code completion. The `@appliedtorecord` tag corresponds to the *Applied To* record in your NetSuite script deployment and is only applicable for client and user event script functions that rely on the specified *Applied To* record. The `@record` tag can be used in conjunction with standard JSDoc tags such as `@param`, `@returns`, and `@type` to specify the needed record context. For more information about the different ways to maximize code completion, see [SuiteCloud IDE FAQ](https://www.netsuite.com/what-is-suitecloud-ide).
**Activate code completion automatically.** By default, code completion is activated in SuiteCloud IDE by pressing the shortcut, Ctrl + Space. However, you can automatically trigger code completion through the **Enable auto activation** preference. To enable auto activation, go to Window > Preferences > JavaScript > Editor > Content Assist in SuiteCloud IDE. Make sure that the preference is checked and that the following fields are set accordingly:

- **Auto activation delay**: 0
- **Auto action triggers for JavaScript**: .nfo

For more information about auto activation and the different ways to maximize code completion, see SuiteCloud IDE FAQ.

**Close unused projects.** For optimum performance, make sure that the only open project in your SuiteCloud IDE is the project you are currently working on. Always close unused projects. For more information about closing a project, see Closing projects in the Eclipse documentation.

**Use unique names.** Make sure that you use unique names for global functions. Additionally, use unique function names and parameter names within a single object literal.

**Increase available memory.** You can increase the memory available for SuiteCloud IDE to allow a maximum amount instead of just the default allocation. However, the maximum heap memory depends on your SuiteCloud IDE version (32-bit or 64-bit, with the latter having a higher maximum), operating system, and Java virtual machine (JVM). For more information, see SuiteCloud IDE FAQ.

**Specify preferred JVM.** You can specify your preferred JVM when there are multiple copies available. For more information about working with multiple copies of JVM, see SuiteCloud IDE FAQ.
Usage Examples

Code completion works by examining the source code around the cursor. When the code completion shortcut (Ctrl + Space) is pressed, the system displays a popup with the appropriate text options to insert at the cursor. This popup may include record types, field names, or other information, depending on where the caret is positioned.

To learn the basics of code completion hands-on, download the DemoProject files included with the SuiteCloud IDE here.

Important: After downloading the DemoProject files, make sure to add the files to a SuiteScript project in your SuiteCloud IDE. For more information, see Importing files in the Eclipse documentation.

If you have no existing SuiteScript projects yet in your SuiteCloud IDE, you need to create one before you add the DemoProject files. For more information, see Creating a SuiteScript Project.

For information about the different ways to maximize code completion, see SuiteCloud IDE FAQ.
Working with SuiteScript Projects

A SuiteScript project is a type of project that is based on the JavaScript project but with the NetSuite SuiteScript library automatically added to enable the auto completion and content assist features.

With SuiteScript projects, you can do the following procedures:

- Creating a SuiteScript Project
- Uploading Files in a SuiteScript Project
- Downloading Files in a SuiteScript Project
Creating a SuiteScript Project

Use the following steps to create a SuiteScript project in the SuiteCloud IDE.

To create a SuiteScript project:

1. Launch SuiteCloud IDE.
3. Enter a project name.
4. Select SuiteScript Project as project type.
5. Select Use default location.
6. If you do not want to use the default location, deselect Use default location and navigate to your desired location.
7. Click Finish.
**Uploading Files in a SuiteScript Project**

You can upload files in a SuiteScript project to the NetSuite file cabinet. If you are uploading files from a project that does not exist yet in the NetSuite file cabinet, a folder with the same name as the SuiteScript project where the files belong to will be created.

**Note:** You can select only one project to upload files from. If you are uploading files for a selected project that is a closed project in the SuiteCloud IDE, you need to re-open the project before you can download the files. For more information about closing and re-opening projects, see *Closing projects* in the Eclipse documentation.

**To upload files in a SuiteScript project:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, right-click a project in the NS Explorer pane. The context menu appears.
3. In the context menu, go to NetSuite > Upload File(s) to Project. The Upload File(s) to Project window opens.
4. Select an account.
5. Select a role.
6. Select a file cabinet folder.
7. Click OK.

**Related Topics**
- Creating a SuiteScript Project
- Downloading Files in a SuiteScript Project
- Working with SuiteScript Projects
- Launching SuiteCloud IDE

**Downloading Files in a SuiteScript Project**

You can download files in a SuiteScript project from the NetSuite file cabinet. If you are downloading files that already exist in your SuiteCloud IDE, the existing files will be backed up prior to download and will have the extension, `.bak`.

**Note:** You can choose to download files to their respective project folders using *Use project name in file cabinet* or to a single target project folder using *Use this project name*.

If you are downloading files for a project that is a closed project in the SuiteCloud IDE, you need to re-open the project first before you can download the files. For more information about closing and re-opening projects, see *Closing projects* in the Eclipse documentation.
To download files in a SuiteScript project:

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, right-click a project in the NS Explorer pane. The context menu appears.
3. In the context menu, go to NetSuite > Download File(s) to Project. The Download File(s) to Project window opens.
4. Select an account.
5. Select a role.
6. Click Get File List.
   
   **Note:** Hidden bundles, inactive files, and empty folders are excluded from the file list.
7. Wait for the progress bar to complete.
8. Select the files that you want to download.
9. Select Use project name in file cabinet or select Use this project name and enter the project name that you want to use.
10. Click OK.

### Related Topics
- Creating a SuiteScript Project
- Uploading Files in a SuiteScript Project
- Working with SuiteScript Projects
- Launching SuiteCloud IDE

### Working with SuiteScript Files

With SuiteScript files, you can do the following procedures:

- Creating a SuiteScript File
- Uploading a SuiteScript File
- Downloading a SuiteScript File
- Comparing a SuiteScript File with File Cabinet Copy
Creating a SuiteScript File

Use the following steps to create a SuiteScript file in the SuiteCloud IDE.

To create a SuiteScript file:

1. Launch SuiteCloud IDE.
3. Select a script type.
4. Enter or select a parent folder.
5. Enter a script filename.
6. Click Finish.

Note: You can also create a SuiteScript file through the toolbar. Click the dropdown arrow beside the NetSuite icon and select SuiteScript File.
Uploading a SuiteScript File

You can upload a SuiteScript file to the NetSuite file cabinet from a SuiteScript project in your SuiteCloud IDE. If you are uploading a SuiteScript file from a project that does not exist yet in the NetSuite file cabinet, a folder with the same name as the SuiteScript project where the file belongs to will be created.

**Note:** Also, you can upload multiple files but these files must belong to a single project at a time.

**To upload a SuiteScript file:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, right-click a file in the NS Explorer pane. The context menu appears.
3. In the context menu, go to NetSuite > Upload Selected File(s). The Upload Selected File(s) window opens.
4. Select an account.
5. Select a role.
6. Select a file cabinet folder.
7. Click OK.

**Note:** You can upload a file directly from your Editor area. Right-click anywhere on the area and then go to NetSuite > Upload File in Editor. You can also use the shortcut, Ctrl + U. For more information, see Shortcuts in SuiteCloud IDE Tips.
Downloading a SuiteScript File

You can download a SuiteScript file from the NetSuite file cabinet to a specified SuiteScript project in your SuiteCloud IDE. If a SuiteScript file already exists, the existing file will be backed up prior to download and will have the extension, .bak.

**Note:** Also, you can download multiple files but these files must belong to a single project at a time.

**To download a SuiteScript file:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, right-click a file in the NS Explorer pane. The context menu appears.
3. In the context menu, go to NetSuite > Download Selected File(s). The Download Selected File(s) window opens.
4. Select an account.
5. Select a role.
6. Select a file cabinet folder.
7. Click OK.

**Note:** You can download a file directly from your Editor area. Right-click anywhere on the area and then go to NetSuite > Download File in Editor.

Comparing a SuiteScript File with File Cabinet Copy

Use the following steps to compare your SuiteScript file with its file cabinet copy.

**To compare a SuiteScript file with file cabinet copy:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, right-click a file in the NS Explorer pane. The context menu appears.
3. In the context menu, go to NetSuite > Compare Selected File with File Cabinet Copy.
Note: You can compare a file with its file cabinet copy directly from your Editor area. Right-click anywhere on the area and then go to NetSuite > Compare Selected File with File Cabinet Copy.

### Related Topics
- Creating a SuiteScript File
- Uploading a SuiteScript File
- Downloading a SuiteScript File
- Working with SuiteScript Files
- Launching SuiteCloud IDE

---

**Working with SSP Application Projects**

SSP application projects are packaged NetSuite Web store customization projects that you can use to fully customize key NetSuite Web store touch points, such as login, cart, and checkout. You can use familiar HTML and SuiteScript, and even other Ecommerce platforms for these customizations.

For more information, see [Using SuiteScript Server Pages for Web Store Customizations](#).

Important: The current version of SuiteCloud IDE only supports code completion for .ss files. Code completion for .ssp files is not yet supported, and the editor utilized for these .ssp files is the default HTML editor of Eclipse.

With SSP application projects, you can do the following procedures:

- Creating an SSP Application Project
- Uploading Files in an SSP Application Project
- Downloading Files in an SSP Application Project
Creating an SSP Application Project

Use the following steps to create an SSP application project in the SuiteCloud IDE.

To create an SSP application project:

1. Launch SuiteCloud IDE.
3. Enter a project name.
4. Select SSP Application Project as project type.
5. Select Use default location.
6. If you do not want to use the default location, deselect Use default location and navigate to your desired location.
7. Click Finish.
Uploading Files in an SSP Application Project

You can upload files in an SSP application project to the NetSuite file cabinet. If you are uploading files from a project that does not exist yet in the NetSuite file cabinet, a folder with the same name as the SSP application project where the files belong to will be created.

**Note:** You can select only one project to upload files from. If you are uploading files for a selected project that is a closed project in the SuiteCloud IDE, you need to re-open the project before you can download the files. For more information about closing and re-opening projects, see *Closing projects* in the Eclipse documentation.

It is recommended that you already have an application record created for your SSP application project. This allows you to easily select the directory structure you will need in the File Cabinet Folder field when you upload your project files. For more information about creating an application record, see *Creating an SSP Application Record*.

**To upload files in an SSP application project:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, right-click a project in the NS Explorer pane. The context menu appears.
3. In the context menu, go to NetSuite > Upload File(s) to Project. The Upload File(s) to Project window opens.
4. Select an account.
5. Select a role.
6. Select a file cabinet folder.
7. Click OK.

**Related Topics**
- Creating an SSP Application Project
- Downloading Files in an SSP Application Project
- Working with SSP Application Projects
- Launching SuiteCloud IDE
- Creating an SSP Application Record

Downloading Files in an SSP Application Project

You can download files in an SSP application project from the NetSuite file cabinet. If you are downloading files that already exist in your SuiteCloud IDE, the existing files will be backed up prior to download and will have the extension, *.bak*.
Note: You can choose to download files to their respective project folders using **Use project name in file cabinet** or to a single target project folder using **Use this project name**.

If you are downloading files for a project that is a closed project in the SuiteCloud IDE, you need to re-open the project first before you can download the files. For more information about closing and re-opening projects, see *Closing projects* in the Eclipse documentation.

**To download files in an SSP application project:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, right-click a project in the NS Explorer pane. The context menu appears.
3. In the context menu, go to NetSuite > Download File(s) to Project. The Download File(s) to Project window opens.
4. Select an account.
5. Select a role.
6. Click **Get File List**.

   **Note:** Hidden bundles, inactive files, and empty folders are excluded from the file list.

7. Wait for the progress bar to complete.
8. Select the files that you want to download.
9. Select **Use project name in file cabinet** or select **Use this project name** and enter the project name that you want to use.
10. Click OK.

**Changing Project Settings**

You can change the settings of a project such as accounts, roles, and file cabinet folders.

**To change project settings:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, right-click a project in the NS Explorer pane. The context menu appears.
3. In the context menu, go to NetSuite > Change Project Settings. The Change Project Settings window opens.

4. Select an account.

5. Select a role.

6. Select a file cabinet folder.

7. Click OK.

---

**Converting a Project to a SuiteScript Project**

You can convert any project to a SuiteScript project.

**Important:** Only the type and the associated NetSuite libraries of a project are changed.

**To convert a project to a SuiteScript project:**

1. Launch SuiteCloud IDE.

2. In SuiteCloud IDE, right-click a non-SuiteScript project in the NS Explorer pane. The context menu appears.

3. In the context menu, go to NetSuite > Convert to SuiteScript Project. The project is converted into a SuiteScript project.
Converting a Project to an SSP Application Project

You can convert a project to an SSP application project.

**Important:** Only the type and the associated NetSuite libraries of a project are changed.

**To convert a project to an SSP application project:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, right-click a non-SSP application project in the NS Explorer pane. The context menu appears.
3. In the context menu, go to NetSuite > Convert to SSP Application Project. The project is converted into an SSP application project.
Logging in to a Project Account

You can log in to a project account directly from the SuiteCloud IDE.

**Important:** The current version of SuiteCloud IDE does not support logging in to a project account for some Linux distributions (or flavors) depending on the desktop environment used. However, it works well for Windows and Mac OS.

**To log in to a project account:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, right-click a file or a project in the NS Explorer pane. The context menu appears.
3. In the context menu, go to NetSuite > Login Project Account. A browser opens with your NetSuite account loaded.
Debugging a Project Account

You can debug a project account using the SuiteScript Debugger directly from the SuiteCloud IDE.

**Important:** The current version of SuiteCloud IDE does not support debugging a project account for some Linux distributions (or flavors) depending on the desktop environment used. However, it works well for Windows and MacOS.

For more information about using the SuiteScript Debugger, see [Debugging SuiteScript](#).

**To debug a project account:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, right-click a file or a project in the NS Explorer pane. The context menu appears.
3. In the context menu, go to NetSuite > Debug Project Account. A browser opens with your NetSuite account loaded in debug mode.
You can launch the SuiteScript Records Browser from the SuiteCloud IDE. The SuiteScript Records Browser provides a Web-based view of all records, fields, sublists, search joins, search filters, search columns, and record transformations that are supported in SuiteScript.

For more information, see Using the SuiteScript Records Browser.

To launch Record Browser:

1. Launch SuiteCloud IDE.
3. In the Contents pane, navigate to NetSuite User Guide > Record Browser. The SuiteScript Records Browser is shown.

Note: You can launch the SuiteScript Records Browser from the editor by pressing Ctrl + mouse over an internal id. For more information, see Shortcuts in SuiteCloud IDE Tips.
Viewing Error Logs

You can view error logs to help you diagnose or troubleshoot the SuiteCloud IDE. Viewing error logs is particularly useful for Support teams.

**Note:** Another way to view error logs in your SuiteCloud IDE is to go to NetSuite > TroubleShoot > View Log File.

**To view error logs:**

1. Launch SuiteCloud IDE.
   
   **Important:** You can see the version of SuiteCloud IDE from this window. The version number is necessary when filing issues for SuiteCloud IDE.
3. Click Installation Details. The SuiteCloud IDE Installation Details window opens.
4. Click Configuration. The Configuration tab is shown.
5. Click View Error Log. A browser opens with the error log details.
If you forgot what your master password is, you need to reset your master password and account info. For security reasons, there is no way for you to retrieve your master password. This is by design.

**To reset your master password and account info:**

1. Launch SuiteCloud IDE.
2. In SuiteCloud IDE, go to NetSuite > Troubleshoot > Reset Master Password and Account Info. A dialog appears warning you that your SuiteCloud IDE will be restarted and that your master password, along with all of your account information, will be deleted.
3. Click OK. SuiteCloud IDE restarts with your workspace still intact but your master password and account info deleted.
4. Set up your master password and account info. For more information, see Setting a Master Password and Adding an Account.
Related Topics

- NetSuite Perspective Overview
- SuiteCloud IDE Tips
- Usage Examples
- Working with SuiteScript Projects
- Working with SuiteScript Files
- Working with SSP Application Projects
- Changing Project Settings
- Converting a Project to a SuiteScript Project
- Converting a Project to an SSP Application Project
- Logging in to a Project Account
- Debugging a Project Account
- Launching the SuiteScript Records Browser
- Working with Your SuiteScript Development Environment (SuiteCloud IDE)
- Launching SuiteCloud IDE
Working with IDEs Other Than SuiteCloud IDE

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 Setting Up Your SuiteScript Development Environment (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).
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

Once 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.

**Note:**

- 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 in the lower-left corner of your screen (see figure).
3. 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 (see figure).
A file with the same name already exists in the selected folder. Click OK to overwrite the existing file.
Part 2  Running a Script in NetSuite
Chapter 4  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 Setting Up Your SuiteScript Development Environment (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 simple 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:

```javascript
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.
Chapter 5  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 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.

Once 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.)
Chapter 6  Step 3: Attach Script to Form

Form-level client scripts are simply “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 your client script has been uploaded into NetSuite. (See Step 2: Add Script to NetSuite File Cabinet.)

2. Go to the desired custom form in NetSuite.
   Form-level client scripts can only be attached to custom entry forms, custom transaction forms, and custom online forms. Click Setup > Customization > [Form].

3. Next, click Customize next to the desired custom form, or click Customize next to an existing standard form in order to create a new custom form based 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 Custom Code tab on the form, select your script file and, if necessary, the library file to associate with the current form (see figure).
   The library script file should contain any commonly used functions. The SuiteScript file should contain functions specific to the current form.

5. Based on the APIs used in your SuiteScript or library files, define which functions should be called on which client events. If you are unsure of which actions trigger each client event, see Client Event Functions. 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 form called Wolfe Electronics Sales Order. 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 will run whenever a NetSuite user navigates to and loads sales order record. This figure indicates that when a sales order loads, three different functions will 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: Be sure to 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.

Important: When adding a client script file to an online form, select the Available Without Login check box to ensure that the script will run on the form. If this check box is not selected, the user will still have access to the form, however, the client script will not run.

The Available Without Login check box appears in the File popup window when adding a new file to the Script File field (by clicking the + icon). (See previous figure.)
If you are selecting your .js file from a list of files in the NetSuite file cabinet, click Edit next to the script file, and then select **Available Without Login** on the script record.

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 Functions**.

If you have created a form-level client script, you do not need to go to **Step 4: Create Script Record** or **Step 5: Define Script Deployment**. You are done!!!!!
Chapter 7  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.
4. Specify the main executing function within the file.

**Important:** See Steps for Creating a Script Record for more detailed information.
Step 4: Create Script Record

**New Script**
- **Name**: Sending follow up email
- **ID**: ue_follow_up_email
- **Owner**: K. Wolfe
- **Script File**: followUpEmail.js
- **Before Load Function**: sendNotification
- **Before Submit Function**: sendNotification
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 Setup > Customization > Scripts > New.
   
   Note that after creating your script record, you can later access the record by going to Setup > Customization > Scripts to see a list view of all Script records.

2. Select the script type (see figure).

   ![Select Type](image)

   **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.

3. 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.)

4. 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.

5. If you are creating a Script record for a Portlet script, select the portlet type from the Portlet Type drop-down list.
   
   For information on portlet types, see Portlet Scripts in the NetSuite Help Center.

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:
   
   **a.** 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 will appear 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 the SuiteScript plug-in for Eclipse.
   
   **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.

   **b.** (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.
   
   Also be aware that the system reads your library files in the order they appear on the Library Script File tab. For example, if your first library file references the second library file, an error will be thrown, since the first library file is loaded before the second.

   **c.** 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.

![Image of script creation interface](image)

• **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.

![Image of script creation interface with bundle deployment events](image)

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** tab, define which individual(s) will be notified if script errors occur.
   - 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.

12. From the Save button:

   a. 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 will not execute until they are deployed.

   b. 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, simply click Save.

   c. 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 that allows you to 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 simply the Script record's internalId.

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 SuiteBundler Overview 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.

![Change ID button on Script Deployment page](image.png)

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.

![Change Script ID page](image.png)

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

**Related Topics**

- Step 4: Create Script Record
- Step 5: Define Script Deployment
Chapter 8  Step 5: Define Script Deployment

Once 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 such an area. 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 Setup > Customization > 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 once the script is executed. See Setting Script Execution Log Levels.

8. In the Execute as Admin check box, select whether you want the script to execute using administrative privileges, regardless of the permissions of the currently logged in user. See Executing Scripts as Admin.

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 *Using the Script Execution Log*.

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 once the Script Deployment page is saved.

Whether creating a custom ID or accepting a system-generated ID, once 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 given script definition.

![Script Deployments](image)

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.
Chapter 9  Viewing Script Deployments

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.

**Related Topics**

- Running Scripts in NetSuite Overview
- Step 5: Define Script Deployment
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 inactivate specific deployments.

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

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.
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
Related Topics

- User Event Scripts
- Client Scripts
- Viewing Script Deployments
- Step 5: Define Script Deployment
Chapter 11 Using the Script Execution Log

Script execution details are logged on the **Execution Log** tab that appears on the Script Deployment page and in the SuiteScript Debugger.

**Important:** If you are using the SuiteScript Debugger to debug a script, all logging details will appear on the Execution Log tab in the Debugger. These details will **not also** appear on the Execution Log tab on the Script Deployment page for the same script. To have logging details appear on the Execution Log tab of the Script Deployment page, you must deploy the script.

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

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

Be aware that the **View** drop-down field in the Execution Log defaults to “Default Script Notes View.” This default shows only the current day’s script executions. If you want to see script executions for days other than the current day, click the Customize View button.

Within the Customize View window, you can specify values such as script execution dates, details, names, and script types (see figure). In the Search Title field you can provide a custom name for this view.
Important: When customizing your Execution Log view, be aware that NetSuite purges user-generated execution log details every 30 days. System errors that appear on the Execution Log are purged every 60 days. Although the UI allows you to set date values as far back as two years (meaning you can see the last two years of script execution details), you will actually only be able to see up to 30 days of user-generated details (captured by nlapiLogExecution) and 60 days for system errors.

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

Related Topics

- Setting Script Execution Log Levels
- Debugging SuiteScript
- nlapiLogExecution(type, title, details)
Part 3  Scripting Records, Fields, Forms, and Sublists
Chapter 12 Working with Records and Subrecords in SuiteScript

The following topics are covered in this section:

- Working with Records in SuiteScript
- How Records are Processed in Scripting
- Working with Subrecords in SuiteScript
- Working with Records in Dynamic Mode

**Note:** For a list of SuiteScript supported records, see SuiteScript Supported Records in the NetSuite Help Center.

**Working with Records in SuiteScript**

The SuiteScript API includes several Record APIs that interact with the entire NetSuite record object. When you work with Record APIs, you are doing things such as creating, deleting, copying, or loading all elements of a record.

Whether you are working with standard NetSuite records (for example, Sale Order, Invoice, Customer, Vendor) or custom records you have created using SuiteBuilder, you will use all the same Record APIs to interact with the record object.

**Note:** For a list of SuiteScript supported records, see SuiteScript Supported Records in the NetSuite Help Center.

The following figure shows a Sale Order record. The elements of this record, as well as most other records in NetSuite, include fields, subtabs, sublists, sublist fields, and buttons.
How Records are Processed in Scripting

When using SuiteScript to interact with records, NetSuite will process your data and execute all core NetSuite business logic once the data is submitted.

As a SuiteScript developer, note the following when working with records:

- You can write your SuiteScript code without having to “reverse engineer” NetSuite logic. When you submit the record, the validation, sourcing, and recalculation logic that automatically occurs in the UI will also occur when you submit a record in SuiteScript. This is considered the standard mode of processing.

- When scripting in standard mode (as opposed to dynamic mode), you do not have to write your code in a way that references fields in the order they are referenced in the UI. (To learn more about dynamic mode scripting, see Working with Records in Dynamic Mode.)
When scripting in **standard** mode (as opposed to **dynamic** mode), you will have to submit and then reload a record to know the values for calculated fields. For example, you will need to submit and then reload a record to know how much an item will cost after NetSuite applies tax and shipping amounts to the data submitted.

### Working with Subrecords in SuiteScript

This section contains following topics. If you are not familiar with subrecords, it is suggested that you read these topics in order:

- 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

**Important:** Subrecords are used in the context of the Advanced Bin / Numbered Inventory Management feature. Currently, the only supported subrecord in NetSuite is the Inventory Detail subrecord. For scripting examples that show how to use Subrecord APIs in the context of this feature, see Using SuiteScript with Advanced Bin / Numbered Inventory Management.

### 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.

**Important:** Subrecords are used in the context of the Advanced Bin / Numbered Inventory Management feature. Currently, the only supported subrecord in NetSuite is the Inventory Detail subrecord.

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 Bin Item.

In this case the parent record is still the Bill record, even though the subrecord tracks inventory details related to the Lot Bin 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 Setup > Customization > Entry Forms. Currently the only supported subrecord type is Inventory Detail, which is associated with the Advanced Bin / Numbered Inventory Management feature. Therefore, 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`).

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 currently supported in client scripts.

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 Inventory Details (inventorydetail) body field on the parent record. To create a subrecord of inventory details related to 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 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.

Related Topics
- What is a Subrecord?
- Creating and Accessing Subrecords from a Sublist Field
- Guidelines for Working with Subrecords in SuiteScript

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 subrecord for the Lot Bin Item is being edited.
For code samples showing how “sublist field” subrecord APIs are used, see Using SuiteScript with Advanced Bin / Numbered Inventory Management.

**Related Topics**
- What is a Subrecord?
- Creating and Accessing Subrecords from a Body Field
- Guidelines for Working with Subrecords in SuiteScript

**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, linenum)` - use if selecting an existing line on the sublist
• `setCurrentLineItemValue(group, name, value)` - use to set the values on a line
• `commitLineItem(group)` - use to commit the line

**Important:** The `napiSetLineItemValue(...)` 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.

```javascript
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.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.

```javascript
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.

```javascript
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.

### Related Topics
- What is a Subrecord?
- Saving Subrecords Using SuiteScript
- Guidelines for Working with Subrecords in SuiteScript

### 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. Once 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.

```javascript
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, linenumber)
- nlapiGetLineItemText(type, fldname, linenumber)
- nlapiFindLineItemValue(type, fldname, val)
- nlapiGetCurrentLineItemText(type, fldname)
- nlapiGetCurrentLineItemValue(type, fldname)
- 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.

### Related Topics
- What is a Subrecord?
- Creating and Accessing Subrecords from a Body Field
- Creating and Accessing Subrecords from a Sublist Field
- Setting Values on Subrecord Sublists
Working with Records in Dynamic Mode

When creating, copying, loading, or transforming records in SuiteScript, you have the choice of working with records in dynamic mode. When scripting in dynamic mode, you are working with a record in a way that emulates the behaviors of the UI. For example, in the UI, if you create a new sales order, select a custom form, and specify a customer, various field values throughout the new record are sourced (see figure).

When you programmatically create a new sales order in dynamic mode, all of the same sourcing behaviors that occur in the UI also occur in your script as each line is executed. As a result, you can obtain sourced, validated, and recalculated field values in “real-time” without first having to submit the record.

When working with records in standard mode, you must submit and then load the record to obtain these same values. In standard mode, all business logic is executed on only after data is submitted.

**Note:** User event script that are triggered through the UI will, by default, execute in the standard mode. They will not execute in dynamic mode.

**Do I have to enable dynamic mode?**

You must use the optional `initializeValues` parameter in the following APIs to control whether a record is processed in dynamic mode.

- `nlapiCopyRecord(type, id, initializeValues)`
- `nlapiCreateRecord(type, initializeValues)`
- `nlapiLoadRecord(type, id, initializeValues)`
In `nlapiTransformRecord(type, id, transformType, transformValues)`, use the `transformValues` parameter to control whether one record will be transformed into another in dynamic mode.

**Important:**

1. If you provide no value for the `initializeValues` parameter, records will copy, create, load, and transform in the standard mode of execution, where sourcing, validation, and recalculations occur only after a record is submitted.

2. Dynamic processing of a record is not currently supported in **user event** scripts. For example, even if you were to call `nlapiCreateRecord('salesorder', {recordmode: 'dynamic'})` in a user event script, the fields in the new sales order will not be processed dynamically.

The `initializeValues` parameter is an Object that can contain an array of name/value pairs of defaults to be used during record initialization. To initialize a record in dynamic mode, you set the `recordmode` initialization type to `dynamic`. For example:

- `var record = nlapiCopyRecord('salesorder', 55, {recordmode: 'dynamic'});`
- `var record = nlapiCreateRecord('salesorder', {recordmode: 'dynamic'});`
- `var record = nlapiLoadRecord('salesorder', 111, {recordmode: 'dynamic'});`
- `var transformRecord = nlapiTransformRecord('salesorder', 111, 'itemfulfillment', {recordmode: 'dynamic'});`

**Note:** For a list of additional initialization types that can be specified for the `initializeValues` parameter, see **Record Initialization Defaults** in the NetSuite Help Center. Note that you do not need to run scripts in dynamic mode to use these other initialization types.

**Is dynamic mode better than standard mode?**

Yes and no. There are obvious benefits to getting / setting field values in real-time. When working with records in dynamic mode, you do not need to submit the record for all business logic to be executed. In dynamic mode you can get field values and write logic around these values without having to submit a record and wonder what will be returned when the record is reloaded.

Note, however, when scripting in dynamic mode, the order in which you set field values matters. For some developers, this aspect might feel constraining. It is likely that scripting in dynamic mode will require you to refer back to the UI often. For example, on an invoice in the UI, you would not set the Terms field before setting the Customer field (see figure). The reason is that as soon as you set the Customer field, the value of Terms will be overridden. On an invoice, the value of Terms is sourced from the terms specified on the Customer record. The same behavior will happen in dynamic scripting. In your scripts, if you do not correctly set field values in the order that they are sourced in the UI, some of the values you set could be overridden.
In non-dynamic mode, you do not have to worry about the order in which you set field values. For some developers, this fact alone will make scripting in non-dynamic mode preferable.

**Can I change existing code to run in dynamic mode?**

Yes. If you decide to convert an existing script to one that runs dynamically, you will need to do the following:

1. Set the new initializeValues parameter to {recordmode: 'dynamic'} - for example:
   ```javascript
   var recDynamic = nlapiCreateRecord('salesorder', {recordmode: 'dynamic'});
   ```

2. Ensure that the field values you have set in your scripts are set in the correct order.
   As the second example Field Ordering shows, if you do not set your fields in the right order, you may end up overriding certain field values.

3. Update any code that uses either the `nlapiSetLineItemValue(...)` function or the `nlobjRecord.setLineItemValue(...)` method.
   Neither of these APIs will execute in dynamic mode. For details, see Working with Sublists in Dynamic Mode and Client SuiteScript in the NetSuite Help Center.

**Standard vs. Dynamic Mode Code Samples**

This section uses code snippets to further demonstrate some of the differences between scripting in standard mode and dynamic mode. The areas covered are field sourcing, field ordering, and field calculation.

**Sourcing**

**Example 1: Standard mode (sourcing does not occur real-time)**

This sample shows a script executing in standard (non-dynamic) mode. In standard mode, no real-time sourcing occurs. You cannot get values that you have not already set in your script.
// Create a new sales order and set the customer
var record = nlapiCreateRecord('salesorder');
record.setFieldValue('entity', 343);

// Try to get the value of salesrep. Note that null is returned because the value
// of salesrep is not sourced as you step through the code
var sr = record.getFieldValue('salesrep');

Example 2: Dynamic mode (sourcing occurs as each line executes)
This sample shows the same script, but running in dynamic mode. In this script, the value of
salesrep is sourced after the entity is set. Ultimately this allows you to write fewer lines of code,
as you are able to get many field values without first having to set them.

    // Create a new sales order and set the customer
    var record = nlapiCreateRecord('salesorder', {recordmode: 'dynamic'});
    record.setFieldValue('entity', 343);

    // Get the value of salesrep. Note that John Smith will be returned. The value of the salesrep
    // field is sourced from the salesrep value specified on the customer record
    var sr = record.getFieldValue('salesrep');

Field Ordering

Example 1: Standard mode (field ordering does not matter)
This sample shows that the order in which you set values does not matter in standard mode. In
this sample, you can set a sales rep before setting the entity, even though in the UI there is a
sourcing relationship between these two fields. In the UI, once you set the entity, the value of
salesrep is automatically sourced. In standard mode, field sourcing relationships are not
respected.

    // Create a sales order. First set salesrep, then set the customer (entity). When the
    // record is submitted, salesrep remains as 88, the internal ID for sales rep Bud Johnson.
    var record = nlapiCreateRecord('salesorder');
    record.setFieldValue('salesrep', 88);
    record.setFieldValue('entity', 343);

Example 2: Dynamic mode (field ordering matters)
In dynamic mode, if you write the same script, the value of salesrep will be overridden when
the next line of code is executed. In this example, when you submit the record the value of
salesrep will change from 88 to 333 (the value of the salesrep specified on the customer record).

    var record = nlapiCreateRecord('salesorder', {recordmode: 'dynamic'});
    record.setFieldValue('salesrep', 88);
    record.setFieldValue('entity', 343);

In dynamic mode, if you want salesrep to remain as 88, you must write:
var record = nlapiCreateRecord('salesorder', {recordmode: 'dynamic'});
record.setFieldValue('entity', 343);
record.setFieldValue('salesrep', 88);
Field Calculation

Example 1: Standard mode (field totals are not calculated in real-time)

In this beforeSubmit user event script, if a line is added to a sales order, the totals are not recalculated until you submit the line. The following sample shows a new line being added in a beforeSubmit script, however, the total is not recalculated.

```javascript
function beforesubmit(type)
{
    var record = nlapiGetNewRecord();
    record.selectNewLineItem('item');
    record.setCurrentLineItemValue('item', 'item', 441);
    record.setCurrentLineItemValue('item', 'quantity', '2');
    record.commitLineItem('item');
}
```

Example 2: Dynamic mode (field totals are calculated real-time)

In dynamic mode, you can add a line to a sublist (the Items sublist in this example) and not worry about recalculating the amount and subtotal fields. Even before the record is submitted, you can get accurate data in a beforeSubmit event.

```javascript
function beforesubmit(type)
{
    var record = nlapiGetNewRecord();
    record.selectNewLineItem('item');
    record.setCurrentLineItemValue('item', 'item', 441);
    record.setCurrentLineItemValue('item', 'quantity', '2');
    record.commitLineItem('item');
}
```

Client Scripting and Dynamic Mode

Generally speaking, the concept of dynamic mode does not apply to client scripting. The only exception is on the pageInit client event. Scripting in a "current record context" is always in dynamic mode.

Note that client (remote object) scripting does not support dynamic scripting. In a client script, if you attempt to copy, create, load, or transform a remote record object (an object on the NetSuite server), you will not be able to work with the record in dynamic mode.

The following is an example of a client (remote object) script. On the saveRecord client event, an estimate record is created. If you attempt to create the record in dynamic mode, an error is thrown. For example:

```javascript
function onSave()
{
    var rec = nlapiCreateRecord('estimate', {recordmode: 'dynamic'}); // this will not work
    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;
}
```
Chapter 13 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.

The following figure shows a combination of standard and custom fields appearing in the body sections of a record. 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 fields that appear on a sublist. Fields that appear on a sublist are typically referred to as “line items” or sublist fields and are accessed using Sublist APIs.

On the figure below:

1. **Body fields** - can be a mix of standard and custom fields.
2. **Body fields** - directly below a subtab, can be a mix of standard and custom fields
3. **Sublist fields** - not to be confused with body fields. See Working with Subtabs and Sublists for more information.
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 Working with Records and Subrecords 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)`

**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:

```javascript
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.

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 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 _contactfax. Note that you do not need to write the custom field’s prefix in the ID field (see figure below). Once 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.
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.

<table>
<thead>
<tr>
<th>Custom field type</th>
<th>Custom field prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity field</td>
<td>custentity</td>
</tr>
<tr>
<td>Item field</td>
<td>custitem</td>
</tr>
<tr>
<td>CRM field</td>
<td>custevent</td>
</tr>
<tr>
<td>Transaction body field</td>
<td>custbody</td>
</tr>
<tr>
<td>Transaction column field</td>
<td>custcol</td>
</tr>
<tr>
<td>Transaction item options</td>
<td>custcol</td>
</tr>
<tr>
<td>Item number fields</td>
<td>custitemnumber</td>
</tr>
<tr>
<td>Other custom fields</td>
<td>custrecord</td>
</tr>
</tbody>
</table>
Chapter 14 Working with Subtabs and Sublists

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: 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?
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).

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).

The next figure shows the Financial subtab, also on the Customer record. Notice that the information on this subtab is simply 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.

**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 `nlapiInsertLineItem(type, line)` and `nlapiRemoveLineItem(type, line)`.

**Sublist Types**

There are four types of sublists in NetSuite:

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

**Related Topics**

- Sublist Types
- Adding Sublists with SuiteScript
- Subtabs and Sublists - What’s the Difference?
**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 nllobjForm.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 sublist 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 nllobjRecord.commitLineItem(group) 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:** Server scripts are considered to be Suitelets, user event scripts, scheduled scripts, and portlet scripts. If you are not familiar with the differences between client and server scripting.

The following figure shows the Address Sublist, which is currently the only editor sublist in NetSuite that supports SuiteScript. In this figure, two line items exist on the Address sublist.
Working with Subtabs and Sublists

Sublist Types

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) 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: Server scripts are considered to be Suitelets, user event scripts, scheduled scripts, and portlet scripts.

Related Topics

- Sublist Types
- Subtabs and Sublists - What's the Difference?
- Working with Sublist Line Items
- Scriptable Sublists
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.

The next figure shows the Sales Team Sublist. This sublist appears on the Sales subtab of a Customer record. Any changes you make to the sublist line items are made inline.

**Note:** To see the Sales Team sublist you must have the Team Selling feature enabled.
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 customziation 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 Functions.)

The following figure shows the Subscriptions 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 deselect the check boxes).

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.
The last figure provides another example of a list sublist—the Billable Expenses Sublist 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.

### 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.

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.

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.
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.

Related Topics
- Sublist Types
- Subtabs and Sublists - What’s the Difference?
- Working with Sublist Line Items
- Scriptable Sublists
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 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 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.

```javascript
//Define the user event beforeLoad function
function tabsToSalesOrder(type, form) {
    //Define the values of the beforeLoad type argument
    if (type == 'edit' || type == 'view') {
        //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', 'customer', 'custpage_sample_subtab');

        //Add a second subtab to the first tab
        var sampleSubTab2 = form.addSubTab('custpage_sample_subtab2', 'Second Sample Subtab', 'custpage_sample_tab');

        //Add a field to the second subtab
        var newSubField2 = form.addField('custpage_sample_field2', 'select', 'My Employees', 'employee', 'custpage_sample_subtab2');
    }
}
```
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_mynewsubtab’

**Important:** To add sublists through point-and-click customization, see Custom Sublists in the NetSuite Help Center. When adding a sublist through UI customization, you are essentially just 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.

```javascript
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 nlobForm 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:**

```javascript
function demoSuiteletSublist(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
```
Working with Subtabs and Sublists

Adding Sublists with SuiteScript

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 );
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

// On the Sales Team sublist, the Primary checkbox appearing on
// line 2 will be set to true - or 'T' in the case of check boxes in NetSuite
nlapiSetLineItemValue('salesteam', 'isprimary', 2, "T");
Working with Sublist 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 `nlapiGetCurrentLineItemCount(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:
   a. Call `nlapiSelectNewLineItem(type)` to select and insert a new line (as you would in the UI).
   b. 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)` or `nlobjRecord.removeLineItem(group, linenum)` 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:
   a. Call `nlobjRecord.selectNewLineItem(group)` to select and insert a new line (as you would in the UI).
   b. Call `nlobjRecord.setCurrentLineItemValue(group, name, value)` to set the value of the line.

5. Call `nlobjRecord.commitLineItem(group)` 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.

```javascript
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);
```
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.

```javascript
// 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');
    }
}

Related Topics
- Getting and Setting Line Item Values
- Adding Sublists with SuiteScript
- Subtabs and Sublists - What’s the Difference?
- Sublist APIs

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 reppct = originalSo.getLineItemValue('salesteam', 'contribution', rep);
    if (reppct != '0.0%')
        continue;
}
var repId = originalSo.getLineItemValue('salesteam', 'employee', rep);

// keep the percent the same
if (reppct.substring(reppct.length-1) == '%')
{
  //remove the percent sign % from the end
  reppct = reppct.substring(0, reppct.length-1);
}
so.insertLineItem('salesteam', iRep);
so.setCurrentLineItemValue('salesteam', 'contribution', iRep, reppct);
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 solId = nlapiSubmitRecord(so, true);
Example 2

This sample shows how to get the values of the line items on the Address (addressbook) sublist.

```javascript
function getCustomerAddressBook()
{
    var record = nlapiLoadRecord('customer', 87);
    var numberOfAddresses = record.getLineItemCount('addressbook');
    nlapiLogExecution('DEBUG', 'numberOfAddresses', numberOfAddresses);

    for (var i=1; i <= numberOfAddresses; i++)
    {
        var internalid = record.getLineItemValue('addressbook', 'internalid', i);
        var defaultshipping = record.getLineItemValue('addressbook', 'defaultshipping', i);
        var defaultbilling = record.getLineItemValue('addressbook', 'defaultbilling', i);
        var label = record.getLineItemValue('addressbook', 'label', i);
        var address = record.getLineItemValue('addressbook', 'addrtext', i);

        nlapiLogExecution('DEBUG', 'Address Info', 'internalid=' + internalid + '
defaultshipping=' + defaultshipping + '
defaultbilling=' + defaultbilling + '
address=' + address);
    }
}
```

This figure shows field values as they appear in the Execution Log of the Script Deployment page.

The following figure shows field values for this script if you are running it in the SuiteScript Debugger. Note that when running a script in the Debugger, script execution details appear on the Execution Log tab in the Debugger, not on the Execution Log tab on the Script Deployment page.
Example 3

The following sample is a validateLine client script which uses `nlapiGetLineItemValue(type, fldnam, linenum)` to prevent the addition of Sales Order item lines with an amount greater than 10000.

```javascript
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.

```javascript
var rec = nlapiCreateRecord('cashsale');
```
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 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 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, simply 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 to 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');
```

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.
Related Topics

- Adding and Removing Line Items
- Adding Sublists with SuiteScript
- Subtabs and Sublists - What’s the Difference?
- Sublist APIs
Chapter 15 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 in the offline client and 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

* nlapiAddDays(d, days)
* nlapiAddMonths(d, months)
* nlapiCancelLineItem(type)
* nlapiDateToString(d, format)
* nlapiDisableField(fldnam, val)
* nlapiDisableLineItemField(type, fldnam, val)
* nlapiEncrypt(s, algorithm, key)
* nlapiEscapeXML(text)
* nlapiFormatCurrency(str)
* nlapiGetCurrentLineItemIndex(type)
* nlapiGetCurrentLineItemText(type, fldnam)
* nlapiGetCurrentLineItemValue(type, fldnam)
* nlapiGetFieldText(fldnam)
* nlapiGetLineItemCount(type)
* nlapiGetFieldValue(fldnam)
* nlapiSetFieldValue(fldnam, value, firefieldchanged, synchronous)
* nlapiGetLineItemValue(type, fldnam, linenum)
* nlapiSelectNode(node, xpath)
* nlapiSelectNodes(node, xpath)
* nlapiSelectValue(node, xpath)
* nlapiSelectValues(node, path)
* nlapiStringToDate(str, format)
* nlapiStringToXML(text)
* nlapiXMLToString(xml)
* nlapiSetLineItemValue(type, fldnam, linenum, value)
* nlapiInsertLineItem(type, line)
* nlapiRemoveLineItem(type, line)
* nlapiGetRecordType()
* nlapiGetRecordId()
* nlapiGetString()
* nlapiGetUser()

**Note:** See also Client SuiteScript and Online Forms, which discusses the Available Without Login deployment preference as it applies to client SuiteScript.
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.

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`.

SuiteScript and the Offline Client

Only a subset of the SuiteScript API is supported in the offline client. A list of these APIs are provided in the topic Working with Online Forms.

If you use SuiteScript APIs that are are not supported in the offline client, end users will receive the following error message when they save a record:
“You cannot submit this form due to an unexpected error.”
Chapter 16 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 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.

**Note:** • 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 simply 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.

**Related Topics**

- Inline Editing and SuiteScript Overview
- Inline Editing Using nlapiSubmitField

### 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 simply needs to click away from the field.
In SuiteScript, the programmatic equivalent of inline editing the Phone field on customer record 140 is:

```javascript
var updatefield = nlapiSubmitField('customer', '149', 'phone', '800-555-1234');
```

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:

```javascript
var rec = nlapiLoadRecord('customer', '149');  // 10 units
rec.setFieldValue('phone', '800-555-1234');
var id = nlapiSubmitRecord(); // 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:

```javascript
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.

```javascript
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.

**Related Topics**
- Inline Editing and SuiteScript Overview
- What's the Difference Between xedit and edit User Event Types?
- Inline Editing and nlapiGetNewRecord()
- Inline Editing and nlapiGetOldRecord()
- User Event Scripts
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 as a result of 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 to use `nlapiGetOldRecord()` in a user event script that will execute in the context of an inline edit or mass update. This script returns all the fields on the record prior to the field updates being committed to the database.

```javascript
function getUpdatedFields(type)
{
    // if the record is inline edited or mass updated, run this script
    if (type == 'xedit')
    {
        // call nlapiGetOldRecord to get all field values on the current record
        // prior to the write operation
        var fields = nlapiGetOldRecord().getAllFields()

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

### Related Topics
- Inline Editing and SuiteScript Overview
- Inline Editing (xedit) as a User Event Type
- Inline Editing and `nlapiGetNewRecord()`
- User Event Scripts
Part 4  Understanding NetSuite Script Types
Chapter 17 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:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suite</td>
<td>Build interactive web applications by scripting web requests</td>
</tr>
<tr>
<td>User Event</td>
<td>Define business logic that is triggered when records are created, updated, viewed, or deleted</td>
</tr>
<tr>
<td>Scheduled</td>
<td>Schedule complex batch operations or queue them on-demand for execution</td>
</tr>
<tr>
<td>Client</td>
<td>Define business logic and perform client-side validation on your forms</td>
</tr>
<tr>
<td>Portal</td>
<td>Push scriptable portlets to your dashboards and centers</td>
</tr>
<tr>
<td>Mass Update</td>
<td>Perform an update to a record as part of a mass update</td>
</tr>
<tr>
<td>Workflow Action</td>
<td>Defines a custom action on a record that can be used as part of a workflow</td>
</tr>
<tr>
<td>Bundle Installation</td>
<td>Defines scripts that run as part of bundle installation or update</td>
</tr>
</tbody>
</table>

To actually run a script in NetSuite, see Running a Script in NetSuite.
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**.
Chapter 19 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 Functions
- 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 SuiteScript and Online Forms
- 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 are discussed in the section Client Event Functions. 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

Once 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 just a specific form on a single record.

**Related Topics**
- Client Event Functions
- Form-level and Record-level Client Scripts
- Role Restrictions in Client SuiteScript
- Client SuiteScript Samples

**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 Functions), 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 Functions**

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().

Related Topics
- Form-level and Record-level Client Scripts
- Client SuiteScript Samples
- Client Event Functions
<table>
<thead>
<tr>
<th>Client Event Type (and sample function name)</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pagelnit</td>
<td>type: the mode in which the record is being accessed. These modes can be set to: create, copy, edit</td>
<td></td>
<td>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. This is similar to an onLoad JavaScript client-side event. See Page Init Sample.</td>
</tr>
<tr>
<td>saveRecord</td>
<td></td>
<td>Boolean</td>
<td>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 saveRecord event. A return value of <code>false</code> suppresses submission of the form. See Save Record Sample.</td>
</tr>
<tr>
<td>validateField</td>
<td>type: the sublist internal ID name: the field internal ID linenum: line number if this is a sublist. Line numbers start at 1, not 0.</td>
<td>Boolean</td>
<td>This client event occurs whenever a field is about to be changed by the user or by a client side call. Returning false from this function prevents the field's value from changing. This function is automatically passed up to three arguments by the system: <code>type</code>, <code>name</code>, <code>linenum</code>. This event type is similar to an onBlur JavaScript client-side event. In NetSuite, validateField events execute on fields added in beforeLoad user event scripts. Note: This event type does NOT apply to drop-down select or check box fields. See Validate Field Sample.</td>
</tr>
<tr>
<td>fieldChanged</td>
<td>type: the sublist internal ID name: the field internal ID linenum: line number if this is a sublist. Line numbers start at 1, not 0.</td>
<td></td>
<td>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. This function is automatically passed up to three arguments by the system: <code>type</code>, <code>name</code>, <code>linenum</code>. This event type is similar to an onChange JavaScript client-side event. See Field Changed Sample.</td>
</tr>
<tr>
<td>postSourcing</td>
<td>type: the sublist internal ID name: the field internal ID</td>
<td></td>
<td>This client event occurs following a field change once all the field's child field values are sourced from the server. Enables fieldChange style functionality to occur once all dependent field values have been set. This function is automatically passed up to two arguments from the system: <code>type</code>, <code>name</code>. See Post Sourcing Sample.</td>
</tr>
</tbody>
</table>
### Client Event Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>validateLine</td>
<td>type: the sublist internal ID</td>
<td>Boolean</td>
<td>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 <code>nlapiGetCurrent***</code> functions. This function is automatically passed the <code>type</code> argument from the system.</td>
</tr>
</tbody>
</table>
| recalc     | type: the sublist internal ID |         | Event occurs after a line has been added to a sublist. This allows for any global actions that change whenever the contents of the sublist change. Notes:  
- Recalc functions will not execute when the Add Multiple button is used to add multiple line items.  
- Scripts that execute on recalc events do not need to include a call to `nlapiCommitLineItem(type)` since recalculation occurs automatically. Calling `nlapiCommitLineItem` will end up calculating the line twice. |
| validateInsert | type: the sublist internal ID | Boolean | The validateInsert event occurs when you insert a line into an edit sublist. For information on the edit sublist type, see Editor Sublists in the NetSuite Help Center. 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 `nlobjRecord.insertLineItem(...)`.
Note that returning false on a validateInsert blocks the insert. |
| validateDelete | type: the sublist internal ID | Boolean | The validateDelete event occurs when you try to remove an existing line from an edit sublist. Returning false blocks the removal. For information on the edit sublist type, see Editor Sublists in the NetSuite Help Center. |

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

---

**Form-level and Record-level Client Scripts**

---

SuiteScript Developer and Reference Guide
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 five 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.

### 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.

```javascript
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 as Admin.

How Many Client Events Can I Execute on One Form?

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

For information on client event functions, see Client Event Functions.
**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.

The following figure shows an alert that is thrown because the function specified in the pageInit event is not properly defined. When the logged-in users receive this alert, they can contact their NetSuite administrators for further instruction.

Note that alerts provide the scriptId (in this case *customscript28*) 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 specify who, if anyone, they want to receive emailed messages when script errors occur. (Go to Setup > Customization > Scripts > New > Client to access the Script record page for record-level client scripts.)

Additionally, the Script Deployment page for **record-level** client scripts includes an Execution Log subtab, on which all script errors are logged. (Go to Setup > Customization > Script Deployments > to select your deployment.)

**Related Topics**

- What is Client SuiteScript?
- Form-level and Record-level Client Scripts
- Client SuiteScript Samples

**Client SuiteScript and Online Forms**

When running client scripts in online forms, you must select the Available Without Login check box to ensure that the script will run on the form (see figure). If the Available Without
Login check box is not selected, users will still have access to the form, however, the client script will not run.

Additionally, when working with online forms you can only use the APIs listed in the table called SuiteScript APIs available on online forms and externally available Suitelets.

To access the Available without Login preference for client scripts:

1. Navigate to your form and click the Custom Code tab.
2. In the Script File field, click the + (New) option. The File popup appears.
3. Select the Available without Login check box.
4. Load your client script into the Select File field, and click Save.

Related Topics
- What is Client SuiteScript?
- Client Script Execution
- Client SuiteScript Samples

**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 simple script that has a function on every exposed event. Consider the following client script:

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

function mySaveRecord()
{
    alert('mySaveRecord');
    return true;
}

function myValidateField(type, name, linenum)
{
    if (name == 'custentity_my_custom_field')
    {
        alert('myValidateField');
        alert('type=' + type);
        alert('name=' + name);
        alert('linenum=' + linenum);
    }
    return true;
}

function myFieldChanged(type, name, linenum)
{
    alert('myFieldChanged');
```
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 (see figure).

**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.
When a person opens any Case record, the following alerts are thrown right way on the pageInit (page load) trigger:

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 PageInit 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

```javascript
function pageInit()
{
    // 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 pageInit()
{
    // 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(type, name)
{
    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')
    {
        // Once 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**

```javascript
function ValidateField(type, name)
{
    // if fieldA is not greater than 5 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.

```javascript
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 way 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

```javascript
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.");
    }
}
```
Chapter 20 User Event Scripts

The following topics are covered in this section. If you are new to user event script, 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 triggered to run by other user event scripts. You can, however, execute a user event script from a call within a scheduled, portlet, or Suitelet script.

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”.

SuiteScript Developer and Reference Guide
Which User Event Types are Available in SuiteScript?

The user event types that are available in scripting are:

- **Before Load** – event occurs when a read operation on a record takes place
- **Before Submit** – event occurs when a record is submitted, but before the changes are committed to the database
- **After Submit** – event occurs after the changes are committed to the database

See User Event Script Execution Types or specific details on these event types.

---

User Event Script Execution

User events scripts are executed based on operations types defined as: **before load**, **before submit**, and **after submit**. 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 before load operation:

1. The client sends a read operation request for record data. (The client request can come from the user interface, Web services or Server 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 **before load** operation occurs – before the requested data is returned to the client.
4. The client receives the now validated/processed **before load** data.
**User Event beforeSubmit and afterSubmit Operations**

The following steps and diagram provide an overview of what occurs on submit (after submit and before submit) 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 or Server 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 before submit operation

   The submitted data has NOT yet been committed to the database.

2. Once data has been validated, it is committed to the database.

3. If this (newly committed) data is then called by an after submit operation, the data is taken from the database and is sent to the application server for additional processing. Examples of after submit 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)
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 before load operation for a given record, 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();
```

Related Topics

- Setting the User Event type Argument
- User Event Script Samples
- SuiteScript Functions
- Enabling SuiteScript
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.

**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.)

```javascript
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');
    }
}
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:
<table>
<thead>
<tr>
<th>Operation Type</th>
<th>Execution Event Type</th>
<th>Notes</th>
</tr>
</thead>
</table>
| **beforeLoad** | **type**: the read operation type  
- create  
- edit  
- view  
- copy  
- print  
- email  

*form*: an nlobjForm object representing the current form  
*request*: an nlobjRequest object representing the GET request (Only available for browser requests.) | 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, or calling nlapiLoadRecord.  
The user-defined function is executed prior to returning the record or page. The function is passed either the *type*, *form*, or *request* arguments by the system.  
**Note**: beforeLoad user events cannot be triggered when you load/access an online form. |
### Operation Type

**beforeSubmit**

<table>
<thead>
<tr>
<th>Execution Event Type</th>
<th>Notes</th>
</tr>
</thead>
</table>
| **type**: the write operation type | 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. 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. **Notes:**
- The approve, cancel, and reject argument types are only available for record types such as sales orders, expense reports, timebills, purchase orders, and return authorizations.
- Only 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.
- **Best practices:** To set a field on a record or make ANY changes to a record that is being submitted, do so on a beforeSubmit operation, NOT an afterSubmit operation. If you set a field on an afterSubmit, you will be duplicating a record whose data has already been committed to the database. |
| create | |
| edit | |
| delete | |
| xedit - (see Inline Editing and SuiteScript) | |
| approve - (only available for certain record types) | |
| reject - (only available for certain record types) | |
| cancel - (only available for certain record types) | |
| pack - (only available for certain record types, for example Item Fulfillment records) | |
| ship - (only available for certain record types, for example Item Fulfillment records) | |
| markcomplete (specify this type for a beforeSubmit script to execute when users click the Mark Complete link on call and task records) | |
| reassign (specify this type for a beforeSubmit script to execute when users click the Grab link on case records) | |
| editforecast (specify this type for a beforeSubmit script to execute when users update opportunity and estimate records using the Forecast Editor) | |
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.

### 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.

**Script:**

```javascript
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 simple 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 *after submit* event that will create the phone call record. See the following sample code:

```javascript
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 PhoneCall
            var call = nlapiCreateRecord("phonecall");

            //Setting the title field on the PhoneCall 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.

In the above use case, *after submit* is a better event to handle the logic than *before submit*. In the *before submit* event, the customer data has not yet been committed to the database. Hence, putting the phone call logic in the *after submit* 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.

The phone call use case may be further enhanced by redirecting the user to the Phone Call page once it is created. This 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 as a result of user actions carried out through the browser, they are also triggered by other means as well (for example, CSV, Web Services, offline client).

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, in order 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):

```javascript
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 transactions 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 before load event can provide flexibility to runtime customization.

**Note:** For more specific information about NetSuite entry forms and transaction forms, see 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. 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 before load event of the sales order record.

```javascript
/***************************************************************
* 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
        else if (month == 0 | month == 2 | month == 4 | month == 6 | month == 7 | month == 9 | month == 11)
```

```javascript
```
When the script is deployed, all sales order forms will have the Eligible EOM Promotion checkbox only during the last five days of the month as shown below.

Note that since these UI elements are created dynamically, they are superficial and do not have supporting backend data models. There is a disconnect between the UI and backend 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 backend 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. As a result, the real fields are populated and the data is saved.
Related Topics

- What Are User Event Scripts?
- User Event Script Execution
- Setting the User Event type Argument
Chapter 21 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
- Restricted Parameters 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.

Shown below are screenshots of a simple 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.

Once a Suitelet has been deployed, developers can create NetSuite tasklinks to these scripts, which can then be used to customize existing NetSuite centers.

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.

Below is the source code for this Suitelet. It is executed on the server, which generates HTML and sends it to the browser.

```javascript
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 field');
        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: Client side alerts are not supported in Suitelets, as Suitelets are considered to be "server side" SuiteScripts. The following snippet will run in client SuiteScripts, however, it will not run if the script is a Suitelet:

    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.

**Note:** Suitelets can support up to 50 concurrent connections from a single user login. Note that any concurrent RESTlet connections also count towards this limit.
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 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 `custpagecustomtab` or `custpagemytab`.

The figure below shows a custom interface that has been built with the following SuiteScript UI objects:

- `nlobjForm`
• nobjTab
• nobjSubList
• nobjField

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.

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.

```javascript
/***********************************************
* 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 = new Array();
            var columns = new Array();

            //Use the supplied custparam_phone value as filter
            filters[0] = new nlobjSearchFilter('phone', null, 'is', request.getParameter('custparam_phone'));
            columns[0] = new nlobjSearchColumn('entityid', null, null);

            //Search for customer records that match the filters
            var results = nlapiSearchRecord('customer', 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 + ' |';
                    }
                }
            }
        }
    }
}```
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=(123)-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.

---

### Related Topics

- Suitelet Script Execution
- Suitelets Samples
- UI Objects

---

### Restricted Parameters in Suitelet URLs

There are certain URL parameters that are reserved, and users should not use them when creating custom parameters for their Suitelet URLs. For example, the following URL includes a custom parameter called `id`.

```
https://.../site/hosting/scriptlet.nl?script=44&deploy=3&id=500
```

In this case, accessing a Suitelet using a URL that includes a custom `id` parameter throws an error if you try to build a NetSuite entry form using the nlapiCreateForm function. The following tables provides a list of reserved parameters:
If you include any of these URL parameters you may get 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 `custom_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.)
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.

When the Script Deployment page is saved, a link to the Suitelet appears (see figure).
Suitelets Samples

The following sample Suitelets show how to return HTML and XML documents, embed in iFrame, as well as how to create simple 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
- Create a Form with Embedded Inline HTML
- Embed a Suitelet in iFrame
Writing Your First Suitelet

This simple Hello World! sample shows how to return an HTML document in a Suitelet.

Script:

```javascript
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:

```javascript
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:**

```javascript
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 );
    }
}
```
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:**

```javascript
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,'is','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 );

Add a Suitelet to a Tab

Add a Suitelet to a Tab

[UE=HELP_TOPIC_OUT_OF_DATE_ALERT=UE]

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');
    }
}

Create a Suitelet Email Form

The following sample shows how to create a Suitelet form to email the results.

Script:

```javascript
/**
 * 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');
    }
    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 simple 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:

```javascript
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);
}
```
Create a Form with Embedded Inline HTML

Here is another “Hello World” example, but this time using Inline HTML which is embedded in a NetSuite form. A field is added with the nlobjForm.addField method of the type 'inlinehtml', and the nlobjField.setDefaultValue method is used to provide the HTML code.

**Script:**

```javascript
function HelloWorldInlineHTML(request, response) {
    var form = nlapiCreateForm('My Form', false);
    var myInlineHtml = form.addField('custpage_btn', 'inlinehtml');
    myInlineHtml.setDefaultValue('<html><body><h1>Hello World</h1></body></html>);

    response.writePage(form);
}
```
Embed a Suitelet in iFrame

This sample shows how to embed a Suitelet in iFrame, through its external URL. You need to append `ifrmcntnr=T` to the external URL of your Suitelet deployment.

**Note:** This is the recommended approach especially when you are using Firefox as the browser to render your HTML page.

**Script:**

```javascript
function EmbediniFrame(request, response) {
    var form = nlapiCreateForm("", true);
    var stateSelect = form.addField("custpage_state", "select", "State", null);

    stateSelect.setMandatory(true);
    stateSelect.addSelectOption("AL", "Alabama");
    stateSelect.addSelectOption("AK", "Alaska");

    var submitButton = form.addSubmitButton("Okay!");
    response.writePage(form);
}
```

During script deployment, make sure to do the following:
- Set the status to **Released**
- Select **Available Without Login**
- Set Audience to **All Roles** and **All Employees**

For more information about deploying a script, see **Step 5: Define Script Deployment**.

Next, you need to create the HTML file where you will embed your Suitelet in iFrame using the external URL of your script deployment. Note how `ifrmcntnr=T` is appended to the external URL.

**HTML:**
```html
<html>
<head></head>
<body>
<iframe src="https://forms.netsuite.com/app/site/hosting/scriptlet.nl?script=207&deploy=1&compid=563214&h=cfdf4a34a46c39a0f3d2&ifrmcntnr=T"/>
</body>
</html>
```

Loading the HTML file in your browser (such as Firefox) renders the following output:
Chapter 22 RESTlets

REST is an acronym for Representational State Transfer, a style of software architecture that focuses on representation and resources. A design that conforms to the constraints of REST is described as RESTful. A description of these constraints and of general REST principles is available at http://en.wikipedia.org/wiki/Representational_State_Transfer.

RESTlets, a new type of server SuiteScript, expose the power of the SuiteScript interface to REST and allow access to a REST-based Web services API. This new API is a lightweight, flexible, and standards-based framework that provides an alternative to SOAP-based Web services. Developers can use RESTlets to define custom RESTful integrations with NetSuite.

RESTful Web services can support a significant expansion of supported behaviors over NetSuite's SOAP-based Web services, which are limited to those defined as SuiteTalk operations. RESTful Web services also can enhance usability and performance over SuiteTalk. In addition, RESTlets are more secure than Suitelets, which are made available to users without login.

RESTlets provide the following additional benefits:

- Support of stateless requests and responses between client and server
- Built-in authentication using user credentials in HTTP header
- Ease of adoption for developers familiar with SuiteScript
- Control over both client and server implementation
- Suitability for development of: mobile clients on platforms such as iPhone and Android, integration of external Web-based applications such as Gmail or other Google Apps, and backends for Suitelet-based user interfaces.

For details about working with RESTlets, see:

- What Are 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
Related Topics

- Script Types Overview
- Suitelets
- What is SuiteScript?
- SuiteTalk Platform Overview
RESTlets are a form of Web services integration. Each RESTlet is a server-side script that operates in a request-response model, and is invoked by an HTTP request to a system-generated URL.

RESTlets follow the principles of the REST architectural style and use HTTP. They use 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 SuiteScript, see the SuiteScript help in the NetSuite Help Center.

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

**Related Topics**

- 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
- SuiteScript - The Basics
- SuiteScript Governance
- Debugging SuiteScript
- Error Handling APIs

**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, or through a check for an existing NetSuite session.
   - For an externally hosted client, request-level credentials are required.
   - For a NetSuite-hosted client, the existing NetSuite session is reused.

3. The RESTlet script is invoked, providing access to the server SuiteScript API.

4. 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.

5. The RESTlet interrogates the string or object, and can perform any of the full range of SuiteScript actions.

6. The RESTlet returns results as a string or a serialized object.

**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.

### Related Topics
- 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

### 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.

- For a RESTlet called from an external client, you need to use the NetSuite-specific method NLAuth in the HTTP Authorization header to pass 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 a valid NetSuite session occurs, and this existing session is reused.

**Using NLAuth in the Authorization Header**

NLAuth passes in the following login credentials:

- **nlauth_account** - NetSuite company ID (optional)
- **nlauth_email** - NetSuite user name (required)
- **nlauth_signature** - NetSuite password (required)
- **nlauth_role** - internal ID of the role used to log in to NetSuite (optional)

**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.

**Authorization Header Formatting**

The Authorization header should be formatted as:

```
NLAuth<space><comma-separated parameters>
```

For example:

```
Authorization: NLAuth nlauth_account=123456, nlauth_email=jsmith@ABC.com,
nlauth_signature=xxxxxxxx, 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 needs to include the domain https://rest.netsuite.com, which has been created to support RESTlets.

In this case, you receive an error if you attempt to use another domain.

**Important:** As of September 2012, NetSuite began hosting new customer accounts in multiple data centers. As a result, the domain used for external client RESTlet access to NetSuite may vary per customer account. Currently, this domain could be https://rest.netsuite.com or https://rest.na1.netsuite.com. NetSuite provides a service that you can use to dynamically discover the correct domain. 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 does not include the domain.

In this case, the RESTlet uses the https://system.netsuite.com domain because that is the domain used in the client session reused by the RESTlet.

The following RESTlet deployment record shows examples of URLs used by NetSuite-hosted and externally hosted clients:

**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

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:
  - systemDomain - https://system.netsuite.com or https://system.na1.netsuite.com
  - webservicesDomain - https://webservices.netsuite.com or https://webservices.na1.netsuite.com

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.

**Note:** A Web services operation also is available, in the 2012.2 and later endpoints, to support dynamic discovery of URLs for external client access. For details, see `getDataCenterUrls`.

**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://system.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=****

**Note:** If you specify an account ID in this type of request, an error is returned.

**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:

```json
[{
    "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:

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Input</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Parameter Object</td>
<td>Object</td>
<td>Requests a representation of the specified resource.</td>
</tr>
<tr>
<td>POST</td>
<td>Object</td>
<td>Object</td>
<td>Submits data to be processed, for example from an HTML form. Data is included in the body of the request, and can result in creation of a new resource, updates of existing resource(s), or both.</td>
</tr>
</tbody>
</table>
RESTlets

What Are RESTlets?

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.

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Input</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>Parameter Object</td>
<td>No Content</td>
<td>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.</td>
</tr>
<tr>
<td>PUT</td>
<td>Object</td>
<td>Object</td>
<td>Uploads a representation of the specified resource.</td>
</tr>
</tbody>
</table>

**Related Topics**

- 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
- RESTlet Governance and Session Management
- RESTlet Debugging
- RESTlet Error Handling

**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.

RESTlets can support up to 50 concurrent connections from a single user login. (Note that any concurrent Suitelet connections also count toward this limit.) If more than 50 concurrent connections are made, an HTTP error code of 500 is returned, and an ExceededRequestLimitFault is thrown. This is the same fault that is thrown when concurrent Web services SOAP requests exceed limits.

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 Debugging SuiteScript.

**Important:** For deployed debugging of a RESTlet, you need to set the cookie of the client application that runs the RESTlet to the same cookie listed for the RESTlet in the Debugger. This cookie contains the NetSuite version and the JSESSIONID. 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.

Related Topics
- 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
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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>RESTlets</th>
<th>SuiteTalk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported Operations</td>
<td>get, search, add, update&lt;br&gt;(heterogeneous)</td>
<td>get, search, add, update&lt;br&gt;(homogenous)</td>
</tr>
<tr>
<td>Authentication Supported?</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Supported HTTP Methods</td>
<td>GET, PUT, POST, DELETE</td>
<td>POST</td>
</tr>
<tr>
<td>Passing of Login Details</td>
<td>in authorization header</td>
<td>in body (SOAP)</td>
</tr>
<tr>
<td>Passing of Parameters</td>
<td>GET parameters on URL</td>
<td>all parameters in body (SOAP)</td>
</tr>
<tr>
<td>Supported Content Types</td>
<td>JSON, text/xml (explicit)</td>
<td>text/xml (explicit)</td>
</tr>
<tr>
<td>Environment</td>
<td>lightweight, more suitable for mobile devices, bundleable</td>
<td>heavy programming and deployment environment (C#, Java)</td>
</tr>
<tr>
<td>Supported Concurrency</td>
<td>up to 50 (combined with Suitelets), no special license required</td>
<td>up to 10, SuiteCloud Plus license required</td>
</tr>
<tr>
<td>URL Clarity?</td>
<td>Yes&lt;br&gt;<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>&lt;br&gt;(Note that for clients hosted by NetSuite, use the relative URL that does not include the domain.)</td>
<td>No&lt;br&gt;<a href="https://webservices.netsuite.com/services/NetSuitePort_2011_1">https://webservices.netsuite.com/services/NetSuitePort_2011_1</a></td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>RESTlets</th>
<th>Suitelets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported Operations</td>
<td>get, search, add, update</td>
<td>get, search, add, update</td>
</tr>
<tr>
<td>Authentication Supported?</td>
<td>Yes</td>
<td>No, when available without login and executed as admin programmatically</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Yes, when accessed from a browser by a logged-in NetSuite user</td>
</tr>
<tr>
<td>Script Functions and HTTP Methods</td>
<td>individual script function for each HTTP method</td>
<td>one script function for all HTTP method</td>
</tr>
<tr>
<td>Content Handling</td>
<td>built-in handling of JSON input/output</td>
<td>must write code to convert JSON input/output</td>
</tr>
<tr>
<td>Governance</td>
<td>5,000 usage units per script</td>
<td>1,000 usage units per script</td>
</tr>
<tr>
<td>URL Clarity?</td>
<td>Yes <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> (Note that for clients hosted by NetSuite, use the relative URL that does not include the domain.)</td>
<td>No <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></td>
</tr>
</tbody>
</table>

Related Topics
- RESTlets
- What Are RESTlets?
- Creating a RESTlet
- Debugging a RESTlet
- Sample RESTlet Code
- Sample RESTlet Input Formats
- RESTlet Status Codes and Error Message Formats
- Suitelets
- SuiteTalk Platform Overview
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. Once 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 the SuiteScript plug-in for Eclipse, this process is automated. For more information, see Step 2: Add Script to NetSuite File Cabinet.

   a. Go to [TP=LIST_MEDIAITEMFOLDER=TP], 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.

   b. Click Add File, and browse to the .js file.
Create the RESTlet Script Record

Once 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 [TP=EDIT_SCRIPT=TP], and click RESTlet.

   ![RESTlet Script Record Form]

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 in your script: 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)**

Once 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 Setup > Customization > 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:** Once 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.

**Related Topics**

- RESTlets
- What Are RESTlets?
- RESTlets vs. Other NetSuite Integration Options
- Debugging a RESTlet
- Sample RESTlet Code
- Sample RESTlet Input Formats
- RESTlet Status Codes and Error Message Formats
- Running Scripts in NetSuite Overview
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 Setup > Customization > Script Debugger, or log in to the debugger domain [https://debugger.netsuite.com](https://debugger.netsuite.com). (See Deployed Debugging for details.)
4. Click the Debug Existing button in the main Script Debugger page.
5. Select the RESTlet script that you want to debug in the Script Debugger popup. Once you click the Select option button, the RESTlet's cookie displays in a banner. This cookie includes the NetSuite version and the JSESSIONID.
6. Copy the cookie and paste it into a text file so that you have it 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 cookie in your client application to the value 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 cookie, as described in the steps above. This cookie includes a valid NetSuite version.

• Invalid JSESSIONID
  Ensure that you have correctly copied the cookie, as described in the steps above. This cookie includes a valid JSESSIONID.

• 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. For example, 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](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 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 Request from Android
- Example RESTlet Request Using nlapiRequestURL

---

#### Simple Example to Get Started

Use the following example as a very simple GET method test when you are getting started with RESTlets:

```javascript
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**

```javascript
// 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**

```javascript
// 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);
```

**Related Topics**
- RESTlets
- Sample RESTlet Code
- Sample RESTlet Input Formats
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 Request from Android
HttpPost post = new HttpPost( URL + urlParams );

HttpPost httpParameters = new BasicHttpParams();
HttpConnectionParams.setConnectionTimeout( httpParameters, 20000 );
HttpConnectionParams.setSoTimeout( httpParameters, 42000 );

String authorization = "NLAuth nlauth_account=" + account + ", nlauth_email=" + email + ", nlauth_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 = new Array();
headers['User-Agent-x'] = 'SuiteScript-Call';
headers['Authorization'] = 'NLAuth nlauth_account='+cred.account+', nlauth_email='+cred.email+', nlauth_signature='+cred.password+', nlauth_role='+cred.role;
headers['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 );

    //Bellow is been used to put breakpoint
    var i=0;

*****RESTLET Code*****

// Create a standard NetSuite record
function createRecord(datain)
{
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**

```json
{
    "shipcomplete":false,
    "giveaccess":false,
    "globalsubscriptionstatus":"1",
    "isperson":false,
    ...
    "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"
}
```

Item Record Format

**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**

```json
{
    "salesdescription":"Cat 5 Patch Cable 10 ft",
    "vendorname":"CABL0002-64",
    "averagecost":3.50,
    ...
    "pricing":
    [
        ...
        {"currency":
        {
            "name":"British pound",
            "internalid":2
        },
        "pricelist":
        ["pricelevel":
        {
            "name":"Alternate Price 1",
            "internalid":2
        }
        ]
    ]
}
```
[{
    "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
  }
}

**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 each combination of these features is 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}
],
"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}
]}

**Quantity Pricing Only Enabled**

"pricing":
[
  
  "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}
    ]
  ]
Multiple Prices, Multiple Currencies Enabled

'pricing':
[
    {
        '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': 79.00, '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, Quantity Pricing, Multiple Currencies Enabled
["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":79.00,"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,  
        "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"  
}

Sales Order Record Format

**JSON**

```json
{
  "total": 64.04,
  "altshippingcost": 5.67,
  "taxtotal": 4.45,
  "tranid": "120",
  "orderstatus": "E",
  "shipcomplete": false,
  "discounttotal": 0.00,
  "entity": "76",
  "billaddress": "Doug Fabre\nChess\nChess Art Gallery\n150 N Ocean Dr\nMonterey CA 93940",
  "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,
```

---

RESTlets Guide
"recordtype":"salesorder",
"trandate":"10/14/2006",
"fax":"831-555-5230",
"customform":"88",
"links":
[
  {"trandate":"10/14/2006","tranid":"8","type":"Item Fulfillment","linktype":"Receipt/Fulfillment"}
],
"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":
[
  {
    "istaxable":"T","description":"10 ft Serial Cable DB25M DB25F",
    "custcol6":429,"custcol7":2.5,
    "item":"46","quantity":1,"isestimate":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,"isestimate":false,"commitinventory":"1"
  }
],
"excludecommission":false,
"shipaddress":"Chess\nChess Art Gallery\n150 N Ocean Dr\r\nMonterey CA 93940","tobeemailed":false
}
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

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:
- **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 more information about HTTP status codes, see http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.
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: fevsjhw4tjij2juy3le73

- 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**

```json
{
    "error": {
        "code": "SSS_INVALID_SCRIPTLET_ID",
        "message": "That Suitelet is invalid, disabled, or no longer exists."
    }
}
```

**Text**

```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.
Related Topics

- RESTlets
- What Are RESTlets?
- RESTlets vs. Other NetSuite Integration Options
- Creating a RESTlet
- Debugging a RESTlet
- Sample RESTlet Code
- Sample RESTlet Input Formats
- Error Handling APIs
Chapter 23 Scheduled Scripts

Scheduled scripts run on the NetSuite server. Compared to user event scripts, scheduled scripts are given a higher limit of usage governance. Therefore, scheduled scripts are ideal for long running tasks and batch jobs.

**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 to five by purchasing NetSuite’s SuiteCloud Plus module. See Using Multiple Script Queues to Run Scheduled Scripts for more information.

People new to scheduled scripts should read all of the following topics. However, these topics do not need to be read in order:

- **Understanding Scheduled Script Execution** - explains how scheduled scripts execute in NetSuite. The table in Deployment Status and Script Execution Summary provides a succinct overview of what users can and cannot do with a scheduled script based on its deployment status.

- **Deploying Scheduled Scripts** - explains how to deploy a scheduled script either to run immediately or to run at some scheduled future time.

- **Creating Multiple Deployments for a Scheduled Script** - explains how to create multiple deployments for the same scheduled script and why you may want to.


- **Setting the Scheduled Script type Argument** - describes how to associate an event type with a scheduled script.

- **Setting Recovery Points in Scheduled Scripts** – describes the recovery point mechanism, and why you would want to use this feature.

- **Understanding Memory Usage in Scheduled Scripts** – explains how scripts consume memory, and how to use memory more effectively.

- **Monitoring Scheduled Scripts** - provides details on monitoring script activity using both the UI and APIs.

- **Scheduled Script Samples** - provides scheduled script code samples.

- **Using Multiple Script Queues to Run Scheduled Scripts** - describes the SuiteCloud Plus module, which is available for purchase. Adding this module to your existing account upgrades your number of scheduled script queues from one to five.

- **Scheduled Script Best Practices**
Understanding Scheduled Script Execution

Whether a scheduled script is executed “on-demand” for immediate execution, or it scheduled to run in the future, once a script has been placed into the NetSuite scheduling queue, it is executed serially on a per-company basis. In other words, 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 is immediately executed.

A scheduled script may not run at precisely the time it was scheduled if there are multiple long-running scripts before it in your company’s queue. The first of your company’s scripts to be placed in the queue will be the first script to run. Although multiple scheduled scripts can exist in the queue, only a single script can be executed at any given time.

For example, if your company has scheduled two scripts to start at 10:00 p.m., the second script that was also set to run at 10:00 p.m. will not start until the first script completes. Once the first script completes, the second script will execute immediately.

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 to five by purchasing NetSuite’s SuiteCloud Plus module. See Using Multiple Script Queues to Run Scheduled Scripts for more information.

Scheduled Script Deployment Statuses

The following definitions and diagram show each of the deployment phases for a scheduled script. These phases appear in the Status field of the Script Deployment page.

- **In Queue**: The script is in the NetSuite scheduling queue and is waiting to be run.

- **In Progress**: The script is currently running.

- **Completed**: The script was successfully deployed and executed. The user must manually change the script’s deployment status to either **Scheduled** or **Not Scheduled** before the script can be run again.


- **Scheduled**: If the script has been set to execute on a recurring basis, the script’s deployment status will appear as **Scheduled** after it executes. The script will then be re-run at its specified time(s).
**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.

A script’s deployment status set to **Scheduled** or **Not Scheduled** determines its path of execution in NetSuite.

1. **SCHEDULED** deployment status. Scripts can be executed at recurring times or just once at some point in the future.

When the script deployment status is **In Queue** or **In Progress**, the deployment is locked. The script cannot be modified when it is in either of these statuses. However, the script can be canceled from the Scheduled Script Status page. See Using the Scheduled Script Status Page.

2. **In Queue**

3. **In Progress**

4. **Completed**

5. **Scheduled**

If the script has been set to execute on a recurring basis, the script’s deployment status appears as **Scheduled** after execution. The script will then be re-run at its specified time(s).

If the script was scheduled to run only once at a pre-defined time, the script’s deployment status appears as **Completed** after execution.
These figures show an example of the **Not Scheduled** -> **Completed** execution process. Notice the deployment status changes from Not Scheduled to Completed once it has finished its execution.

**Figure 1**

Even for scripts that fail, the deployment status still changes to **Completed**. Note that for another immediate (ad-hoc) execution, users must manually change the deployment status back to **Not Scheduled**. Clicking **Save and Execute** for a script that has a Completed status will do nothing.

**Figure 2**

**Important:** Scripts with the a **Completed** deployment status CAN be put into the NetSuite scheduling queue programmatically using `nlapiScheduleScript`. See **Running Scheduled Scripts Using nlapiScheduleScript** for details.

**Deployment Status and Script Execution Summary**

The following table summarizes what you can and cannot do with a scheduled script depending on its deployment status.

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`.

<table>
<thead>
<tr>
<th>Deployment Status</th>
<th>UI</th>
<th>nlapiScheduleScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Scheduled</td>
<td><strong>YES.</strong> You must click the Save and Execute button on the Script Deployment page.</td>
<td><strong>YES</strong></td>
</tr>
<tr>
<td>Scheduled</td>
<td><strong>N/A.</strong> No action is required by the user. With a deployment status set to Scheduled, the script will just run again at the next scheduled time.</td>
<td><strong>NO</strong></td>
</tr>
<tr>
<td>Deployment Status</td>
<td>UI</td>
<td>nlapiScheduleScript</td>
</tr>
<tr>
<td>-------------------</td>
<td>----</td>
<td>---------------------</td>
</tr>
<tr>
<td>Completed</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Testing</td>
<td>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.</td>
<td>NO</td>
</tr>
</tbody>
</table>
Deploying Scheduled Scripts

Scheduled scripts can be deployed to run immediately, or they can be scheduled to run at a later time. You can also create multiple deployments for the same scheduled script file. See these sections for details:

- Scheduling a Script for Immediate Execution
- Scheduling a Script to Run in the Future
- Creating Multiple Deployments for a Scheduled Script

You can set deployment options for scheduled scripts on both the Script record page or the Script Deployment page. For simplicity, all examples in the remaining sections will show screenshots from the Script Deployment page.
Scheduling a Script for Immediate Execution

Scheduled scripts with a deployment status of **Not Scheduled** or **Testing** can be executed immediately.

**Important:** Deployments set to **Not Scheduled** can also be invoked on-demand using the `nlapiScheduleScript` API. For details, see Running Scheduled Scripts Using `nlapiScheduleScript`.

To execute a scheduled script on-demand, set the deployment status to either **Not Scheduled** or **Testing** for the following reasons:

<table>
<thead>
<tr>
<th>Deployment Status</th>
<th>Use Case</th>
</tr>
</thead>
</table>
| Testing           | Set to Testing for the following reasons:  
  - 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 **Save and Execute** on the Script Deployment page to run the script.  
  - 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. |
| Not Scheduled     | 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.  
  **Important:** If you set your scheduling options, set the deployment status to **Not Scheduled**, and simply click **Save**, the script will not run at the times you may have specified.  
  Scripts with a deployment status set to Not Scheduled will only run if you click **Save and Execute** (or the scripts are placed into the NetSuite scheduling queue via `nlapiScheduleScript`). |

**To schedule a script for immediate execution:**

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, click the Deployed checkbox if it is not already checked, and select **Not Scheduled** from the Status drop-down (see figure).
7. Click the **Single Event** radio button.
8. Next, click **Save and Execute**.
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 immediate (ad-hoc) execution, NetSuite automatically sets the deployment status to Completed. Regardless of any additional Daily, Weekly, etc., deployment times that might have been specified, the script cannot execute again until its deployment status is manually changed from Completed to either Not Scheduled, Testing, or Scheduled. See the diagram in Scheduled Script Deployment Statuses for details.
Scheduling a Script to Run in the Future

Scheduled scripts with a deployment status set to **Scheduled** can be set to run once at a pre-defined time in the future, or they can be scheduled to run on a more regular daily, weekly, monthly, or yearly basis. Note that deployment times can only be scheduled in 30 minute intervals, for example 2:00 pm., 2:30 pm., 3:00 pm.

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 schedule a script to run at a scheduled time or recurring times:

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 function.
4. Next, click Save.
5. Click the **Deploy Script** button on the page that appears.
6. When the Script Deployment page opens, click the Deployed checkbox if it is not already checked, and select **Scheduled** from the Status drop-down (see figure).
7. On the Schedule tab, set all deployment options (see figure).
8. Click Save.
Creating Multiple Deployments for a Scheduled Script

On either the Script record 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

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. At the same time, you might also want this script deployed and executed every Monday morning at 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 to five by purchasing NetSuite’s SuiteCloud Plus module. See Using Multiple Script Queues to Run Scheduled Scripts 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.
To edit or access each deployment, go to Setup > Customization > Script Deployments. The following figure shows the Script Deployments list page, which lists all deployments in your account.

Creating additional deployments

You can easily 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 as 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 \texttt{nlapiScheduleScript}, see \textit{Running Scheduled Scripts Using \texttt{nlapiScheduleScript}}. Also see the API documentation for \texttt{nlapiScheduleScript(scriptId, deployId, params)}.
Running Scheduled Scripts Using `nlapiScheduleScript`

You can programmatically place a scheduled script into your company’s scheduling queue using the `nlapiScheduleScript` API. Be sure that the script’s deployment status appears as either Not Scheduled or Completed 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.

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 requeued manually through the Script Deployment page. See Example 2 in Scheduled Script Samples for code that shows how to programmatically requeue 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.

**Setting the Scheduled Script type Argument**

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.

**Related Topics**

- Understanding Scheduled Script Execution
- Scheduled Script Best Practices
- Setting Recovery Points in Scheduled Scripts
- Understanding Memory Usage in Scheduled Scripts
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 (ie., 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

```javascript
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 simply 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 the Memory Footprint of your Scripts**

The table below shows an estimation of how various parts of a script can consume memory.
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.

<table>
<thead>
<tr>
<th>Empty Function</th>
<th>111 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each Instruction within a function</td>
<td>32 bytes</td>
</tr>
<tr>
<td>Each local variable reference</td>
<td>32 bytes</td>
</tr>
<tr>
<td>Standard mode record (customer)</td>
<td>40 kilobytes (depends on record type)</td>
</tr>
<tr>
<td>Dynamic mode record (customer)</td>
<td>460 kilobytes (depends on record type)</td>
</tr>
<tr>
<td>Number Instance</td>
<td>32 bytes</td>
</tr>
<tr>
<td>String Instance</td>
<td>32 bytes + 4 bytes per character</td>
</tr>
<tr>
<td>Empty Object</td>
<td>32 bytes</td>
</tr>
</tbody>
</table>

Related Topics

- Scheduled Scripts
- Understanding Scheduled Script Execution
- Scheduled Script Samples
- Setting Recovery Points in Scheduled Scripts
Monitoring Scheduled Scripts

You can monitor scheduled script execution by:

- Using the Scheduled Script Status Page
- Using the Scheduled Scripts Calendar Page
- Using APIs to Work with Scheduled Scripts

Using the Scheduled Script Status Page

The Scheduled Script Status page shows the current and past statuses of all scheduled scripts that have been executed in your account. Access this page by going to Setup > Customization > Script Deployments > Status.

Note: Script execution details are purged after 30 days.

Notice that the Status column displays values that are different from the deployment status values. The values that appear in the Status column reflect internal NetSuite scheduled script work queue statuses, NOT deployment statuses.

The following table provides a mapping of these status types:

<table>
<thead>
<tr>
<th>Script work queue statuses (as they appear on the Scheduled Script Status page)</th>
<th>Script deployment statuses (as they appear on the Scheduled Deployment page)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pending</td>
<td>In Queue</td>
</tr>
<tr>
<td>Processing</td>
<td>In Progress</td>
</tr>
<tr>
<td>Deferred</td>
<td>In Queue</td>
</tr>
<tr>
<td>Complete</td>
<td>Completed or Scheduled</td>
</tr>
<tr>
<td>Failed</td>
<td>Completed or Scheduled</td>
</tr>
<tr>
<td>Retry</td>
<td>In Progress</td>
</tr>
</tbody>
</table>

Using the Scheduled Scripts Calendar Page

The Scheduled Scripts calendar page shows you all scheduled scripts that are set to deploy (in both Testing and Release mode) during a given day, week, or month. To access this page, go to
a schedule script's Script page or Script Deployment page and click the Go To Calendar crosslink (see figure).

If you are on the day view of the calendar, you can create a new scheduled script to run on that day by clicking the New Scheduled Script button at the bottom of the page. If you are on the week or month views, you can create a new scheduled script by clicking the plus (+) icon that appears in each day field of the week or month.

**Using APIs to Work with Scheduled Scripts**

The following APIs are helpful when working with and monitoring scheduled scripts:

- `nlapiLogExecution(type, title, details)`
- `nlobjContext methods:`
  - `setPercentComplete(pct)`
  - `getPercentComplete()`
  - `getRemainingUsage()`
  - `getScriptId()`
  - `getDeploymentId()`

**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.

```javascript
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' : 'cashsale';
        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.

```javascript
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 simple 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
• custscript_converted 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.

```javascript
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( custscript_converted );
}

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 phononenumber = 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 && phononenumber != null )
    {
        var call = nlapiCreateRecord('phonecall');
        call.setFieldValue('title','Follow up Call for '+leadName);
        call.setFieldValue('startdate', today );
        call.setFieldValue('assigned', salesrep );
        call.setFieldValue('phone', phononenumber );
        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.

```javascript
/****************************************************
* 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', context.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.getRemainingUsage());

//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.getText('entity'), subject, body);
    nlapiLogExecution('DEBUG', 'Remaining usage after sending thank you email', context.getRemainingUsage());
}
}
}

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);
    var params = new Array();
    params['custscriptstartdate'] = startDate.toUTCString();
    params['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);
}
Using Multiple Script Queues to Run Scheduled Scripts

All companies that run NetSuite are provided a single queue for running their scheduled scripts. As part of the SuiteCloud Plus offering, which is available for purchase, companies can upgrade their number of scheduled script queues from one to five. This offering allows larger accounts to divide their scheduled script work into categories such as script type, script length, department, and so on.

Important: There are never more than five scheduled script queues regardless of how many SuiteCloud Plus licenses are purchased.

When you upgrade your account to include five 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 five queues, the scripts will execute concurrently. For example, if you have one script in each of the five queues, all scripts will run at the same time.

After purchasing the SuiteCloud Plus module, you will notice that a Queue dropdown field is added to the Script Deployment record for scheduled scripts. 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.

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 add-on module, 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 this new add-on module, contact your NetSuite account manager.

Setting the Scheduled Script queue Argument

Once you have purchased the SuiteCloud Plus add-on module, you can use the queue argument to obtain the queue number assigned to the scheduled script. The queue argument is the second argument passed to the launch function of a scheduled script. (The first argument is the type argument. For details see, Setting the Scheduled Script type Argument.)

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.

Valid values for the *queue* argument are: 1, 2, 3, 4, 5, as there are five queues offered through SuiteCloud Plus.

Example

```javascript
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
}
```

Related Topics

- Scheduled Scripts
- Understanding Scheduled Script Execution
- Scheduled Script Samples
Chapter 24 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 portlets). 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 simple 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). Once 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.
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 simply write:

```javascript
function mySamplePortlet(portlet) {
    remainder of portlet script...
}
```

Portlet scripts can only run after users have added the scripts to their dashboards. Once 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
**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:**

```javascript
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 nlobjSearchColumn('trandate');
    returncols[1] = new nlobjSearchColumn('tranid');
    returncols[2] = new nlobjSearchColumn('entity');
    returncols[3] = new nlobjSearchColumn('salesrep');
    returncols[4] = new nlobjSearchColumn('amount');
    var results = nlapiSearchRecord('estimate', null, new
        nlobjSearchFilter('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 nlobjPortlet for a list of portlet object methods.

**Note:** If you are viewing this sample through the NetSuite Help Center, be sure to widen the main frame until there is no longer a horizontal scrollbar under the sample. In some browsers, the scrollbar may end up cutting off the last line of the sample.

**Script:**

```javascript
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', 'customdeploy_simpleform'), 'Submit');
}
**HTML Portlet**

This portlet script generates the HTML required to download and display a FLASH animation in an HTML portlet. See `nlobjPortlet` for a list of portlet object methods.

**Script:**

```javascript
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'></OBJECT>";
    "<OBJECT CLASSID='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'><PARAM NAME='MOVIE' VALUE='/images/flash/tomato.swf'></OBJECT>";
    "<embed src='/images/flash/tomato.swf'></embed>";
    "</tr></table>;"
    content = '<td><span>'+ content + '</span></td>';
    portlet.setHtml( content );
}
```

**Links Portlet**

This script makes an external request to slashdot.org in order 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:**

```javascript
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;
}
Chapter 25 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 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 direct list 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:

```javascript
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. Once 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.
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. Once 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 direct list edit
- Updating a field that is not available through direct list edit
- Updating a field based on a script parameter value

Updating a field that is available through direct list 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 direct list editing. In SuiteScript, fields that are direct list editable are updated using the `nlapiSubmitField(type, id, fields, values, doSourcing)` function.

```javascript
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 direct list 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 direct list edited through the UI. For example mass update scripts that update fields that are not available through direct list edit, see Updating a field that is not available through direct list edit.

Related Topics

- What Are Mass Update Scripts?
- Mass Update Script Execution
- Mass Update Scripts Samples
Updating a field that is not available through direct list 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.

Once 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.

```javascript
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 in order 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 direct list edited. In other words, you cannot simply 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 direct list 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.

**Note:** The screenshots in this section depict the NetSuite user interface prior to Version 2010 Release 2.

**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.

```javascript
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 Retail (see figure).

7. Next, as with any mass update, use tabs on the Mass Update page to:
   a. Define which records the custom mass update will apply to (Criteria tab).
   b. Define which records you want to see when you preview the custom mass update (Results tab).
   c. 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.
d. 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.

Related Topics
- What Are Mass Update Scripts?
- Mass Update Script Execution
- Mass Update Scripts Samples
Chapter 26 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.

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 Admin box should be checked 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.
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

Once 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 or the SuiteScript plug-in for Eclipse, 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 [TP=LIST_MEDIAITEMFOLDER=TP], 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

Once 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 [TP=EDIT_SCRIPT=TP], and click Bundle Installation Script.

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 in your script: Before Install, After Install, Before Update, After Update, or Before Uninstall.

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 Bundle Installation Script Deployment

Once 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 Setup > Customization > Script Deployments > [deployment] > 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

[FD=CREATESUITEBUNDLES=FD]

Once 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 [TP=TRAN_BUNDLEBUILDER=TP].
   - If you are editing an existing bundle, go to Setup > Customization > 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](#).
Once the bundle has been saved, this script record and related file(s) are listed as Bundle Components on the Bundle Details page.

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
• Sample Bundle Installation Script Record

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.

```javascript
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.');
    }
}

Sample Bundle Installation Script Record
Once script.js files are created, they must be uploaded to the NetSuite file cabinet. For information about this process, see Add the Bundle Installation Script File to the File Cabinet.
Then a bundle installation script record can be created. The following screenshot shows this record. For steps to create this type of record, see Create the Bundle Installation Script Record.

Related Topics
- Script Types Overview
- What are Bundle Installation Scripts?
- Setting Up a Bundle Installation Script
- Creating a Bundle
- Using the Bundle Builder
- Using Bundle Installation Scripts
Part 5  Setting Runtime Options
Chapter 27 Setting Runtime Options
Overview

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 as Admin
- 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
- Using the Script Execution Log
Chapter 28 Setting Script Execution Event Type from the UI

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.

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 easily specify a script execution context right at the time of script deployment – without having to modify your .js file. Once 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:

```javascript
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......
}
Chapter 29 Setting Script Execution Log Levels

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).

![Script Deployment Page](image)

**Note:** See Using the Script Execution Log for details on how to further customize your view of all log entries.

The Log Level field is essentially used as a simple 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 given 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.
Chapter 30 Executing Scripts as Admin

In the **Execute as Admin** check box on the Script Deployment page, select the check box if you want the script to execute using administrative privileges, regardless of the permissions of the currently logged-in user.

**Note:** The Execute as Admin check box appears on the Script Deployment page for all script types except the global client script type.

For information about role restrictions in client SuiteScript, see [Role Restrictions in Client SuiteScript](#).

Also, be aware that only users with the administrator role have access to the Execute as Admin checkbox. This checkbox is disabled for all others 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 Admin checkbox. This is also true for the Execute As Admin checkbox that appears on the summary page in SuiteFlow.

There are scripts which may require that they run with administrative privilege. For example, if you have a script that creates follow-up tasks once 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 Execute as Admin.

All bundle installation scripts need to Execute as Admin, so this option should always be enabled for this type of script deployment.

However, use of Execute as Admin 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 Execute as Admin whenever any meaningful interaction with the system is required.
Chapter 31 Setting Available Without Login

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.

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.

<table>
<thead>
<tr>
<th>Suitelet URL Type</th>
<th>Deployment Status</th>
<th>Select All check box</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>internal</td>
<td>Testing</td>
<td>not checked</td>
<td>Suitelet deploys successfully</td>
</tr>
<tr>
<td>internal</td>
<td>Testing</td>
<td>checked</td>
<td>Suitelet deploys successfully</td>
</tr>
<tr>
<td>internal</td>
<td>Released</td>
<td>not checked</td>
<td>Error message: You do not have privileges to view this page.</td>
</tr>
<tr>
<td>internal</td>
<td>Released</td>
<td>checked</td>
<td>Suitelet deploys successfully</td>
</tr>
<tr>
<td>external</td>
<td>Testing</td>
<td>not checked</td>
<td>Error message: You are not allowed to navigate directly to this page.</td>
</tr>
<tr>
<td>external</td>
<td>Testing</td>
<td>checked</td>
<td>Error message: You are not allowed to navigate directly to this page.</td>
</tr>
<tr>
<td>external</td>
<td>Released</td>
<td>checked</td>
<td>Suitelet deploys successfully</td>
</tr>
<tr>
<td>external</td>
<td>Released</td>
<td>not checked</td>
<td>Error message: You do not have privileges to view this page.</td>
</tr>
</tbody>
</table>
Chapter 32 Setting Script Deployment Status

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 Scheduled Script Deployment Statuses or Deployment Status and Script Execution Summary.

**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.
Chapter 33 Defining Script Audience

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 in order 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 once all audience members are 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). Once you determine that the script runs as expected for the Support Management role, you can change the script’s deployment status to Released. Once 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.
Part 6  Creating Script Parameters (Custom Fields)
In the context of SuiteScript, script parameters are essentially just 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.
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 allows you to easily 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 they 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.
Chapter 36 Creating Script Parameters

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 Setup > 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 black and accept a system-generated ID. (It is 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 **Custom Field Types** 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).
Creating Script Parameters

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.

![Simple Form](image)

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 (see below), 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 (see below). 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.
Chapter 37 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:

```javascript
//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.
Chapter 38 Setting Script Parameter Preferences

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 (see figure 1), 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 (see figure 1), 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.

- **Portlet**: If the preference is set to Portlet, the parameter’s value is read from a user’s personal portlet preferences.

- **<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 (see figure 2).

**Note**: See Creating Script Parameters for steps on creating a script parameter. Also see Referencing Script Parameters for information on accessing script parameter values.

Figure 1
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.

Figure 2

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 SuiteBundler Overview 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.

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).
Setting Script Parameter Preferences

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 **Setting Bundle Object Preferences**
Part 7 Searching with SuiteScript
Chapter 39 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
Chapter 40 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 - used to define search return columns for the search
3. nlobjSearchResult - used to get the values of specific search results

Once 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:

```javascript
var filters = new Array();
filters[0] = new nlobjSearchFilter('companyname', null, 'startswith', 'A');
```

**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 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:

```javascript
var columns = new Array();
columns[0] = new nlobjSearchColumn('entity');
columns[1] = new nlobjSearchColumn('phone');
columns[2] = new nlobjSearchColumn('companyname');
```

**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:

```javascript
// 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.

```javascript
// 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]);
Chapter 41 Search Samples

The following are samples of SuiteScript searches.

- Creating Saved Searches
- Executing 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

```javascript
// 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' );

Executing 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. This allows you to 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).

**Note:** For general information on NetSuite saved searches, see 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.

**Tip:** 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 Table](image)

In the following code, a Customer saved search is executed. The ID customsearch57 references a specific saved search.

**Note:** The second parameter in `nlapiSearchRecord(...)` will be 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.

```javascript
function executeSavedSearch()
{
   // 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 searchresult = searchresults[ i ];
   }
}
```
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 probability = 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
Search Samples
Filtering a Search

350

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);

SuiteScript Developer and Reference Guide


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.

```javascript
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,

```javascript
searchFilters[0] = new nlobjSearchFilter('class', null, 'noneof', '@NONE@');
```

In the following example, only customer records that match the entityid of test1 are returned.

```javascript
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 given filter variable, try performing a search that returns the value of a field as a result to 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 a saved customer search are returned.

```javascript
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:** In order to increase 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` 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.

```javascript
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 Warranty 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.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');
    }
}

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');
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` 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, null, 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.

In SuiteScript, this looks like:

```javascript
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.

```javascript
//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).

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.

**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.

```javascript
var x = nlapiSearchRecord('item', null, new nlobjSearchFilter('itemid', null, 'haskeywords', itemidvalue));
```

**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 once 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 `nlapiSearchDetection(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 Duplicate Record Detection in the NetSuite Help Center.
Performing Global Searches

NetSuite's global search allows you to 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 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:

```javascript
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

```javascript
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)
}

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')
    }
}
Chapter 42 Supported Search Operators, Summary Types, and Date Filters

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

The following table lists each field type and its supported search operator.

<table>
<thead>
<tr>
<th>Search Operator</th>
<th>Currency, Decimal Number, Time of Day</th>
<th>Date</th>
<th>Check Box</th>
<th>Document, Image</th>
<th>Email Address, Free-Form Text, Long Text, Password, Percent, Phone Number, Rich Text, Text Area, Multi Select</th>
<th>List/Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>after</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allof</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>any</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>anyof</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>before</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>between</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>contains</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>doesnotcontain</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>doesnotstartwith</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>equalto</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>greaterthan</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>greaterthanorequalto</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Search Summary Types

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.

<table>
<thead>
<tr>
<th>Search Operator</th>
<th>Currency, Decimal Number, Time of Day</th>
<th>Date</th>
<th>Check Box</th>
<th>Document, Image</th>
<th>Email Address, Free-Form Text, Long Text, Password, Percent, Phone Number, Rich Text, Text Area, Multi Select</th>
<th>List/Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>is</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isempty</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>isnot</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>isnotempty</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lessthan</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lessthanorequalto</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>noneof</td>
<td>[ ]</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>notafter</td>
<td>[ ]</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>notallof</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>notbefore</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>notbetween</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>notempty</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>notequalto</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>notgreaterthan</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>notgreaterthanorequalto</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>notlessthan</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>notlessthanorequalto</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>noton</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>notonorafter</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>notonorbefore</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>notwithin</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>onorafter</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>onorbefore</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>startswith</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>within</td>
<td>[ ]</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary Type</td>
<td>Type Internal ID</td>
<td>Purpose</td>
<td>Example</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>---------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td>group</td>
<td>Rolls up search results under the column to which you apply this type.</td>
<td>In a search for sales transactions, you can group the transactions found by customer name.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Count</td>
<td>count</td>
<td>Counts the number of results found that apply to this column.</td>
<td>In a search of items purchased by customers, you can view a count of the number of items purchased by each customer.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td>sum</td>
<td>Sums search results.</td>
<td>In a search of purchases this period, you can total the Amount column on your results.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>min</td>
<td>Shows the minimum amount found in search results.</td>
<td>In a search of sales transactions by sales rep, you can show the minimum amount sold in the transaction.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum</td>
<td>max</td>
<td>Shows the maximum amount found in search results.</td>
<td>In a customer search by partner, you can show the maximum amount of sales by each partner.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>avg</td>
<td>Calculates the average amount found in your search results.</td>
<td>In an employee search, you can average the amounts of your employees' company contributions.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Search Date Filters**

The following table lists all supported search date filters.

**Note:** For relative date filters (filters that end with ago and fromnow), append the desired time value to the filter to define the time period. For example, daysago15 is 15 days ago from today or daysfromnow5 is 5 days from today.

### Search Date Filter Values

- daysago
- daysfromnow
- lastbusinessweek
- lastfiscalquarter
- lastfiscalquartertodate
- lastfiscalyear
### Search Date Filter Values

- `lastfiscalyeartodate`  
- `lastmonth`  
- `lastmonthtodate`  
- `lastrollingquarter`  
- `lastrollingyear`  
- `lastweek`  
- `lastweektodate`  
- `monthsago`  
- `monthsfromnow`  
- `nextbusinessweek`  
- `nextfiscalquarter`  
- `nextfiscalyear`  
- `nextfourweeks`  
- `nextmonth`  
- `nextonemonth`  
- `nextonequarter`  
- `nextoneweek`  
- `nextweek`  
- `previousoneday`  
- `previousonemonth`  
- `previousonequarter`  
- `previousoneweek`  
- `previousoneyear`  
- `previousrollingquarter`  
- `previousrollingyear`  
- `quartersago`  
- `quartersfromnow`  
- `samemonthlastfiscalquarter`  
- `samemonthlastfiscalquartertodate`  
- `samemonthlastfiscalyear`  
- `samemonthlastfiscalyeartodate`  
- `samequarterlastfiscalyear`  
- `samequarterlastfiscalyeartodate`  
- `thisbusinessweek`  
- `thisfiscalquarter`
### Search Date Filter Values

- `thisfiscalquarter`todate
- `thisfiscalyear`
- `thisfiscalyeartodate`
- `thismonth`
- `thismonthtodate`
- `thisrollingquarter`
- `thisrollingyear`
- `thisweek`
- `thisweektodate`
- `thisyear`
- `today`
- `tomorrow`
- `weeksago`
- `weeksfromnow`
- `yearsago`
- `yearsfromnow`
- `yesterday`
Part 8  Working with UI Objects
Chapter 43 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. The following figure is a simple 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.

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

```javascript
/**
 * 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 = nlapiGetUser();
    var subject = request.getParameter('subject');
    var recipient = request.getParameter('recipient');
    var message = request.getParameter('message');
    nlapiSendEmail(currentuser, recipient, subject, message);
}
}
Chapter 44 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 nlobjFormUI 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 custpagecustomtab or custpagemytab.

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 [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.
Chapter 45 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 develop 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.

An example of an application implemented with a hybrid of inlineHTML and UI objects is a blog hosted within NetSuite. A blog page displays blog entries in descending chronological order with “Read More” hyperlinks. Due to the potentially large number of entries, pagination is required. Readers should also be able to read and leave comments. These requirements cannot be easily 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.
Welcome to our Blog

Some exciting products coming in the pipeline

Thanks to our R&D team's efforts, we're excited to give everyone a sneak peek of some very cool prod... (Read more)

Comments(11)
Author: Terry Chan
Date Created: 1/15/2008 12:16 pm

Re-cap of the Sales Team's kick-off

Yesterday we had the kick-off meeting for the sales team at the Metro Convention Hall. Thank you ev... (Read more)

Comments(0)
Author: Terry Chan
Date Created: 1/15/2008 12:14 pm
Chapter 46 Building a NetSuite Assistant with UI Objects

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 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 > Web Site > 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 Setup > Customization > 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, once 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.
Building a NetSuite Assistant with UI Objects

Using UI Objects to Build an Assistant

Related Topics
- Understanding NetSuite Assistants
- Using UI Objects to Build an Assistant
- UI Objects
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 deplomentId 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 once they have completed other tasks, you must append the customwhence parameter to the URL you are redirecting to.

```javascript
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. Once they complete the assistant they have been linked to, they will simply 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 be will 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.

```javascript
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")
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.

```javascript
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.

The following figure shows an inlineeditor sublist on Step 3 of an assistant 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 simple 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.)

Once the script is deployed, you can launch the assistant Suitelet by simply 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.

```javascript
/**
 * 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)
{
    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 important company information in the fields below.");
    assistant.addStep('companypreferences', 'Setup Company Preferences').setHelpText("Setup your important company preferences in the fields below.");
    assistant.addStep('enterlocations', 'Enter Locations').setHelpText("Add Locations to your account. 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 company data.");
    assistant.addStep('configurepricing', 'Configure Pricing').setHelpText("Configure your item pricing.");
    assistant.addStep('summary', 'Summary Information').setHelpText("Summary of your Assistant Work. 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 initial 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>: 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>: 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 assistant
            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('endrow');
                assistant.getField('orgtype', 'b2b').setDefaultValue('b2b');
                assistant.addField('companysizelabel', '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').setMandatory(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');

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);
    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 Customer 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 ','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 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", null, "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( host + 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?", null, "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('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');

assistant.addField('companyname', 'text','Company Name', null, 'companysummary').setDisplayType('inline');
assistant.addField('city', 'text', 'City', null, 'companysummary').setDisplayType('inline');
assistant.addField('abbreviatestates', 'checkbox', 'Use State Abbreviations in Addresses', 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('orgtype'));
assistant.getField('companysize', 's').setDefaultValue(ciStep.getFieldValue('companysize'));
assistant.getField('companyname').setDefaultValue(ciStep.getFieldValue('companyname'));
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 page */
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) altogether 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() == "next")
    {
        // update the company information page
        var configCompInfo = nlapiLoadConfiguration(‘companyinformation’);
        configCompInfo.setFieldValue( ‘city’, request.getParameter(‘city’) ) ;
        nlapiSubmitConfiguration( configCompInfo );
    }
    if (assistant.getLastStep().getName() == "companypreferences" && assistant.getLastAction() == "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;
}
Part 9  Debugging SuiteScript
Chapter 47 Debugging SuiteScript

Debugging Overview

You can use the SuiteScript Debugger to debug server scripts. Server script types are user event, Suitelet, scheduled, and portlet scripts.

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.

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 Debugger)
2. SuiteScript Debugger metering and permission restrictions (see 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 Using the Script Execution Log.

Using the SuiteScript Debugger

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 that already has a defined SuiteScript deployment. These scripts include User Event, Portlet, Scheduled, and Suitelet scripts. The status of these scripts must be set to Testing before they can be loaded into the Debugger. You must also be the owner of the script.
Related Topics

- Before Using the Debugger
- SuiteScript Debugger Interface
- Debugger Metering and Permissions
Chapter 48 Before Using the Debugger

Before using the SuiteScript Debugger, you must be aware of the following:

• There are three separate Debugger domains, accessible by going to Setup > Customization > 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:
    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 have a Beta account and choose to run the Debugger in this environment, go to this Debugger domain.
    Note that if you go to Setup > Customization > 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 that if you go to Setup > Customization > 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.

Once you are logged on to a Debugger domain, you will 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 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 Setup > Customization > 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 have a Beta account and choose to run the Debugger in this 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 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.

3. Next, click the Debug Script button.
   The script is immediately loaded into the Debugger and the program's execution stops at the first line of executable code. In this example, the first line of executable code is:

   ```javascript
   var g = 100;
   ```
Note that this sample includes an `f1()` calling function at the bottom of the script. Without a calling function, this script will simply compile with nothing to execute.

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.
4. 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.
### Related Topics

- Debugger Metering and Permissions
- Using the SuiteScript Debugger
- Deployed Debugging
- SuiteScript Debugger Interface
Chapter 50 Deployed Debugging

Deployed-mode debugging is for testing and inspecting scripts that have a defined SuiteScript deployment—in other words, you have already set the deployment for this script on the Script Deployment page (see figure below).

**To use the Debugger in deployed-mode:**

1. Go to Setup > Customization > 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 have a Beta account and choose to run the Debugger in this 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 Debugger to learn about how the changes you make to your account on a Debugger domain can affect your production, Beta, or sandbox account.
2. Once you are on a Debugger domain, navigate to the script’s deployment page and verify the script status is set to Testing. If you wish to debug a script that has already been released into production, you must change the script status from Released to Testing.

3. Verify that you are the assigned owner of the script. You can only debug scripts in which 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, you can navigate to the Debugger by going to Setup > Customization > 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 User Event, Portlet, Suitelet, and Scheduled scripts that are available for debugging (see figure below). Note that only scripts 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 checkbox field called Export to OpenAir to true.

![Script Debugger](image)

7. After clicking Select and Close, the Waiting for User Action screen appears (see figure below). To load User Event, Portlet, and Suitelet scripts into the Debugger, you must perform the action that calls the script.

**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.
8. 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 as it loads.

- The first figure shows that a new Customer will be created in a separate window.
- The second figure shows the setOpenAirField.js file as it loads into the Script Debugger window.
- The third figure shows a new Customer record in a separate window (or separate tab). Note the **Export to OpenAir** field defaults to true (meaning that the checkbox is automatically selected).
Debugging User Event: Set OpenAir Field (Customer)

```javascript
function setOpenAirField(type, form, request) {
    // User event script on Customer. Sets web address field on beforeload
    nlapiSetFieldValue('customerexport_url_openair', '');
}

setOpenAirField('type', 'form', 'request');
```
9. With the code in the Debugger text area, you have the following options:
   a. Click the Step Over button to begin stepping through each line of code.
   b. Add watches and evaluate expressions.
   c. Set break points and click the Run button to run the code. The Debugger will stop code execution at the first break point set.
   d. 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.

10. After testing/debugging a deployed-mode 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.
**Important:** 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. You must then re-load the updated .js file into the NetSuite file cabinet.

**Related Topics**
- Debugger Metering and Permissions
- Using the SuiteScript Debugger
- Ad-hoc Debugging
- SuiteScript Debugger Interface
Chapter 51 Debugger Metering and Permissions

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 once 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):

If you receive this message and you are in deployed-debugging mode, click the Go Back button. You must then reload your script by clicking the Debug Existing button (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.

Related Topics
- Before Using the Debugger
- Using the SuiteScript Debugger
- SuiteScript Debugger Interface
Chapter 52 SuiteScript Debugger Interface

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**

There are five Debugger buttons that can be used to control/resume script execution once the Debugger stops at a particular line:

- **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 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.
Note: Keyboard shortcuts are enabled for all five Debugger buttons. See Debugger Keyboard Shortcuts for details.

SuiteScript Debugger Tabs

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.

**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.

Due to performance considerations, only the first 50 entries of an Array are displayed. Also, only the first 500 characters of a String are displayed.

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.

**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 simply 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:**

- To add a break point, click between the line number and the actual line of code (see figure below).
- 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).
Related Topics

- SuiteScript Debugger Buttons
- Before Using the Debugger
- Using the SuiteScript Debugger
Chapter 53 Debugger Keyboard Shortcuts

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.

<table>
<thead>
<tr>
<th>Key</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Step Into</td>
</tr>
<tr>
<td>space</td>
<td>Step Over</td>
</tr>
<tr>
<td>o</td>
<td>Step Out</td>
</tr>
<tr>
<td>shift+space</td>
<td>Continue</td>
</tr>
<tr>
<td>q</td>
<td>Quit</td>
</tr>
<tr>
<td>d</td>
<td>Debug Script (New)</td>
</tr>
<tr>
<td>a</td>
<td>Debug Script (Existing)</td>
</tr>
<tr>
<td>r</td>
<td>Re-run Script</td>
</tr>
</tbody>
</table>

Related Topics
- Before Using the Debugger
- Using the SuiteScript Debugger
- SuiteScript Debugger Interface
## Chapter 54 Debugger Glossary

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line Break Point</td>
<td>User-defined line in source code where program halts execution</td>
</tr>
<tr>
<td>User Event Break Point</td>
<td>User event possibly invokable during script execution where the program halts execution</td>
</tr>
<tr>
<td>Watch</td>
<td>A variable expression that will be monitored throughout the program's execution in the current scope</td>
</tr>
<tr>
<td>Call Stack</td>
<td>A stack (most recent on top) of all the active functions (and their local variables) called up until the current line of execution</td>
</tr>
</tbody>
</table>
Part 10  SuiteScript API
Chapter 55 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.
Chapter 56 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 sublists 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 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
- 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

**Record APIs**

<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlapiAttachRecord(type, id, type2, id2, attributes)</td>
</tr>
<tr>
<td>nlapiCopyRecord(type, id, initializeValues)</td>
</tr>
<tr>
<td>nlapiCreateCSVImport()</td>
</tr>
<tr>
<td>nlapiCreateRecord(type, initializeValues)</td>
</tr>
<tr>
<td>nlapiDeleteRecord(type, id)</td>
</tr>
<tr>
<td>nlapiDetachRecord(type, id, type2, id2, attributes)</td>
</tr>
<tr>
<td>nlapiGetNewRecord()</td>
</tr>
<tr>
<td>nlapiGetOldRecord()</td>
</tr>
<tr>
<td>nlapiGetRecordId()</td>
</tr>
<tr>
<td>nlapiGetRecordType()</td>
</tr>
<tr>
<td>nlapiLoadRecord(type, id, initializeValues)</td>
</tr>
<tr>
<td>nlapiMergeRecord(id, baseType, baseld, altType, altid, fields)</td>
</tr>
<tr>
<td>nlapiMergeTemplate(id, baseType, baseld, altType, altid, fields)</td>
</tr>
<tr>
<td>nlapiSubmitCSVImport(nlobjCSVImport)</td>
</tr>
<tr>
<td>nlapiSubmitRecord(record, doSourcing, ignoreMandatoryFields)</td>
</tr>
<tr>
<td>nlapiTransformRecord(type, id, transformType, transformValues)</td>
</tr>
<tr>
<td>nlapiPrintRecord(type, id, mode, properties)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlobjCSVImport object and all methods</td>
</tr>
<tr>
<td>nlobjRecord object and all methods</td>
</tr>
</tbody>
</table>

**Subrecord APIs**

<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlapiCreateCurrentLineItemSubrecord(sublist, fldname)</td>
</tr>
<tr>
<td>nlapiCreateSubrecord(fldname)</td>
</tr>
<tr>
<td>nlapiEditCurrentLineItemSubrecord(sublist, fldname)</td>
</tr>
<tr>
<td>nlapiEditSubrecord(fldname)</td>
</tr>
<tr>
<td>nlapiRemoveCurrentLineItemSubrecord(sublist, fldname)</td>
</tr>
<tr>
<td>nlapiRemoveSubrecord(fldname)</td>
</tr>
</tbody>
</table>
SuiteScript Functions Overview

Field APIs

Functions

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

Objects

- `nlobjField` object and all methods
- `nlobjRecord` and all methods
- `nlobjSelectOption` and all methods

Sublist APIs

Functions

- `nlapiCancelLineItem(type)`
- `nlapiCommitLineItem(type)`
- `nlapiDisableLineItemField(type, fldnam, val)`
- `nlapiFindLineItemField(type, fldnam, val)`
- `nlapiFindLineItemMatrixValue(type, fldnam, val, column)`
- `nlapiFindLineItemValue(type, fldnam, val)`
- `nlapiGetCurrentLineItemIndex(type)`
**SuiteScript Functions**

### SuiteScript Functions Overview

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlapiGetCurrentLineItemMatrixValue(type, fldnam, column)</td>
<td>Get current line item matrix value</td>
</tr>
<tr>
<td>nlapiGetCurrentLineItemText(type, fldnam)</td>
<td>Get current line item text</td>
</tr>
<tr>
<td>nlapiGetCurrentLineItemValue(type, fldnam)</td>
<td>Get current line item value</td>
</tr>
<tr>
<td>nlapiGetCurrentLineItemValues(type, fldnam)</td>
<td>Get current line item values</td>
</tr>
<tr>
<td>nlapiGetLineItemCount(type)</td>
<td>Get line item count</td>
</tr>
<tr>
<td>nlapiGetLineItemField(type, fldnam, linenum)</td>
<td>Get line item field</td>
</tr>
<tr>
<td>nlapiGetLineItemMatrixField(type, fldnam, linenum, column)</td>
<td>Get line item matrix field</td>
</tr>
<tr>
<td>nlapiGetLineItemMatrixValue(type, fldnam, linenum, column)</td>
<td>Get line item matrix value</td>
</tr>
<tr>
<td>nlapiGetLineItemText(type, fldnam, linenum)</td>
<td>Get line item text</td>
</tr>
<tr>
<td>nlapiGetLineItemValue(type, fldnam, linenum)</td>
<td>Get line item value</td>
</tr>
<tr>
<td>nlapiGetLineItemValues(type, fldname, linenum)</td>
<td>Get line item values</td>
</tr>
<tr>
<td>nlapiGetMatrixCount(type, fldnam)</td>
<td>Get matrix count</td>
</tr>
<tr>
<td>nlapiGetMatrixField(type, fldnam, column)</td>
<td>Get matrix field</td>
</tr>
<tr>
<td>nlapiGetMatrixValue(type, fldnam, column)</td>
<td>Get matrix value</td>
</tr>
<tr>
<td>nlapiInsertLineItem(type, line)</td>
<td>Insert line item</td>
</tr>
<tr>
<td>nlapiInsertLineItemOption(type, fldnam, value, text, selected)</td>
<td>Insert line item option</td>
</tr>
<tr>
<td>nlapiIsLineItemChanged(type)</td>
<td>Check if line item is changed</td>
</tr>
<tr>
<td>nlapiRefreshLineItems(type)</td>
<td>Refresh line items</td>
</tr>
<tr>
<td>nlapiRemoveLineItem(type, line)</td>
<td>Remove line item</td>
</tr>
<tr>
<td>nlapiRemoveLineItemOption(type, fldnam, value)</td>
<td>Remove line item option</td>
</tr>
<tr>
<td>nlapiSelectLineItem(type, linenum)</td>
<td>Select line item</td>
</tr>
<tr>
<td>nlapiSelectNewLineItem(type)</td>
<td>Select new line item</td>
</tr>
<tr>
<td>nlapiSetCurrentLineItemMatrixValue(type, fldnam, column, value, firefieldchanged, synchronous)</td>
<td>Set current line item matrix value</td>
</tr>
<tr>
<td>nlapiSetCurrentLineItemText(type, fldnam, text, firefieldchanged, synchronous)</td>
<td>Set current line item text</td>
</tr>
<tr>
<td>nlapiSetCurrentLineItemValue(type, fldnam, value, firefieldchanged, synchronous)</td>
<td>Set current line item value</td>
</tr>
<tr>
<td>nlapiSetCurrentLineItemValues(type, fldnam, values, firefieldchanged, synchronous)</td>
<td>Set current line item values</td>
</tr>
<tr>
<td>nlapiSetLineItemValue(type, fldnam, linenum, value)</td>
<td>Set line item value</td>
</tr>
<tr>
<td>nlapiSetMatrixValue(type, fldnam, column, value, firefieldchanged, synchronous)</td>
<td>Set matrix value</td>
</tr>
</tbody>
</table>

### Objects

- 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)
### SuiteScript Functions Overview

**SuiteScript Functions**

#### Scheduling APIs

<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlapiScheduleScript(scriptId, deployId, params)</td>
</tr>
<tr>
<td>nlapiSetRecoveryPoint()</td>
</tr>
<tr>
<td>nlapiYieldScript()</td>
</tr>
</tbody>
</table>

#### Execution Context APIs

<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlapiGetContext()</td>
</tr>
<tr>
<td>nlapiGetDepartment()</td>
</tr>
<tr>
<td>nlapiGetLocation()</td>
</tr>
<tr>
<td>nlapiGetRole()</td>
</tr>
<tr>
<td>nlapiGetSubsidiary()</td>
</tr>
<tr>
<td>nlapiGetUser()</td>
</tr>
<tr>
<td>nlapiLogExecution(type, title, details)</td>
</tr>
</tbody>
</table>

#### Objects |

- nlobjContext object and all methods

#### UI Builder APIs

<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlapiCreateAssistant(title, hideHeader)</td>
</tr>
<tr>
<td>nlapiCreateForm(title, hideNavbar)</td>
</tr>
<tr>
<td>nlapiCreateList(title, hideNavbar)</td>
</tr>
</tbody>
</table>

### Objects

- nlapiSearchDuplicate(type, fields, id)
- nlapiSearchGlobal(keywords)
- nlapiSearchRecord(type, id, filters, columns)

#### Functions

- nlapiGetContext()
- nlapiGetDepartment()
- nlapiGetLocation()
- nlapiGetRole()
- nlapiGetSubsidiary()
- nlapiGetUser()
- nlapiLogExecution(type, title, details)
Application Navigation APIs

Functions
- `nlapiRequestURL(url, postdata, headers, callback, httpMethod)`
- `nlapiRequestURLWithCredentials(credentials, url, postdata, headers, httpMethod)`
- `nlapiResolveURL(type, identifier, id, displayMode)`
- `nlapiSetRedirectURL(type, identifier, id, editmode, parameters)`

Objects
- `nlobjRequest object and all methods`
- `nlobjResponse object and all methods`

Date APIs

Functions
- `nlapiAddDays(d, days)`
- `nlapiAddMonths(d, months)`
- `nlapiDateToString(d, format)`
- `nlapiStringToDate(str, format)`

Objects
- `nlobjAssistant object and all methods`
- `nlobjAssistantStep object and all methods`
- `nlobjButton object and all methods`
- `nlobjColumn object and all methods`
- `nlobjField object and all methods`
- `nlobjFieldGroup and all methods`
- `nlobjForm object and all methods`
- `nlobjList object and all methods`
- `nlobjPortlet object and all methods`
- `nlobjSubList object and all methods`
- `nlobjTab object and all methods`
## Currency APIs

**Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nlapiExchangeRate(sourceCurrency, targetCurrency, effectiveDate)</code></td>
<td></td>
</tr>
<tr>
<td><code>nlapiFormatCurrency(str)</code></td>
<td></td>
</tr>
</tbody>
</table>

## Encryption APIs

**Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nlapiEncrypt(s, algorithm, key)</code></td>
<td></td>
</tr>
</tbody>
</table>

## XML APIs

**Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nlapiEscapeXML(text)</code></td>
<td></td>
</tr>
<tr>
<td><code>nlapiSelectNode(node, xpath)</code></td>
<td></td>
</tr>
<tr>
<td><code>nlapiSelectNodes(node, xpath)</code></td>
<td></td>
</tr>
<tr>
<td><code>nlapiSelectValue(node, xpath)</code></td>
<td></td>
</tr>
<tr>
<td><code>nlapiSelectValues(node, path)</code></td>
<td></td>
</tr>
<tr>
<td><code>nlapiStringToXML(text)</code></td>
<td></td>
</tr>
<tr>
<td><code>nlapiXMLToString(xml)</code></td>
<td></td>
</tr>
<tr>
<td><code>nlapiXMLToPDF(xmlstring)</code></td>
<td></td>
</tr>
</tbody>
</table>

## File APIs

**Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nlapiCreateFile(name, type, contents)</code></td>
<td></td>
</tr>
<tr>
<td><code>nlapiDeleteFile(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>nlapiLoadFile(id)</code></td>
<td></td>
</tr>
<tr>
<td><code>nlapiSubmitFile(file)</code></td>
<td></td>
</tr>
</tbody>
</table>

**Objects**

- `nlobjFile object and all methods`
Error Handling APIs

**Functions**

- `nlapiCreateError(code, details, suppressNotification)`

**Objects**

- `nlobjError object and all methods`

Communication APIs

**Functions**

- `nlapiOutboundSSO(id)`
- `nlapiRequestURL(url, postdata, headers, callback, httpMethod)`
- `nlapiSendCampaignEmail(campaigneventid, recipientid)`
- `nlapiSendEmail(author, recipient, subject, body, cc, bcc, records, attachments)`
- `nlapiSendFax(author, recipient, subject, body, records, attachments)`

Configuration APIs

**Functions**

- `nlapiLoadConfiguration(type)`
- `nlapiSubmitConfiguration(name)`

**Objects**

- `nlobjConfiguration object and all methods`

SuiteFlow APIs

**Functions**

- `nlapiInitiateWorkflow(recordtype, id, workflowid)`
- `nlapiTriggerWorkflow(recordtype, id, workflowid, actionid)`

Portlet APIs

**Functions**

- `nlapiRefreshPortlet()`
- `nlapiResizePortlet()`
SuiteAnalytics APIs

Functions

- nlapiCreateReportDefinition()
- nlapiCreateReportForm(title)

Objects

- nlobjPivotColumn object and all methods
- nlobjPivotColumnHierarchy object and all methods
- nlobjPivotRow object and all methods
- nlobjPivotTable object and all methods
- nlobjPivotTableHandle object and all methods
- nlobjReportColumn object and all methods
- nlobjReportColumnHierarchy object and all methods
- nlobjReportDefinition object and all methods
- nlobjReportForm object and all methods
- nlobjReportRowHierarchy object and all methods

User Credentials APIs

Function

- nlapiGetLogin()

Objects

- nlobjLogin object and all methods

Record APIs

For an overview of NetSuite records, see Working with Records and Subrecords 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()
- nlapiCreateRecord(type, initializeValues)
- nlapiDeleteRecord(type, id)
- nlapiDetachRecord(type, id, type2, id2, attributes)
SuiteScript Functions

Record APIs

• nlapiGetNewRecord()
• nlapiGetOldRecord()
• nlapiGetRecordId()
• nlapiGetRecordType()
• nlapiLoadRecord(type, id, initializeValues)
• nlapiMergeRecord(id, baseType, baseId, altType, altId, fields)
• nlapiMergeTemplate(id, baseType, baseId, altType, altId, fields)
• nlapiSubmitCSVImport(nlobjCSVImport)
• nlapiSubmitRecord(record, doSourcing, ignoreMandatoryFields)
• nlapiTransformRecord(type, id, transformType, transformValues)
• nlapiPrintRecord(type, id, mode, properties)
• nlobjCSVImport
• nlobjRecord

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

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

• Issue attached to Support Case
• Contact attached to Customer|Partner|Vendor|Lead|Prospect|Project
• File attached to any transaction, item, activity, custom, or entity record
• Custom child record attached to supported parent record
• Entity to a static entity group. Note that if attaching an entity to 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, and Suitelet scripts.

Parameters

• type {string} [required] - The record internal ID name for the record being attached to. For a list of supported record types and their internal IDs, see SuiteScript Supported Records in the NetSuite Help Center. Note that if you want to attach a file from the file cabinet, set type to file.
• id {string} [required] - The record internalId for the record being attached to, for example 555 or 78.
• `type2` [string] [required] - The record internal ID for the record being attached. Note that if attaching an entity to a static entity group, the internal ID for the entity group record type is `entitygroup`.

• `id2` [string] [required] - The record internal ID for the record being attached.

• `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 an Issue record to a Support Case record.

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

```javascript
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). Once attached, the file will appear on the Files tab of the Customer record.

**Important:** Be aware that although the file internal ID is being reference 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.

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

**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 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 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 partner record from an existing one.
`var partner = nlapiCopyRecord('partner', 20)`
`partner.setFieldValue('entityid', 'New Partner')`

**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 simple (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:

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

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

//returns the internal id of the new job created in workqueue
var jobld = nlapiSubmitCSVImport(job);
```
This sample uses an internal ID to reference the import map and a CSV file internal ID to reference CSV data:

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

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

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

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

```javascript
var mappingFileId = "CUSTIMPORTentityMultiFile";

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

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

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

var jobId = nlapiSubmitCSVImport(job);
```

For more details about the methods used in these samples, see `nlobjCSVImport`.

---

**Back to Record APIs | Back to SuiteScript Functions**

### `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 client, user event, scheduled, portlet, and Suitelet scripts. See API Governance for the unit cost associated with this API.
**Note:** Values for all required fields **must** be provided or a newly instantiated record cannot be submitted. See 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.

```javascript
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.

```javascript
function createTaskRecord()
{
    var taskTitle = 'Follow up regarding new Opportunity';
    var record = nlapiCreateRecord('task');
}
```
Example 3

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

```javascript
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)`

Use this API to delete an existing record. 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 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

**Returns**

- void

**Throws**

- SSS_INVALID_RECORD_TYPE
- SSS_TYPE_ARG_REQD
- SSS_INVALID_INTERNAL_ID
- SSS_ID_ARG_REQD

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

**Example 1**

The following example deletes a specific task record in the system.
**Example 2**

In the next example a resultant record set from a customer saved search is deleted. Once 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.

```javascript
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());
    }
}
```

---

**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.

**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.

```javascript
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.

```javascript
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')
```
SuiteScript Functions
Record APIs

Back to Record APIs | Back to SuiteScript Functions

**nlapiGetNewRecord()**

Available in beforeLoad, beforeSubmit, and afterSubmit user event scripts. This API never reflects any changes set by the system during the submit operation. It only reflects the field and line item (sublist) values submitted by the user.

Note that you are not allowed to submit the current or previous record returned by nlapiGetNewRecord(). Also note that only fields that are changed (submitted) will be available via nlapiGetNewRecord().

**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.
**nlapiGetRecordType()**

Use this API to retrieve the record type of the current record in a user event script. This API is available to client and user event scripts only. If there is no current record type, the value of null will be returned.

**Returns**

- The string recordType name for the record whose form the user is currently on, or for the record that the current User Event script is executing on.

[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 client, user event, scheduled, portlet, and Suitelet scripts. 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 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`

**Parameters**

- `type {string} [required]` - The record internal ID name. 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. Once 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.

```javascript
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.

```javascript
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, baseId, altType, altId, fields)

Performs a mail merge operation and returns an nlobjFile object containing the results. This API is supported in user event, scheduled, and Suitelet scripts.

**Note:** There is a 5MB limitation to the size of the file that can be merged using this API. See API Governance for the unit cost associated with this API.

**Important:** This API supports the same records types that are supported in NetSuite’s mail merge feature. Supported record types include NetSuite transactions, entities, custom records, and support cases. Note that activity type records such as tasks and events are not currently supported. Also note that transaction column fields are not supported in NetSuite mail merge and therefore are not available through nlapiMergeRecord.

**Note:** With nlapiSendEmail(author, recipient, subject, body, cc, bcc, records, attachments) you can use NetSuite email templates to construct the body of the email using nlapiMergeRecord. The nlapiMergeRecord API performs a merge operation using a NetSuite email template and up to two business records.

**Parameters**
- *id* {int} [required] - The internalId of the email template
- *baseType* {string} [required] - The recordType for the primary record used in the merge
- *baseId* {int} [required] - The internalId for the primary record used in the merge
- *altType* {string} [optional] - The recordType for the secondary record used in the merge
- *altId* {int} [optional] - The internalId for the secondary record used in the merge
- *fields* {hashtable} [optional] - Array of NL tag names to data values (custom name-value merge fields used for merge operation). Note that tag names must begin with NL.

**Returns**
- An nlobjFile object containing the results of the merge operation

**Since**
- Version 2008.1

**Example**
The following sample is a user event script that is deployed on an afterSubmit event. This script applies to the Customer record type.

```javascript
function afterSubmit(type) {
    
```
```javascript
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);
}
```

**SuiteScript Developer and Reference Guide**

**SuiteScript Functions**

**Record APIs**

---

**nlapiMergeTemplate(id, baseType, baseId, altType, altId, fields)**

**THIS API HAS BEEN DEPRECATED**

This API was deprecated in NetSuite version 2008.1, however, it continues to be supported. This function will not be enhanced in future versions of NetSuite.

In its place, users can use `nlapiMergeRecord(id, baseType, baseId, altType, altId, fields)`, which performs exactly the same as `nlapiMergeTemplate`. The only distinction between the two functions is that `nlapiMergeRecord` returns a `nlobjFile` object (instead of a String).

**Deprecation Since**

- Version 2008.1

**See Also**

- `nlapiMergeRecord(id, baseType, baseId, altType, altId, fields)`

---

**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 Checking CSV Import Status.

Executing this API consumes 100 governance units.

---

**Back to** Record APIs | **Back to** SuiteScript Functions
Note that this API cannot be used to import data that is imported by simple (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
- `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, just 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:

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

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

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

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

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

var job = nlapiCreateCSVImport();
```
var mappingFileId = "CUSTIMPORTentityMultiFile";

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

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

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

var jobId = nlapiSubmitCSVImport(job);

For more details about the methods used in these samples, see nlobjCSVImport.
• **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 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.

```javascript
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.

```javascript
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 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.

**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 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:

```javascript
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_URLCATEGORY
- 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.

```javascript
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.

```javascript
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 given 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 5

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:

<table>
<thead>
<tr>
<th>Record Type</th>
<th>Record Name</th>
<th>Transform Type</th>
<th>Transform Name (Target Record)</th>
</tr>
</thead>
<tbody>
<tr>
<td>assemblyitem</td>
<td>Build/Assembly</td>
<td>assemblybuild</td>
<td>Assembly Build</td>
</tr>
<tr>
<td>assemblybuild</td>
<td>Assembly Build</td>
<td>assemblyunbuild</td>
<td>Assembly Unbuild</td>
</tr>
<tr>
<td>cashsale</td>
<td>Cash Sale</td>
<td>cashrefund</td>
<td>Cash Sale</td>
</tr>
<tr>
<td>customer</td>
<td>Customer</td>
<td>cashsale</td>
<td>Cash Sale</td>
</tr>
<tr>
<td>customer</td>
<td>Customer</td>
<td>customerpayment</td>
<td>Customer Payment</td>
</tr>
<tr>
<td>customer</td>
<td>Customer</td>
<td>estimate</td>
<td>Quote</td>
</tr>
<tr>
<td>customer</td>
<td>Customer</td>
<td>invoice</td>
<td>Invoice</td>
</tr>
<tr>
<td>customer</td>
<td>Customer</td>
<td>opportunity</td>
<td>Opportunity</td>
</tr>
<tr>
<td>customer</td>
<td>Customer</td>
<td>salesorder</td>
<td>Sales Order</td>
</tr>
<tr>
<td>employee</td>
<td>Employee</td>
<td>expensereport</td>
<td>Expense Report</td>
</tr>
<tr>
<td>employee</td>
<td>Employee</td>
<td>timebill</td>
<td>Time</td>
</tr>
<tr>
<td>estimate</td>
<td>Quote</td>
<td>cashsale</td>
<td>Cash Sale</td>
</tr>
<tr>
<td>estimate</td>
<td>Quote</td>
<td>invoice</td>
<td>Invoice</td>
</tr>
<tr>
<td>estimate</td>
<td>Quote</td>
<td>salesorder</td>
<td>Sales Order</td>
</tr>
<tr>
<td>invoice</td>
<td>Invoice</td>
<td>creditmemo</td>
<td>Credit Memo</td>
</tr>
<tr>
<td>invoice</td>
<td>Invoice</td>
<td>customerpayment</td>
<td>Customer Payment</td>
</tr>
<tr>
<td>invoice</td>
<td>Invoice</td>
<td>returnauthorization</td>
<td>Return Authorization</td>
</tr>
<tr>
<td>lead</td>
<td>Lead</td>
<td>opportunity</td>
<td>Opportunity</td>
</tr>
<tr>
<td>opportunity</td>
<td>Opportunity</td>
<td>cashsale</td>
<td>Cash Sale</td>
</tr>
<tr>
<td>opportunity</td>
<td>Opportunity</td>
<td>estimate</td>
<td>Quote</td>
</tr>
<tr>
<td>opportunity</td>
<td>Opportunity</td>
<td>invoice</td>
<td>Invoice</td>
</tr>
<tr>
<td>opportunity</td>
<td>Opportunity</td>
<td>salesorder</td>
<td>Sales Order</td>
</tr>
<tr>
<td>prospect</td>
<td>Prospect</td>
<td>estimate</td>
<td>Quote</td>
</tr>
<tr>
<td>prospect</td>
<td>Prospect</td>
<td>opportunity</td>
<td>Opportunity</td>
</tr>
<tr>
<td>prospect</td>
<td>Prospect</td>
<td>salesorder</td>
<td>Sales Order</td>
</tr>
<tr>
<td>purchaseorder</td>
<td>Purchase Order</td>
<td>itemreceipt</td>
<td>Item Receipt</td>
</tr>
<tr>
<td>purchaseorder</td>
<td>Purchase Order</td>
<td>vendorbill</td>
<td>Vendor Bill</td>
</tr>
<tr>
<td>purchaseorder</td>
<td>Purchase Order</td>
<td>vendorreturnauthorization</td>
<td>Vendor Return Authorization</td>
</tr>
<tr>
<td>returnauthorization</td>
<td>Return Authorization</td>
<td>cashrefund</td>
<td>Cash Refund</td>
</tr>
<tr>
<td>returnauthorization</td>
<td>Return Authorization</td>
<td>creditmemo</td>
<td>Credit Memo</td>
</tr>
</tbody>
</table>
## SuiteScript Functions

### Record APIs

<table>
<thead>
<tr>
<th>Record Type</th>
<th>Record Name</th>
<th>Transform Type</th>
<th>Transform Name (Target Record)</th>
</tr>
</thead>
<tbody>
<tr>
<td>returnauthorization</td>
<td>Return Authorization</td>
<td>itemreceipt</td>
<td>Item Receipt</td>
</tr>
<tr>
<td>salesorder</td>
<td>Sales Order</td>
<td>cashsale</td>
<td>Cash Sale</td>
</tr>
<tr>
<td>salesorder</td>
<td>Sales Order</td>
<td>invoice</td>
<td>Invoice</td>
</tr>
<tr>
<td>salesorder</td>
<td>Sales Order</td>
<td>itemfulfillment</td>
<td>Item Fulfillment</td>
</tr>
<tr>
<td>salesorder</td>
<td>Sales Order</td>
<td>returnauthorization</td>
<td>Return Authorization</td>
</tr>
<tr>
<td>salesorder</td>
<td>Sales Order</td>
<td>revenuecommitmentreversal</td>
<td>Revenue Commitment Reversal Note: The return authorization must be approved and received for this transform to work.</td>
</tr>
<tr>
<td>transferorder</td>
<td>Transfer Order</td>
<td>itemfulfillment</td>
<td>Item Fulfillment</td>
</tr>
<tr>
<td>vendor</td>
<td>Vendor</td>
<td>vendorbill</td>
<td>Vendor Bill</td>
</tr>
<tr>
<td>vendor</td>
<td>Vendor</td>
<td>vendorcredit</td>
<td>Vendor Credit</td>
</tr>
<tr>
<td>vendor</td>
<td>Vendor</td>
<td>vendorpayment</td>
<td>Vendor Payment</td>
</tr>
<tr>
<td>vendor</td>
<td>Vendor</td>
<td>vendorreturnauthorization</td>
<td>Vendor Return Authorization</td>
</tr>
<tr>
<td>vendor</td>
<td>Vendor Return Authorization</td>
<td>itemfulfillment</td>
<td>Item Fulfillment</td>
</tr>
<tr>
<td>vendor</td>
<td>Vendor Return Authorization</td>
<td>vendorcredit</td>
<td>Vendor Credit</td>
</tr>
<tr>
<td>workorder</td>
<td>Work Order</td>
<td>assemblybuild</td>
<td>Assembly Build</td>
</tr>
</tbody>
</table>

**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 5MB 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)` 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.

**Parameters**

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

- `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.

```javascript
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.

```javascript
function printStatement()
{
  //create an array to set the STATEMENT properties
  var sdate = new Array();
  sdate.startdate = '02/07/2008';
  sdate.statemendate = '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.

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

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.

Important: Subrecords are used in the context of the Advanced Bin / Numbered Inventory Management feature. Currently, the only supported subrecord in NetSuite is the Inventory Details subrecord. For scripting examples that show how to use subrecord APIs in the context of this feature, see Using SuiteScript with Advanced Bin / Numbered Inventory Management.

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.

This API is currently used only in the context of the Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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**

See Creating a subrecord in the NetSuite Help Center.
**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 currently supported in form-level or record-level client SuiteScripts associated with the parent record.

This API is currently used only in the context of the Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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

**Example**

See Creating a subrecord in the NetSuite Help Center.

---

**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.

This API is currently used only in the context of the Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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

See Editing a 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 associated with the parent record.

This API is currently used only in the context of the Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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

Example

See Editing a subrecord in the NetSuite Help Center.
### nlapiRemoveCurrentLineItemSubrecord(sublist, fldname)

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

This API is currently used only in the context of the Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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**

- `void`

**Since**

- Version 2011.2

---

### 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 Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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`
SuiteScript Functions

Subrecord APIs

Since

• Version 2011.2

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 Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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

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 Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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

---

**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 Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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.

Back to Subrecord APIs | Back to SuiteScript Functions

**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(fldnam, 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 given 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} [required] - If set to true the field is disabled. If set to false it is enabled.
  **Important:** The values for this parameter can be true or false, not T or F.

**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, which then allows you to 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. The 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 name of the field being set

**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.

```javascript
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.

Parameters

- `fldnam` [string] [required] - The internal ID name of the field being set

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.

```javascript
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 name 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 checkbox, the return value will be `F` if the field is unset.
This API is available in client and user event scripts only.

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 name 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:

```javascript
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

### `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} [optional] - If not set, this argument defaults to false. If set to `true`, the select option becomes the default option. **Important:** The values for this parameter can be `true` or `false`, not `T` or `F`.

Returns

- `void`

### `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.

```javascript
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.

```javascript
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 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 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 given body field. This API can be used in user event `beforeLoad` scripts to initialize field 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. **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. Once the value is set for `entity`, the value of the Sales Rep (`salesrep`) field synchronously sources, and an alert is thrown to identify the Sales Rep. If the value of the `synchronous` parameter had not been set to `true` for `nlapiSetFieldValue`, there is a possibility that the alert would be thrown **before** it included the sales rep ID. With `synchronous` set to `true`, the alert cannot be thrown **until** the `salesrep` field data has been correctly sourced from the `entity` field.

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

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

Back to Field APIs | Back to SuiteScript Functions

**nlapiSetFieldValues (fldnam, value, firefieldchanged, synchronous)**

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

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

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

**Parameters**

- **fldnam** [string] [required] - The internal ID name of the field being set
- **value** [string] [required] - The value the field is being set to (Array).
- **firefieldchanged** [boolean] [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.

  **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

Example

```javascript
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} [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. **Note:** `doSourcing` takes the values of `true` or `false`, not `T` or `F`.

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.

```javascript
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 `nlapiSubmitField` in the context of a Suitelet.

```javascript
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 `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.

**Returns**

- `void`

---

**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 `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.

**Returns**

- `void`
nlapiDisableLineItemField(type, fldnam, val)

Sets the given line item field of a sublist to disabled or enabled based on the value (true or false).

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 line item field to enable/disable
- **val** {boolean} [required] - If set to true the field is disabled. If set to false it is enabled.

Important: The values for this parameter can be true or false, not T or F.

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 given 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 given 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.

```javascript
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.

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 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

```javascript
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 `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.

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.

```javascript
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.

**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 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)**

Retrieves 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 the field does not exist.

---

**nlapiGetCurrentLineItemValues(type, fldnam)**

Retrieves 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 (on the currently selected line)

**Since**

- Version 2012.1

---

**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(group)`.

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

Returns

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

Example

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

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

nlapiGetLineItemField(type, fldnam, linenum)

Use this function to obtain sublist (line item) field metadata. Calling this function instantiates the nlobjField object, which then allows you to 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 `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 sublist field
- `linenum` {int} [required] - 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.

```javascript
function clientSideScript(type, form) {
    var field = nlapiGetLineItemField('item', 'quantity');
    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, which then allows you to 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.

```javascript
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.

**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
- **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 simple client script that throws an alert with the value of the `myText` variable.

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

**Sublist APIs**

---

**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:** 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 addressbook as the ID for the Address sublist). In the NetSuite Help Center, see [Scriptable Sublists](https://help netsuite com) 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 addressbook as the ID for the Address sublist). In the NetSuite Help Center, see [Scriptable Sublists](https://help netsuite com) for a list of sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
### 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](https://help.netsuite.com/) 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](https://help.netsuite.com/) 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.

```javascript
function getCount() {
    // Code goes here
}
```
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**

- nllobjField 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. Once you call nlapiGetMatrixField() you can use all the methods on the nllobjField object to get whatever field metadata you might need.

```javascript
function getMatrixHeaderInfo()
{
  var qtyObject = nlapiGetMatrixField('price', 'price', 2);
  var fieldLabel = qtyObject.getLabel();
}
```
var fieldName = qtyObject.getName();
var fieldType = qtyObject.getType();

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

Back to Sublist APIs | Back to SuiteScript Functions

**nlapiGetMatrixValue(type, fldnam, column)**

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

**Note:** Currently the Pricing sublist and the Demand Plan Detail sublist are the only matrix sublist types that support SuiteScript, and this API is used only for the Pricing sublist. For details on working with the Pricing sublist, see 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 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.

- 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 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 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} [optional] - If not set, this argument defaults to false. If set to true, the selected option will become the default selection. **Note:** The values for this parameter are **true** or **false**, not T or F.

**Returns**

- **void**

---

**nlapiIsLineItemChanged(type)**

Determines whether any changes have been made to a sublist

**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.

**Returns**

- **Returns true** if the currently selected line of the sublist has been edited
**nlapiRefreshLineItems(type)**

Makes a server call in order 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 *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.

**Returns**

- void

**Back to Sublist APIs | Back to SuiteScript Functions**

**nlapiRemoveLineItem(type, line)**

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

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

**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.

- *line* {int} [required] - The line number you want to remove (only valid for User Event scripts and Suitelets). 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 *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 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 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.

• 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 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.

Returns

• void

Example

```javascript
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);
nlapiSetCurrentLineItemValue('item', 'location', 102);

// 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 given 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.

```javascript
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 `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
- `text` {string} [required] - The display name associated with the value that the field is being set to
SuiteScript Functions

Sublist APIs

SuiteScript Developer and Reference Guide

497

• 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 given 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 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

• 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

---

**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 `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 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.

```javascript
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 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
• **linenum** [int] [required] - The line number for this field. Note the first line number on a sublist is 1 (not 0).
• **value** [string] [required] - The value the field is being set to

Returns

• **void**

Back to Sublist APIs | Back to SuiteScript Functions

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

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

**Note:** Currently the Pricing sublist and the Demand Plan Detail sublist are the only matrix sublist types that support SuiteScript, and this API is used only for the Pricing sublist. For details on working with the Pricing sublist, see 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.

**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. **Note:** 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] [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.

```javascript
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');
}
```
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 just changed, a risk of creating an unending loop exists as follows:

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

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

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

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

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

The following API calls can set the firefieldchanged parameter.

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

- nlapiSetFieldValue(fldnam, value, firefieldchanged, synchronous)
- nlapiSetFieldText(fldname, txt, firefieldchanged, synchronous)
- nlapiSetCurrentLineItemValue(type, fldnam, value, firefieldchanged, synchronous)
• `nlapiSetCurrentLineItemText(type, fldnam, text, firefieldchanged, synchronous)`

**Note:** The `firefieldchanged` parameter is provided for convenience. To prevent this loop, you could also include code that either checks to ensure that you are not writing the same value to the field or that tracks whether you just 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 reused later in the UI or using `nlapiLoadSearch(type, id)`.

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

**Parameters**

- `type` `{string} [required]` - The record internal ID of the record type you are searching (for example, `customer|lead|prospect|partner|vendor|contact`). For a list of internal IDs, in the NetSuite Help Center see SuiteScript Supported Records.
- `filters` `{nlobjSearchFilter | nlobjSearchFilter[] | Object[]} [optional]` - A single `nlobjSearchFilter` object - or - an array of `nlobjSearchFilter` objects - or - a search filter expression.
Note: You can further filter the returned nlobjSearch object by passing additional filter values. You will do this using the nlobjSearch.addFilter(filter) method or nlobjSearch.addFilters(filters) method.

- columns {nlobjSearchColumn or nlobjSearchColumn[]} [optional] - A single nlobjSearchColumn object - or - an array of nlobjSearchColumn 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(...).

```javascript
// 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.

```javascript
var search = nlapiLoadSearch('opportunity', 'customsearch_blackfriday');
var newSearch = nlapiCreateSearch(search.getSearchType(), search.getFilters(), search.getColumns());
newSearch.addFilter(new nlobjSearchFilter(...)); //Specify your own criteria here to add as a filter
newSearch.setIsPublic(true);
newSearch.saveSearch('My new opp search', 'customsearch_blacksaturday');
```
Example 3

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

```javascript
//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.

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

Back to Search APIs | Back to SuiteScript Functions

**nlapiLoadSearch(type, id)**

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

Executing this API consumes 5 governance units.

**Parameters**

- `type` {string} [optional] - The record internal ID of the record type you are searching (for example, customer|lead|prospect|partner|vendor|contact). For a list of internal IDs, in the NetSuite Help Center see SuiteScript Supported Records.
SuiteScript Functions

Search APIs

SuiteScript Developer and Reference Guide

506

- **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.

```javascript
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.

```javascript
var search = nlapiLoadSearch('opportunity', 'customsearch_blackfriday');
var newSearch = nlapiCreateSearch(search.getSearchType(), search.getFilters(), search.getColumns());
newSearch.addFilter(new nlobjSearchFilter(...));
newSearch.setIsPublic(true);
newSearch.saveSearch('My new opp search', 'customsearch_blacksaturday');
```

**Example 3**

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

In the following figure, a user selects 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 once 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. Once 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 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.

SuiteScript Developer and Reference Guide

 SuiteScript Functions

 Search APIs

SuiteScript Developer and Reference Guide
var searchID = nlapiGetFieldValue('custentity_mysavedsearch');
var results = nlapiSearchRecord(null, searchID);

nlapiLookupField(type, id, fields, text)

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

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|address|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.

Example

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

```javascript
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();
}
```

Back to Search APIs | Back to SuiteScript Functions

### 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

- `{nlobjSearchResult[]} - 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`.

```javascript
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:

```
[UE=HELP_TOPIC_OUT_OF_DATE_ALERT=UE]
```

### Global Search: Results

<table>
<thead>
<tr>
<th>Edit</th>
<th>User</th>
<th>Type</th>
<th>Name/ID</th>
<th>Additional Info 1</th>
<th>Additional Info 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit</td>
<td>User</td>
<td>Customer</td>
<td>Abe Simpson</td>
<td><a href="mailto:jdsj@netsuke.com">jdsj@netsuke.com</a></td>
<td>504-231-1111(san) 955-1234(ab)</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Customer</td>
<td>Abe Simpson</td>
<td><a href="mailto:jdsj@netsuke.com">jdsj@netsuke.com</a></td>
<td>504-231-1111(san) 955-1234(ab)</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Customer</td>
<td>Abe Simpson</td>
<td><a href="mailto:jdsj@netsuke.com">jdsj@netsuke.com</a></td>
<td>504-231-1111(san) 955-1234(ab)</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Opportunity</td>
<td>OP12001</td>
<td>Abe Simpson</td>
<td>23.1.2003</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Opportunity</td>
<td>OP12008</td>
<td>Abe Simpson</td>
<td>23.1.2003</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Opportunity</td>
<td>OP12005</td>
<td>Abe Simpson</td>
<td>Remodeling</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Opportunity</td>
<td>OP12007</td>
<td>Abe Simpson</td>
<td>25.7.2006</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Opportunity</td>
<td>OP12006</td>
<td>Abe Simpson</td>
<td>25.7.2006</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Opportunity</td>
<td>OP12001</td>
<td>Abe Simpson</td>
<td>25.7.2006</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Opportunity</td>
<td>OP12002</td>
<td>Abe Simpson</td>
<td>25.7.2006</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Opportunity</td>
<td>OP12008</td>
<td>Abe Simpson</td>
<td>25.7.2006</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Opportunity</td>
<td>OP12009</td>
<td>Abe Simpson</td>
<td>25.7.2006</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Opportunity</td>
<td>OP12010</td>
<td>Abe Simpson</td>
<td>25.7.2006</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Opportunity</td>
<td>OP12011</td>
<td>Abe Simpson</td>
<td>25.7.2006</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Opportunity</td>
<td>OP12012</td>
<td>Abe Simpson</td>
<td>25.7.2006</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Project</td>
<td>Abe Simpson</td>
<td><a href="mailto:jdsj@netsuke.com">jdsj@netsuke.com</a></td>
<td>504-231-1111(san) 955-1234(ab)</td>
</tr>
<tr>
<td>Edit</td>
<td>User</td>
<td>Project</td>
<td>Abe Simpson</td>
<td><a href="mailto:jdsj@netsuke.com">jdsj@netsuke.com</a></td>
<td>504-231-1111(san) 955-1234(ab)</td>
</tr>
</tbody>
</table>

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:

```javascript
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 [Global Search](https://help.netsuite.com/2022-08/app/help/GlobalSearch) in the NetSuite Help Center.

### 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](https://help.netsuite.com/2022-08/app/help/CustomListSearch) 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. However, you can accomplish sorting using either of the following methods:

1. Reference a saved search that is sorted by internalid or internalidnumber
2. 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 object - or - an array of nlobjSearchColumn objects. Note that you can further filter the returned saved search by passing additional search return column values.
Returns

- `{nlobjSearchResult[]}` - An array of `nlobjSearchResult` objects corresponding to the searched records.
  
  **Important:** The array returned by this API is read-only. Note that if there are no search results, `null` is returned.

Throws

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

Examples

For code samples showing the kinds of searches you can execute using the `nlapiSearchRecord` function, see 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` - 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)

Schedules a long-running script for immediate execution if its current status appears as Completed or Not Scheduled on the Script Deployment page. If the deployment status is set to Scheduled, the script cannot be executed immediately. The script will run at the time(s) specified on the Script Deployment page.

The nlapiScheduleScript API consumes 20 units per call. This API is supported in user event, 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 on running scheduled scripts in NetSuite, see Scheduled Scripts.

**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 will result in an error.

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).
  - size {integer} – The size of the saved object.
  - information {string} – Additional information about the status.

**Important:** If an 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
- nlobjFile
- 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.

```javascript
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
    }
}
```
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 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);
    }
}

checkGovernance();
function cleanUpMemory(){...set references to null, dump values seen in maps, etc}

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.

### 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).

• **size** {integer} – The size of the saved object.

• **information** {string} – Additional information about the status.

**Important:**

• 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 finally 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.

Back to Scheduling APIs | Back to SuiteScript Functions

---

**Execution Context APIs**

Context APIs are used to get system information or metadata about a script that is running, a user in a NetSuite account, or certain settings that have been applied to account.

All APIs listed below are in alphabetical order.

• **nlapiGetContext()**
• **nlapiGetDepartment()**
• **nlapiGetLocation()**
• **nlapiGetRole()**
• **nlapiGetSubsidiary()**
• **nlapiGetUser()**
• **nlapiLogExecution(type, title, details)**
• **nlobjContext**

---

**nlapiGetContext()**

Used to branch scripts depending on the metadata or context of the execution. For example, you may want the script to perform in one way when a form is accessed via the UI and another when the form is accessed via 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.

---

### 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)
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.

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.

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.

nlapiGetUser()

Returns the internalId of the current NetSuite user. This API is supported in client, user event, scheduled, portlet, and Suitelet scripts.
Returns

- The integer value of the current user (for example, 195, 25, 21). Note that the value of \(-4\) is returned if a user cannot be properly identified by NetSuite. This occurs when the user has not authenticated to NetSuite, for example when using externally available (Available without Login) Suitelets or online forms.

Example

The following sample shows how to use nlapiGetUser in conjunction with nlapiSendEmail. In this sample, the internal ID of the currently logged in user is passed to the \textit{author} argument in nlapiSendEmail, which is a required argument in this API.

```javascript
function afterSubmitEmail(type) {
    //User event script deployed to purchase orders.
    //Set the afterSubmit type to approve. As soon as the PO is
    //approved, an email is sent.
    if (type == 'approve') {
        //Get the user ID of the person approving the PO. This will be the email author.
        var userId = nlapiGetUser();

        //Send an email to the supervisor, K. Wolfe in this case.
        var sendEmail = nlapiSendEmail(userId, 'kwolfe@netsuite.com', 'Purchase Order Notification', 'Purchase order approved', null, null, 'transaction', null);
    }
}
```

See also

nlapiSendEmail(author, recipient, subject, body, cc, bcc, records, attachments)

Back to Execution Context APIs | Back to SuiteScript Functions

\textbf{nlapiLogExecution(type, title, details)}

This API is supported in Suitelet, scheduled, portlet, user event, 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 Using the Script Execution Log to learn more about writing logs to the Execution Log subtab.

\textbf{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 given 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 Using the Script Execution Log in the NetSuite Help Center.

**Parameters**

- *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: 3000 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.

```javascript
//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.

```javascript
//Search for the sales orders with trandate of today
var todaySO = nlapiSearchRecord('salesorder', null, todaySOFilters, todaySOColumns);
nlapiLogExecution('DEBUG', 'Remaining usage after searching sales orders from today', context.getRemainingUsage());
```

[Back to Execution Context APIs](#) | [Back to SuiteScript Functions](#)
---

### `nlobjContext`

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

[Back to Execution Context APIs](#) | [Back to SuiteScript Functions](#)
---

#### **UI Builder APIs**

UI builder APIs allow developers to programmatically create various components of 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)`
- `nlobjAssistant`
- `nlobjAssistantStep`
- `nlobjButton`
SuiteScript Functions

UI Builder APIs

SuiteScript Developer and Reference Guide

- nlobjColumn
- nlobjField
- nlobjFieldGroup
- nlobjForm
- nlobjList
- nlobjPortlet
- nlobjSubList
- nlobjTab

**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.

**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.

```javascript
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.addStep('entercontacts', 'Enter Contacts').setHelpText("Manually add contacts into your account.")
```
assistant.addStep('importdata', 'Import Data').setHelpText("Finally, import records into your account via CSV.");

---

**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 just 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

---

**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
- hideNav {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 just HTML.

When hideNav 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

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.
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)`
- `nlapiResolveURL(type, identifier, id, displayMode)`
- `nlapiSetRedirectURL(type, identifier, id, editmode, parameters)`
- `nlobjRequest`
- `nlobjResponse`

**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.

Also note that if you call `nlapiRequestURL(...)`, 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"

Additionally, `nlapiRequestURL(...)` calls from the server 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.

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)
- **postdata** {string | hashtable} [optional] - Body used for a POST request. It can either be an associative array of form parameters or a string. If set to **null**, then a GET request is used.
- **headers** [hashtable] [optional] - An associative array of name (header) value (header value) pairs
- **callback** {function} [optional] - A callback function called when the request is completed (Client SuiteScript only). If you specify a callback in Client SuiteScript, the request is processed asynchronously, and once 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:

```javascript
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. 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:

Throws

- SSS_REQUEST_TIME_EXCEEDED (if the connection exceeds the 45 second timeout period)

Example 1

Request an XML document from a server and also include a header.

```javascript
var a = new Array();
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

The next example shows how to make an asynchronous request.

```javascript
var a = new Array();
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()+'
';
    output += 'Headers:
';
    for ( var i in headers )
        output += i + ': '+headers[i]+'
';
    output += '

Body:

';
    output += response.getBody();
    alert( output );
}
```

Example 3

The final example shows how to make a request using a new browser window.

```javascript
nlapiRequestURL( 'https://webservices.netsuite.com/wsdl/v1_2_0/netsuite.wsdl', null, a, null);
```

Back to Application Navigation APIs | Back to SuiteScript Functions

**nlapiRequestURLWithCredentials(credentials, url, postdata, headers, httpMethod)**

Allows you to send credentials outside of NetSuite. This API securely accesses a handle to credentials 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 > 5 seconds and/or the request takes > 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"

Usage metering allowed is 10 units. This API is supported in client, user event, scheduled, portlet, and Suitelet scripts.

Parameters

- credentials {string} [required] - 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 will need a list. The handles are 32 byte, globally unique strings (GUIDs).
- url {string} [required] - The HTTP(s) URL being requested - (fully qualified unless NetSuite page)
- postdata {string | hashtable} [optional] - Body used for a POST request. It can either be an associative array of form parameters or a string. If set to null, then a GET request is used.
- headers {hashtable} [optional] - An associative array of name (header) value (header value) pairs
- httpMethod {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 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
Since

- Version 2012.1

Example

The following shows a general process for creating credential fields and then, later, getting their handles and passing them on using \texttt{nlapiRequestURLWithCredentials(...)}.

1. Two custom fields with username and passwords are added to a form:
   
   \begin{verbatim}
   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');
   \end{verbatim}

2. During a beforeSubmit user event, we obtain the values from credential fields and store them in two custom fields, which are not visible on the form:

   \begin{verbatim}
   var credValue = nlapiGetFieldValue('custpage_credname');
   var username = nlapiGetFieldValue('custpage_username');

   nlapiSetFieldValue('custentity_custompassword', credValue);
   nlapiSetFieldValue('custentity_customusername', username);
   \end{verbatim}

3. Before using the credentials, we copy them as a list in a variable. At this point, uname and pwd will contain the GUIDS (credentials handle):

   \begin{verbatim}
   var uname = rec.getFieldValue('custentity_customusername');
   var pwd = rec.getFieldValue('custentity_custompassword');
   var creds = [uname, pwd];
   \end{verbatim}

4. Use credentials in an external call:

   \begin{verbatim}
   var connect = nlapiRequestURLWithCredentials(creds, url);
   \end{verbatim}

---

\textbf{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 given record type. Refer to the \textit{SuiteScript Reference Guide} for a list of available NetSuite tasklinks.

This API is supported in client, user event, scheduled, portlet, and Suitelet scripts.
Note: You can also discover the Tasklink for a page within NetSuite by viewing the HTML page source. Search for a string similar to the following, where LIST_SCRIPT refers to the TASKLINK.

```html
onclick="nlPopupHelp('LIST_SCRIPT','help')"
```

Parameters

- `type {string} [required]` - The base type for this resource. These types include:
  - RECORD - Record Type
  - TASKLINK - Task Link
  - SUITELET - Suitelet

- `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 the 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, set `displayMode` to either `true` or `false` to return either an internal or external Suitelet URL. Set to `true` to return an external URL. Set to `false`, or simply omit the argument, to return an internal URL. For Suitelets, `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.

Throws

- SSS_INVALID_URLCATEGORY
- SSS_CATEGORARGREQD
- SSS_INVALIDTASKID
- SSS_TASKIDREQD
- SSS_INVALIDINTERNALID
- SSS_INVALIDEDITMODEARG
Example

The following lines of code show 5 different approaches for resolving to a record or Suitelet.

```javascript
//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 user event and Suitelet scripts.

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:** You cannot redirect a user to an external URL, unless the **type** parameter is set to **EXTERNAL**. See the documentation for this **type** (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)

• `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} [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. **Important**: The values for this parameter can be true or false, not T or F.

• `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.

```javascript
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.

```javascript
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 into a String using the current user's date format

**Parameters**
- `d {date} [required]` - Date object being converted into a String
- `format {string} [optional]` - Format type to use: date|timeofday with date being the default

**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 using the current user’s NetSuite date format, or to one of the formats passed in the format parameter. Be aware that leading zeroes in the month and day values are not supported.

Parameters

- str {string} [required] - String being converted to a Date
- format {string} [optional] - Use any one of the following format types:
  - datetimetz - converts the date that is passed to a datetimetz format
  - timeofday - converts the date that is passed to a timeofday format
  - datetime - converts the date that is passed to a datetime format

Returns

- Date object. Returns NaN if date includes a leading zero.

Example

```javascript
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

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 client, user event, scheduled, portlet, and Suitelet scripts.

When using this API, the first currency (sourceCurrency) is the one to look up relative to the second (targetCurrency), which MUST be a base currency. 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. Note that the targetCurrency must be the base currency. For information on setting a base currency in your NetSuite account, see Setting a Base Currency in the NetSuite Help Center. If the value you provide for targetCurrency parameter is not the base currency, null will be returned when the API is executed.

- effectiveDate [string|int] [optional] - If not supplied, then effectiveDate reflects the current date.

Returns

- The exchange rate (as a float) in the same precision that is displayed in the NetSuite UI. Note that null is returned if the targetCurrency is not set to a base currency.

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.

```javascript
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)`

Encrypts a clear text String using a SHA-1 hash function. This is the same encryption used for password fields.

**Parameters**

- `s` {string} [required] - String being encrypted
- `algorithm` {string} [optional] - algorithm to use
- `key` {string} [optional] - secret key to use
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)`
- `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'));

---

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

---

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[]

---

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**

---

### 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 5MB limitation to the size of the file that can be created.

The following list includes just 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

**Important:** For details on BFO, available tags, and BFO examples, see the following links:


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).

```javascript
function helloWorld()
{
    var xml = "<?xml version="1.0"?>
<!DOCTYPE pdf PUBLIC "://big.faceless.org/report" ":report-1.1.dtd">
<pdf>
<body font-size="18">
Hello World!
</body>
</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.

```javascript
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 += searchresult.getValue('unitprice', 'pricing');
strName += "<td>
strName += "<barcode codetype="code128‖ showtext="true‖ value=""
strName += searchresult.getValue('unitprice', 'pricing');
strName += ""/>
strName += "</td><tr>
strName += "</table>

// build up BFO-compliant XML using well-formed HTML
var xml = "<?xml version="1.0"?>
<!DOCTYPE pdf PUBLIC "-//big.faceless.org//pdfpdf-1.1.dtd">
";
xml += "<pdf>
<body font-size="12">
<h3>My Pricing List</h3>

xml += strName;
xml += "</body>
</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

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cleaning</td>
<td>24.00</td>
</tr>
<tr>
<td>Replacing of Dental Drill Head</td>
<td>125.00</td>
</tr>
<tr>
<td>Serving</td>
<td>18.00</td>
</tr>
<tr>
<td>Cotton Swabs</td>
<td>7.50</td>
</tr>
</tbody>
</table>

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.

```javascript
function main(Request, Response) {
    var xml = "<?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE pdf PUBLIC \"\\-//big.faceless.org\\/report\\\" \"report-1.1.dtd\">\n    <pdf lang="ru-RU" xml:lang="ru-RU">\n    <head>\n    <link name="russianfont" type="font" subtype="opentype" 
    src="NetSuiteFonts/verdana.ttf" 
    src-bold="NetSuiteFonts/verdanab.ttf" 
    src-italic="NetSuiteFonts/verdanai.ttf" 
    src-bolditalic="NetSuiteFonts/verdanabi.ttf" 
    bytes="2"/>
    </head>
    <body font-family="russianfont" font-size="18">
    <p>Russian: Русский текст</p>
    <p>Russian Italic: <i>Русский текст</i></p>
    <p>Russian Bold: <b>Русский текст</b></p>
    <p>Russian Bold Italic: <b><i>Русский текст</i></b></p>
    </body>
    </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%>`.

**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 5MB limitation to the size of the document that can be created using this API.

The `nlapiCreateFile` API can also be used for streaming to clients (via Suitelets). For streaming or attaching binary content, you can call the following. Note that each of these APIs can load or generate binary content, provided that the `contents` argument is **base-64** encoded.

- `nlapiLoadFile(id)`
- `nlapiPrintRecord(type, id, mode, properties)`
- `nlapiMergeRecord(id, baseType, baseId, altType, altId, fields)`

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

**Important:** Be aware that the `nlapiCreateFile` function does not support the creation of non-text file types such as PDFs, unless the `contents` argument is base-64 encoded.

**Parameters**

- `name` [string] [required] - The name of the file
SuiteScript Functions

File APIs

SuiteScript Developer and Reference Guide

•  

  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 simple text file to use as an email attachment. Note that once created, the file object will not be stored in the file cabinet.

```javascript
function sendAttachment()
{
  var newAttachment = nlapiCreateFile('helloworld.txt', 'PLAINTEXT', 'Hello World
Hello 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.

```javascript
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

---

### nlapiLoadFile(id)

Loads a file from the NetSuite file cabinet (using the file's internal ID or path). Returns an `nlobjFile` containing 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 user event, scheduled, portlet, and Suitelet scripts.

**Note:** There is a 5MB limitation to the size of the document that can be accessed using this API.

**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 will return the file as a NetSuite `nlobjFile` object, which allows you to use `nlobjFile` methods to interact with the file.

```javascript
function logEvent(type)
{
  var f = nlapiLoadFile('Images/logo_goat.jpg');
  if (f)
  {
    nlapiLogExecution('AUDIT', 'Event', 'Type:'+type+' file;'+f.getId());
  }
  else
  
    nlapiLogExecution('AUDIT', 'Event', 'No file;');
}
```
**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:** There is a 5MB limitation to the size of the document that can be submitted using this API.

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.

**nlobjFile**

See nlobjFile - defined in the section on Standard Objects.

**Error Handling APIs**

All APIs listed below are in alphabetical order.

- `nlapiCreateError(code, details, suppressNotification)`
- `nlobjError`
SuiteScript Functions

Communication APIs

SuiteScript Developer and Reference Guide

553

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] [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. **Note:** The values for this parameter can be true or false, not T or F.

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)
- 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. Note that this function works in conjunction
with the Lead Nurturing (campaigndrip) sublist only; it does not work with the E-mail (campaignemail) sublist.

Usage metering allowed is 10 units. 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. Once the email is sent, the sender can use the campaign response ID that is returned for tracking purposes.

```javascript
// 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, {recordmore: dynamic});
var campaign2_campaigndrip_internalid_1 = campaign2.getLineItemValue('campaigndrip', 'internalid', 1);

// 142 is a sample ID representing the ID of a recipient with a valid email address
var campaignResponseId = nlapiSendCampaignEmail(campaign2_campaigndrip_internalid_1, 142);

---

**nlapiSendEmail**(author, recipient, subject, body, cc, bcc, records, attachments)

Sends and records outgoing email to an individual or to a group of individuals. This function can be used for automatic email notifications of critical events or for message logging. Note that you can also send multiple attachments of any media type using this function. Email attachment size cannot exceed 5 MB. In other words, you can send as many attachments as you like, but collectively, all attachments cannot exceed 5 MB.

This function also allows email to be attached to custom records if the user references the custom record by either its internalId or scriptId.

Usage metering allowed is 10 units. This API is supported in client, user event, scheduled, portlet, and Suitelet scripts.

**Note:** You can use NetSuite email templates to construct the body of the email using nlapiMergeRecord(id, baseType, baseId, altType, altId, fields), which performs a merge operation using a NetSuite email template and up to two business records.

**Parameters**

- **author** {int} [required] - The internalId of an employee record (this is the sender). To get the internal ID for an employee, go Lists > Employees > Employees (you must have admin access to the account in order to access the Employees list page). The employee's ID will appear in the Internal ID column on the list page. Note, however, 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] - You can set the following for this parameter:
  - A single external email address, or
  - A list of external addresses (comma separated), or
  - 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.
**Important:** The `subject` argument is **required**. A JavaScript exception is thrown if this argument is left blank, set to `null`, or set to an empty string.

- `body` [string] [required] - Body of the outgoing email.
  **Important:** The `body` argument is required. 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

- `bcc` [string | string[]) [optional] - An array of email addresses or a single email address to blind copy.

- `records` [hashtable] [optional] - An associative array of internal records to associate/attach this email with. The following table lists valid keys -> values.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value (examples)</th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction</td>
<td><code>records['transaction'] = '1000';</code></td>
</tr>
<tr>
<td>(use for transaction and opportunity record types)</td>
<td></td>
</tr>
<tr>
<td>activity</td>
<td><code>records['activity'] = '50'</code></td>
</tr>
<tr>
<td>(use for Case and Campaign record types)</td>
<td></td>
</tr>
<tr>
<td>entity</td>
<td><code>records['entity'] = '555'</code></td>
</tr>
<tr>
<td>(use for all Entity record types, for example, customer, contact, etc.)</td>
<td></td>
</tr>
<tr>
<td>record</td>
<td><code>records['recordtype'] = 'customrecord11';</code></td>
</tr>
<tr>
<td>(custom record internalId - for custom records you must also specify both the record ID and the record type ID)</td>
<td></td>
</tr>
<tr>
<td>recordtype</td>
<td><code>records['recordtype'] = 'customrecord11';</code></td>
</tr>
<tr>
<td>(custom recordtype internalId or scriptId)</td>
<td></td>
</tr>
</tbody>
</table>

- `attachments` [nlobjFile | nlobjFile[]) [optional] - A single nlobjFile object - or - an array of nlobjFile objects to attach to outgoing email (**not** supported in Client SuiteScript).

**Returns**

- `void`

**Throws**

- `SSS_AUTHOR_MUST_BE_EMPLOYEE`
- `SSS_AUTHOR_REQD`
- `SSS_INVALID_RECIPIENT_ID`
- `SSS_MISSING_REQD_ARG`
- `SSS_RECIPIENT_REQD`
• SSS_INVALID_CC_EMAIL
• SSS_INVALID_BCC_EMAIL

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);

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

```javascript
// 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 use SuiteBundler to deploy your scripts into other accounts.

If you do not create a custom scriptId, a system-generated ID will be generated for you once the SuiteSignOn record is saved. You can also use the system-generated ID as the id value.

**Note:** Once the SuiteSignOn record is saved, both the scriptId and system-generated ID are prefixed with `customssso`.

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_SUITESIGNON_NOT_CONFIGURED
- SSS_INVALID_SUITESIGNON

**Since**

- Version 2009.2

**Example 1**

This sample shows how to use `nlapiOutboundSSO(id)` in a portlet script to create a reference to the SuiteSignOn record. Once the portlet is added to the dashboard, the script is executed. The
SuiteScript Functions
Communication APIs

560

value of the nlapiOutboundSSO variable is passed to an iframe, which makes the http request
to load the source.
// create a portlet object
function buildPortlet(portlet, column)
{
// set a portlet title
title = 'My Custom SSO Portlet!'
portlet.setTitle(title)
// pass the scriptId of the SuiteSignOn record
var url = nlapiOutboundSSO('customsso_wlf_sso_partner_portlet');
// create an iframe. It is the iframe that makes the http request to
// load the content of the portlet.
var content = '<iframe src="'+url+'" align="center" style="width: 100%; height: 600px;
margin:0; border:0; padding:0"></iframe>';
// render the content in your portlet
portlet.setHtml( content );
}

Example 2
This sample shows how to use nlapiOutboundSSO(id) in a Suitelet to create a reference to the
SuiteSignOn record. When the Suitelet opens and the content of the iframe is generated, the
URL specified on the SignSignOn record will render.
function buildSuitelet(request, response)
{
if ( request.getMethod() == 'GET' )
{
//create a form
var form = nlapiCreateForm('SSO Suitelet');
var label = form.addField('custpage_label', 'inlinehtml', 'SSO1');
label.setDefaultValue ('<B>Check out my SSO Suitelet!!</B>');
var url = nlapiOutboundSSO('customsso_wlf_sso_partner_suitelet');
var content = '<iframe src="'+url+'" align="center" style="width: 1000px; height: 800px;
margin:0; border:0; padding:0"></iframe>';
var iFrame = form.addField('custpage_sso', 'inlinehtml', '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

SuiteScript Developer and Reference Guide


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.

```javascript
function syncWithExternalApp(type) {
    var url = nlapiOutboundSSO('customsso_my_external_app');
    nlapiRequestURL(url);
}
```

Back to Communication APIs | Back to SuiteScript Functions

### Configuration APIs

NetSuite allows developers to programmatically obtain, and in some cases, change the values on certain account configuration pages. The internal IDs for SuiteScript-supported configuration pages are provided below. For the IDs that represent specific preferences on a configuration page, see 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, Accounting Preferences, Accounting Periods, Tax Periods.

Once a page is loaded, you can set configuration values using `nlobjConfiguration.setFieldValue(name, value)`.

The `nlapiLoadConfiguration` function is available in scheduled and Suitelet scripts only. 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).
• **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**

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.

```javascript
// 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 );
```

**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.

```javascript
// 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)
- nlapiTriggerWorkflow(recordtype, id, workflowid, actionid)
**nlapiInitiateWorkflow(recordtype, id, workflowid)**

Use this function to initiate a workflow on-demand. This function is the programmatic equivalent of the Initiate Workflow 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.

**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**

**nlapiTriggerWorkflow(recordtype, id, workflowid, actionid)**

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.

**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.

**Returns**

• The internal ID (int) of the workflow instance used to track the workflow against the record.

**Since**

• Version 2010.1

---

**Portlet APIs**

Use these APIs to work with NetSuite dashboard portlets. All APIs listed below are in alphabetical order.

• `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:

  ```javascript
  fld = portlet.addField('refrfield','inlinehtml','Refresh');
  fld.setDefaultValue('<a onclick="nlapiRefreshPortlet()" href="#">Refresh Now!</a>);
  ```

---

*Back to SuiteFlow APIs| Back to SuiteScript Functions*
**nlapiResizePortlet()**

Causes a FORM type nlobjPortlet to be resized.

This API can be used to ensure that a custom form portlet's embedded iframe resizes when the size of its contents change. This type of iframe does not resize automatically as its contents change, so when a form portlet's contents shrink they are surrounded by white space, and when contents grow they are cut off. A call to this API prevents these display issues.

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 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.

```javascript
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();
}
```
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.
**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.

---

**nlobjPivotColumn**

See nlobjPivotColumn - defined in the section on Standard Objects.

---

**nlobjPivotRow**

See nlobjPivotRow - defined in the section on Standard Objects.

---

**nlobjPivotTable**

See nlobjPivotTable - defined in the section on Standard Objects.

---

**nlobjPivotTableHandle**

See nlobjPivotTableHandle - defined in the section on Standard Objects.
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.

```javascript
//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](#)
Chapter 57 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 once it is returned in the script. The following is a list of all standard NetSuite objects.

- nlobjConfiguration
- nlobjContext
- nlobjCSVImport
- nlobjError
- nlobjFile
- nlobjLogin
- nlobjPivotColumn
- nlobjPivotRow
- nlobjPivotTable
- nlobjPivotTableHandle
- nlobjRecord
- nlobjReportColumn
- nlobjReportColumnHierarchy
- nlobjReportDefinition
nlobjConfiguration

Primary object used to encapsulate a NetSuite configuration/setup page. Note that nlapiLoadConfiguration(type) returns a reference to this object. Once 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.

Methods:

- getAllFields()
- getField(fldnam)
- getFieldText(name)
- getFieldTexts(name)
-getFieldValue(name)
- getFieldValues(name)
- getType()
- setText(name, text)
- setTexts(name, text)
- setValue(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

**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

**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.

```javascript
var configpage = nlapiLoadConfiguration('companypreferences');
var valtext = configpage.getFieldText('firstdayofweek'); // returns Sunday
```

**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

**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

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
**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

---

**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

---

**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

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

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

• getColorPreferences()
• getCompany()
• getDepartment()
• getDeploymentId()
• getEmail()
• getEnvironment()
getExecutionContext()
getFeature(name)
getLocation()
getLogLevel()
getName()
getPercentComplete()
getPermission(name)
getPreference(name)
getRemainingUsage()
getRole()
getRoleCenter()
getScriptId()
getSessionObject(name)
getSetting(type, name)
getSubsidiary()
getUser()
getVersion()
setPercentComplete(pct)
setSessionObject(name, value)
setSetting(type, name, value)

**getColorPreferences()**

Returns an Object containing name-value pairs of color groups to their corresponding RGB hex color based on the currently logged in user's color theme preferences.

Using this method, developers can get a user's color theme and apply the entire color theme map (or just individual attributes) to bundled Suitelets that have been installed into a user's account. Doing so ensures that the look-and-feel of each Suitelet matches all other records/pages running in the account.

**Note:** NetSuite color themes are read-only. Developers cannot programmatically create new color themes or add additional attributes to NetSuite's existing color theme template. In NetSuite, color themes can be set by going to Home > Set Preferences > Appearances tab.
Returns

- Object array (map), for example:
  
  ```
  values = {array} length=16
  buttonbackground = {string} D2D2C8
  text = {number} 000000
  portlet = {string} 70A837
  portletlabel = {string} FFFFFF
  crumbtext = {string} #FFD11A
  inactivetab = {string} FFCC00
  link = {number} 000000
  backgroundrequiredfld = {string} FFFFFE
  inactivetextontab = {number} 000000
  textontab = {string} FFFFFF
  bodybackground = {string} FFFFFF
  headbackground = {string} FFFFFF
  shadedbackground = {string} F8E7A5
  headerbar = {string} 70A837
  shadedborder = {string} D4BE63
  activetab = {string} 70A837
  ```

Since

- Version 2010.1

Example

```javascript
var ctx = nlapiGetContext();
var colorPrefs = ctx.getColorPreferences();
```

Color Preference Values

The following table provides the names of available NetSuite color theme attributes and a description of what each attribute affects.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buttonbackground</td>
<td>Background color of colored buttons</td>
</tr>
<tr>
<td>text</td>
<td>Text color for all text on the page body</td>
</tr>
<tr>
<td>portlet</td>
<td>Portlet trim (header background) color</td>
</tr>
<tr>
<td>portletlabel</td>
<td>Text in portlet heading</td>
</tr>
<tr>
<td>crumbtext</td>
<td>Text color of breadcrumbs on the header bar (color may be synthesized by page looks)</td>
</tr>
<tr>
<td>inactivetab</td>
<td>Inactive tab color</td>
</tr>
<tr>
<td>link</td>
<td>Text color for all links on the page body</td>
</tr>
</tbody>
</table>
SuiteScript Objects
Standard Objects

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();

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

Standard Objects | UI Objects | SuiteScript Functions

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backgroundrequiredfld</td>
<td>Background color for required fields in entry forms</td>
</tr>
<tr>
<td>inactivetextontab</td>
<td>Text color on the inactive tab</td>
</tr>
<tr>
<td>textontab</td>
<td>Text color on the active tab</td>
</tr>
<tr>
<td>bodybackground</td>
<td>Page background color</td>
</tr>
<tr>
<td>headbackground</td>
<td>Page header area background color</td>
</tr>
<tr>
<td>shadedbackground</td>
<td>Shaded area background color</td>
</tr>
<tr>
<td>headerbar</td>
<td>Header bar color</td>
</tr>
<tr>
<td>shadedborder</td>
<td>Border/header color around shaded areas</td>
</tr>
<tr>
<td>activetab</td>
<td>Active tab color</td>
</tr>
</tbody>
</table>

Standard Objects | UI Objects | SuiteScript Functions

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backgroundrequiredfld</td>
<td>Background color for required fields in entry forms</td>
</tr>
<tr>
<td>inactivetextontab</td>
<td>Text color on the inactive tab</td>
</tr>
<tr>
<td>textontab</td>
<td>Text color on the active tab</td>
</tr>
<tr>
<td>bodybackground</td>
<td>Page background color</td>
</tr>
<tr>
<td>headbackground</td>
<td>Page header area background color</td>
</tr>
<tr>
<td>shadedbackground</td>
<td>Shaded area background color</td>
</tr>
<tr>
<td>headerbar</td>
<td>Header bar color</td>
</tr>
<tr>
<td>shadedborder</td>
<td>Border/header color around shaded areas</td>
</tr>
<tr>
<td>activetab</td>
<td>Active tab color</td>
</tr>
</tbody>
</table>
Example

var context = nlapiGetContext();
var userDeptId = context.getDepartment();

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.

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();
**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

---

**getExecutionContext()**

Returns context information about what triggered the current script. Available values are:

- **userinterface** - Client SuiteScript or user event triggers invoked from the UI
- **webservice** - User event triggers invoked from webservice calls
- **csvimport** - User event triggers invoked during CSV imports
- **offlineclient** - User event triggers invoked during offlineclient
- **smbxml** - User event triggers invoked during SMBXML calls
- **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 user event script deployed to a Case record. Setting `getExecutionContext() to userinterface` ensures that this script is ONLY invoked from a user event occurring through the UI. When invoked, a tab is added to the Case record.

```javascript
function caseBeforeLoad(type, form)
{
    var currentContext = nlapiGetContext();
    if( (currentContext.getExecutionContext() == 'userinterface') && (type == 'edit' | type == 'view'))
    {
        var SampleTab = form.addTab('custpage_sample_tab', 'SampleTab123');
    }
}
```

**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.

```javascript
var ctx = nlapiGetContext();
context.getFeature('ADVBILLING');
```

**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

---

**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)`

---

**getName()**

Returns the currently logged in user’s name

*Note:* In a shopping context where the shopper is recognized but not logged in, this method can be used to return the shopper’s name, instead of getting it from the customer record.

Returns

• The logged in user’s name as a string

Since

• Version 2007.0
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 Setup >
Customization > Script Deployments > Status.

```javascript
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 given 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 given 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.

```javascript
var ctx = nlapiGetContext();
context.getPermission('ADMI_ACCOUNTING ');
```

**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.)

**Note:** 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.

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 checkbox 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.

```javascript
var ctx = nlapiGetContext();
context.getPreference('emailemployeeonapproval');
```

---

**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

See also

- SuiteScript Governance in the NetSuite Help Center
- nlapiGetContext()

Example

```javascript
var context = nlapiGetContext();
var usageRemaining = context.getRemainingUsage();
```
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 2007.0

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

**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.
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.

```javascript
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') );
    }
}
```

getSetting(type, name)

Use this API to get a system or script setting. Note that if you want to get session, feature, or permission settings directly, you can also use these nlobjContext methods:

- getSessionObject(name)
- getFeature(name)
- getPermission(name)
Parameters

- \(type\) [string] [required] - The type of script/system setting. Possible values include:
  - SESSION - session variable (volatile setting defined per session). Supported in server scripts only. **Important:** The SESSION type value is not supported in Client SuiteScript.
  - FEATURE - returns T (enabled) or F (disabled) depending on whether a feature is enabled. Supported in client and server SuiteScript.
    In the NetSuite Help Center, see 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 nlobContext.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:

```javascript
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 2007.0

getUser()

Returns the currently logged in user’s internal ID

Returns
  • The logged in user’s ID as a string

Since
  • Version 2007.0

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

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 Setup > Customization > Script Deployments > Status. See Using the Scheduled Script Status Page for more information about this page.

```javascript
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
    }
}
```

**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.

```javascript
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') );
    }
}
```

**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.

---

**nlobjCSVImport**

Primary object used to encapsulate a CSV import job. This object is passed as a parameter by nlapiSubmitCSVImport(nlobjCSVImport), which is used to asynchronously import record data into NetSuite.

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(file)
- setMapping(savedImport)
- setOption(jobName)
- setPrimaryFile(file)

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(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, 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**

- 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, preceded by an identifier for the record sublist for which data is being imported. For example:
    
    ```javascript
    setLinkedFile("item", nlapiLoadFile(74))
    ```
A list of sublist internal IDs is available in Scriptable Sublists.

- Raw string of the data to be imported.

**Returns**

- void

**Since**

- Version 2012.2

---

**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

---

**setOption(jobName)**

Sets the name of the import job to be shown on the status page for CSV imports.

**Parameters**

- `jobName` {string} [required] - 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
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:
    
    ```javascript
    setPrimaryFile(nlapiLoadFile(73))
    ```
  - Raw string of the data to be imported.

Returns

- void

Since

- Version 2012.2

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).

```javascript
function afterSubmit(type)
{
    if ( type == 'create' )
    {
        try
        {
            var subject = 'A ' + nlaiGetRecordType() + ' with id ' + nlapiGetRecordId() + ' was just created';
            nlapiSendEmail( '-5', 'alerts@company.com', subject );
        }
        catch ( e )
        {
            if ( e instanceof nlobjError )
                nlapiLogExecution( 'DEBUG', 'system error', e.getCode() + '
' + e.getDetails() )
            else
                nlapiLogExecution( 'DEBUG', 'unexpected error', e.toString() )
        }
        nlapiSetRedirectURL('RECORD', nlapiGetRecordType(), nlapiGetRecordId(), true);
    }
}
```

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().
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

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

getStackTrace()

Returns the stacktrace containing the location of the error
Returns
• String[]

Since
• Version 2008.2

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

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.

Important: In SuiteScript there is a 5MB limitation to the size of the file/document that can be created, submitted, or accessed using file APIs such as nlapiLoadFile, nlapiPrintRecord, nlapiMergeRecord, nlapiXMLToPDF, nlapiSubmitFile, and nlapiCreateFile.

nlobjFile Methods
• get_description()
• getFolder()
• getId()
• getName()
• getSize()
• getType()
• getURL()
• getValue()
• isInactive()
• isOnline()
• setDescription(description)
• setEncoding(encodingType)
• setFolder(id)
• setIsInactive(inactive)
• setIsOnline(online)
• setName(name)

**Note:** The following functions return a reference to nlobjFile:

• nlapiCreateFile(name, type, contents)
• nlapiLoadFile(id)
• nlapiMergeRecord(id, baseType, baseId, altType, altId, fields)
• nlapiPrintRecord(type, id, mode, properties)

---

### getDescription()

**Returns**

• The string description of the file. This is the description that appears in the Description field on the folder or file record.

**Since**

• Version 2009.1

---

### 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

**SuiteScript 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

---

**getName()**

Returns the name of the file

*Returns*

- The string value of the file name

---

**getSize()**

Returns the size of the file in bytes

*Returns*

- The integer value of the file size

Since

- Version 2009.1
**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

---

**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

---

**getValue()**

Returns the contents of the file (base 64 encoded for binary files)

**Returns**

- The string value of the file contents

**Since**

- Version 2009.1

---

**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)

### 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)

### 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
**setEncoding(encodingType)**

Sets the character encoding of a file. The available encoding types are as follows:

- UTF-8
- windows-1252
- ISO-8859-1
- GB18030
- GB2312
- SHIFT_JIS
- MacRoman

**Parameters**

- **encodingType** {string} [required] - The type of encoding for the file.

**Returns**

- void

**Since**

- Version 2010.1

**Example**

```javascript
var newFile = nlapiCreateFile('Chinese.csv', 'CSV', csvText);
nnewFile.setFolder(csvFolderId);
nnewFile.setEncoding('UTF-8');
nlapiSubmitFile(newFile);
```

**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
**setIsInactive(inactive)**

Sets the file's inactive status. When you inactive a file or folder, it no longer appears on lists unless (in the UI) you have selected the **Show Inactives** check box.

**Note:** The Show Inactives check box appears in the bottom-left corner of the Folders list. Navigate to the Folders list by going to Documents > Files > File Cabinet.

**Parameters**

- `inactive` [boolean] [required] - The file's inactive status. Set to `true` to inactive the file. Set to `false` to make the file active.

**Returns**

- `void`

**Since**

- Version 2009.1

---

**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
**setName(name)**

Sets the name of the file

**Parameters**

- `name` {string} [required] - The name of the file

**Returns**

- `void`

**Since**

- Version 2009.1

---

**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:** There is a 5MB limitation to the size of the document that can be submitted using the `nlapiSubmitFile` function.

**Example**

```javascript
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)
    }
}
```
// now create file and upload it to the file cabinet. You must use nlapiSubmitFile
// to upload a file to the file cabinet. See nlapiSubmitFile(file).
var id = nlapiSubmitFile(file)

// now attach file to customer record
nlapiAttachRecord("file", id, "customer", entity)

// now 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.

---

**nlobjLogin**

Primary object used to encapsulate NetSuite user login credentials.

**Methods**

- `changeEmail(currentPassword, newEmail, justThisAccount)`
- `changePassword(currentPassword, newPassword)`

**Note:** The following function returns a reference to nlobjLogin:

- `nlapiGetLogin()`

---

**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.

```javascript
//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.

```javascript
//Get the currently logged-in user credentials
var login = nlapiGetLogin();
//Change current password
login.changePassword('MycUrr3ntPa$$word', 'MyNeWPaSw0rD!');
```

Standard Objects | UI Objects | SuiteScript Functions

**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

**getParent()**
Get the parent column.

**Returns**
- nlobjPivotColumn - Null if it does not exist

**Since**
- Version 2012.2

**getLabel()**
Get the column label.

**Returns**
- string - Column label

**Since**
- Version 2012.2

**getSummaryLine()**
Get the summary line.
Returns
  • nlobjPivotColumn - Summary line if it exists, otherwise null

Since
  • Version 2012.2

getValue()
Get the value of the column.

Returns
  • object - The value of this column

Since
  • Version 2012.2

getVisibleChildren()
Get any defined children columns.

Returns
  • nlobjPivotColumn[] - Null if no children columns exist

Since
  • Version 2012.2

isHidden()
Checks if the column is hidden.

Returns
  • boolean - True if the column is hidden

Since
  • Version 2012.2
nlobjPivotRow

Object used to encapsulate a pivot table row.

Methods
- getAlias()
- getChildren()
- getLabel()
- getParent()
- getSummaryLine()
- getValue()
- getValue(pivotColumn)
- isDetailLine()

getAlias()

Get the row alias.

Returns
- string - The row alias.

Since
- Version 2012.2

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
**getLabel()**

Get the row label.

**Returns**
- string - The row label.

**Since**
- Version 2012.2

**getParent()**

Get the parent row if it exists.

**Returns**
- nlobjPivotRow - Null if the row does not exist.

**Since**
- Version 2012.2

**getSummaryLine()**

Get the summary line from the report.

**Returns**
- nlobjPivotRow - Null if the row is a detail line.

**Since**
- Version 2012.2

**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

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

isDetailLine()

Check if the row is a detail line.

Returns

- boolean - True if the row is a detail line.

Since

- Version 2012.2

nlobjPivotTable

Object used to encapsulate the pivot table.
Methods

- `getColumnHierarchy()`
- `getRowHierarchy()`

**getColumnHierarchy()**

Get the column hierarchy.

**Returns**

- `nlobjPivotColumn`

**Since**

- Version 2012.2

**getRowHierarchy()**

Get the row hierarchy.

**Returns**

- `nlobjPivotRow`

**Since**

- Version 2012.2

---

**nlobjPivotTableHandle**

Handle to the pivot table object. A handle is a reference which points to the pivot table.

**Methods**

- `getPivotTable()`
- `isReady()`

**getPivotTable()**

Get the pivot table object from the report definition.
**Note:** This is a blocking call and it will wait until the report definition execution has finished. Using isReady() is recommended to check execution state if blocking is unacceptable.

**Returns**
- nlobjPivotTable

**Since**
- Version 2012.2

---

**isReady()**

Returns the completion status flag of the report definition execution.

**Returns**
- boolean - True if the execution has finished.

**Since**
- Version 2012.2

---

**nlobjRecord**

Primary object used to encapsulate a NetSuite record.

**Methods**
- commitLineItem(group)
- createCurrentLineItemSubrecord(sublist, fldname)
- createSubrecord(fldname)
- editCurrentLineItemSubrecord(sublist, fldname)
- getCurrentLineItemValues(type, fldname)
- editSubrecord(fldname)
- findLineItemMatrixValue(group, fldnam, column, val)
- findLineItemValue(group, fldnam, value)
### SuiteScript Objects

#### Standard Objects

- `getAllFields()`
- `getAllLineItemFields(group)`
- `getCurrentLineItemMatrixValue(group, fldnam, column)`
- `getField(fldnam)`
- `getFieldText(name)`
- `getFieldTexts(name)`
- `getFieldValue(name)`
- `getFieldValues(name)`
- `getId()`
- `getLineItemCount(group)`
- `getLineItemField(group, fldnam, linenum)`
- `getLineItemMatrixField(group, fldnam, linenum, column)`
- `getLineItemMatrixValue(group, fldnam, lineum, column)`
- `getLineItemText(group, fldnam, linenum)`
- `getLineItemValue(group, name, linenum)`
- `getLineItemValues(type, fldname, linenum)`
- `getMatrixCount(group, fldnam)`
- `getMatrixField(group, fldname, column)`
- `getMatrixValue(group, fldnam, column)`
- `getRecordType()`
- `insertLineItem(group, linenum)`
- `removeLineItem(group, linenum)`
- `removeCurrentLineItemSubrecord(sublist, fldname)`
- `removeSubrecord(fldname)`
- `selectLineItem(group, linenum)`
- `selectNewLineItem(group)`
- `setCurrentLineItemMatrixValue(group, fldnam, column, value)`
- `setCurrentLineItemValue(group, name, value)`
- `setFieldText(name, text)`
- `setFieldTexts(name, text)`
• setFieldValue(name, value)
• setFieldValues(name, value)
• 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)

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.

**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);

---

**createCurrentLineItemSubrecord(sublist, fldname)**

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

This API is currently used only in the context of the Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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

See Creating a 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.

This API is currently used only in the context of the Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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

Example

See Creating a 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.

This API is currently used only in the context of the Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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

See Editing a subrecord in the NetSuite Help Center.

---

**getCurrentLineItemValues(type, fldname)**

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.
- **fldname** {string} [required] - The name of the multiselect field

Returns

- An array of string values for the multiselect sublist field

Since

- Version 2012.1

---

**editSubrecord(fldname)**

Returns a nlobjSubrecord object. Use this API to edit a subrecord from a body field on the parent record.
This API is currently used only in the context of the Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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

---

**findLineItemMatrixValue(group, fldnam, column, val)**

Use this method to return the line number of a particular price in a given 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

---
**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

---

**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
**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

---

**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
**getField(fldnam)**

Returns field metadata for a field

**Parameters**

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

**Returns**

- The nlobjField object

**Since**

- Version 2009.1

---

**getFieldText(name)**

Returns the UI display value for a select field. 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.

```javascript
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).
getFieldTexts(name)

Returns the UI display values for a multi-select field. 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.

```javascript
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)`.

getFieldValue(name)

Returns the value 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.

```javascript
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 a read-only array of multi-select field values

**Parameters**

- `name` {string} [required] - The name of the field whose value is being returned

**Returns**

- String[] of field IDs. Returns null if field is not on the record.

**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.

```javascript
var rec = nlapiLoadRecord('customer', 196); // load a customer record
var values = rec.getFieldValues('custentity1'); // get array of internal ID values set in custentity1 field
```

Standard Objects | UI Objects | SuiteScript Functions

**getId()**

Use this method to get the internal ID of a record or NULL for new records.

**Returns**

- Integer value of the record ID
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

getLineItemField(group, fldnam, linenum)

Returns field metadata for a line item (sublist) 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.

- `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

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**

```javascript
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);
```

---

**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

```javascript
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);
```

---

**getLineItemText(group, fldnam, linenum)**

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

```javascript
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(...)`.

---

**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.

**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.

**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, 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 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

```javascript
record = nlapiLoadRecord('inventoryitem', 333);
var itemid = record.getFieldValue('itemid');
var count = record.getMatrixCount('price', 'price');
```

**getMatrixField(group, fldname, column)**

Use this API to get field metadata for 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. 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.
- `fldname` {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.

```javascript
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);

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);
```
**getRecordType()**

Returns the record type (for example `assemblyunbuild` 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

**insertLineItem(group, linenum)**

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](https://app.netsuite.com/app/help/index.jsp?path=help/private/issite.jsp#/help/content/adg/issite.jsp?path=help/private/issite.jsp#/help/content/adg/tm0724240.xml) 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.

**Returns**

- void

**Example**

```javascript
// 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);
```
**removeLineItem(group, linenum)**

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).

**Returns**

- **void**

**Since**

- Version 2009.2

**Example**

```javascript
for (j=1; j <= soRecord.getLineItemCount('item'); j++)
{
    soRecord.removeLineItem('item', '1');
}
```

---

**removeCurrentLineItemSubrecord(sublist, fldname)**

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

This API is currently used only in the context of the Advanced Bin / Numbered Inventory feature. For information, see **Using SuiteScript with Advanced Bin / 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**

- **void**
Since

- Version 2011.2

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.

This API is currently used only in the context of the Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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

See Removing a subrecord 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**

```javascript
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);
```

**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](https://scripting.netsuite.com/) 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)` 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);

---

**setCurrentLineItemMatrixValue(group, fldnam, column, value)**

Use this API to set the value of a given 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

```javascript
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);
```

**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. **Note:** 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)` method.

```javascript
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);

---

**setFieldText(name, text)**

Sets the value of a select field using its corresponding display value

**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**

```javascript
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
```

---

**setFieldTexts(name, text)**

Sets the values for a multiselect field from their display values
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

```javascript
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);
```
**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

**setLineItemValue(group, name, linenum, value)**

Sets the value of a sublist line item.

*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.

**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.

```javascript
/*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.
SuiteScript Objects

## SuiteScript Developer and Reference Guide

### SuiteScript Objects

- **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. **Note:** 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 Multiple Currencies feature has been enabled and all pricing matrix values are being set on the USA pricing tab (price1).

```javascript
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');
```

### 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.

This API is currently used only in the context of the Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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

See Viewing a subrecord in the NetSuite Help Center.

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.

This API is currently used only in the context of the Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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

---

**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.

This API is currently used only in the context of the Advanced Bin / Numbered Inventory feature. For information, see Using SuiteScript with Advanced Bin / 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

Example

See Viewing a 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

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

**getParent()**

Get the parent reference of this column hierarchy.

**Returns**

- Either the parent reference to the `nlobjReportColumnHierarchy` object or null.

**Since**

- Version 2012.2

---

**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.

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[]}` [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.

```javascript
var reportDefinition = nlapiCreateReportDefinition();
var columns = new Array();
var filters = new Array();

columns[0] = new nlobjSearchColumn('internalID', null, 'group');
columns[1] = new nlobjSearchColumn('entity', null, 'group');
filters[0] = new nlobjSearchFilter('status', null, 'anyof', 'inProgress');

reportDefinition.addSearchDataSource('opportunity', null, filters, columns,
{'internalID':columns[0],
'entity':columns[1]});```

---

**executeReport**(form)

Creates the form for rendering from the report definition.

**Parameters**

• **form** `{nlobjReportForm}` [optional] - The form object created with nlapiCreateReportForm.

  If not specified the call waits until the execution is finished (synchronous) and an nlobjPivotTable will be available from the handle. If the parameter is set, the call returns immediately and the returned value references the nlobjReportForm - a pivot table handle will not be available in this case.

**Note**: Only one synchronous pivot table execution is allowed at a time. If a second synchronous execution is called, it will invalidate the first pivot table.

**Returns**

• **nlobjPivotTableHandle** - The identifier of the pivot table handle, or nlobjReportForm.

**Since**

• Version 2012.2
Example 1

This example shows how to create a pivot table for simple rendering as a report in a browser.

```
//Create a form to put the report on
var myForm = nlapiCreateReportForm('Pivot Report Sales Orders');

//Populate form here
...

//Build the form from the report definition
var myReportForm = reportDefinition.executeReport(myForm);

//Write the form back to the browser
response.writePage(myReportForm);
```

Example 2

This example shows how to create a pivot table for further processing with SuiteScript. The pivot table is not rendered.

```
//Create a form to put the report on
var myform = nlapiCreateReportForm('Pivot Report Sales Orders');

//Populate form here
...

//Build the form from the report definition, get the pivot table handle
var myPivotTableHandle = reportDefinition.executeReport();

//Get the pivot table object
var myPivotTable = myPivotTableHandle.getPivotTable();
```

---

**setTitle(title)**

Sets the title of the report definition.

**Parameters**

- `title {string} [optional]` - The name of the report definition.

**Since**

- Version 2012.2
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.

```javascript
function runOpportunitiesPivot(request, response) {
    // Instantiate a report definition to work with
    var reportDefinition = nlapiCreateReportDefinition();

    // Define the rows/column hierarchy and the actual column data
    var customer = reportDefinition.addRowHierarchy('entity', 'Customer', 'TEXT');
    var salesrep = reportDefinition.addColumn('salesrep', false, 'Sales Rep', null, 'TEXT', null);
    var entstat = reportDefinition.addColumnHierarchy('entitystatus', 'Opportunity Status', null, 'TEXT');
    var total = reportDefinition.addColumn('projectedtotal', true, 'Projected Total', entstat, 'CURRENCY', null);
    var prob = reportDefinition.addColumn('probability', false, 'Probability %', entstat, 'PERCENTAGE', 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 report 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.
Moreover, the UI equivalent of defining the row/column hierarchy and the actual column data of a pivot report is as follows:

<table>
<thead>
<tr>
<th>Customer</th>
<th>Sales Rep</th>
<th>Closed Won</th>
<th>Percentage</th>
<th>Proposed Total</th>
<th>Probability</th>
<th>Qualified Total</th>
<th>Probability</th>
<th>Total</th>
<th>Projected Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>106 Jordan Flower</td>
<td>Rachel Liu</td>
<td>$11,940.90</td>
<td>103.0%</td>
<td>$11,940.90</td>
<td>103.0%</td>
<td>$23,881.80</td>
<td>103.0%</td>
<td>$23,881.80</td>
<td></td>
</tr>
<tr>
<td>105 Larry Smith</td>
<td>John</td>
<td>$7,953.90</td>
<td>110.0%</td>
<td>$7,953.90</td>
<td>110.0%</td>
<td>$15,907.80</td>
<td>110.0%</td>
<td>$15,907.80</td>
<td></td>
</tr>
<tr>
<td>1162 Perry Brooks</td>
<td>John</td>
<td>$2,393.90</td>
<td>110.0%</td>
<td>$2,393.90</td>
<td>110.0%</td>
<td>$4,787.80</td>
<td>110.0%</td>
<td>$4,787.80</td>
<td></td>
</tr>
<tr>
<td>1163 John Watson</td>
<td>John</td>
<td>$2,393.90</td>
<td>110.0%</td>
<td>$2,393.90</td>
<td>110.0%</td>
<td>$4,787.80</td>
<td>110.0%</td>
<td>$4,787.80</td>
<td></td>
</tr>
<tr>
<td>1255 Ida Tuck</td>
<td>Rachel Liu</td>
<td>$2,965.03</td>
<td>110.0%</td>
<td>$2,965.03</td>
<td>110.0%</td>
<td>$5,930.06</td>
<td>110.0%</td>
<td>$5,930.06</td>
<td></td>
</tr>
<tr>
<td>1266 Nicole Dyer</td>
<td>Rachel Liu</td>
<td>$2,393.90</td>
<td>110.0%</td>
<td>$2,393.90</td>
<td>110.0%</td>
<td>$4,787.80</td>
<td>110.0%</td>
<td>$4,787.80</td>
<td></td>
</tr>
<tr>
<td>127 Tom Peters</td>
<td>John</td>
<td>$96,343.94</td>
<td>110.0%</td>
<td>$96,343.94</td>
<td>110.0%</td>
<td>$192,687.88</td>
<td>110.0%</td>
<td>$192,687.88</td>
<td></td>
</tr>
<tr>
<td>1280 John Jolly</td>
<td>John</td>
<td>$90,000.00</td>
<td>100.0%</td>
<td>$90,000.00</td>
<td>100.0%</td>
<td>$180,000.00</td>
<td>100.0%</td>
<td>$180,000.00</td>
<td></td>
</tr>
<tr>
<td>1282 Celeste McKeen</td>
<td>Rachel Liu</td>
<td>$4,725.00</td>
<td>110.0%</td>
<td>$4,725.00</td>
<td>110.0%</td>
<td>$9,450.00</td>
<td>110.0%</td>
<td>$9,450.00</td>
<td></td>
</tr>
<tr>
<td>1283 Coop McPherson</td>
<td>Rachel Liu</td>
<td>$2,660.03</td>
<td>110.0%</td>
<td>$2,660.03</td>
<td>110.0%</td>
<td>$5,320.06</td>
<td>110.0%</td>
<td>$5,320.06</td>
<td></td>
</tr>
<tr>
<td>1285 Russell Evers</td>
<td>Rachel Liu</td>
<td>$2,393.90</td>
<td>110.0%</td>
<td>$2,393.90</td>
<td>110.0%</td>
<td>$4,787.80</td>
<td>110.0%</td>
<td>$4,787.80</td>
<td></td>
</tr>
<tr>
<td>1286 Cameron Sandell</td>
<td>John</td>
<td>$387,600.00</td>
<td>100.0%</td>
<td>$387,600.00</td>
<td>100.0%</td>
<td>$775,200.00</td>
<td>100.0%</td>
<td>$775,200.00</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>$1,081,482.73</td>
<td></td>
<td>$1,081,482.73</td>
<td></td>
<td>$2,162,965.46</td>
<td></td>
<td>$2,162,965.46</td>
<td></td>
</tr>
</tbody>
</table>
For more information about creating a pivot report through the UI, see Creating a Pivot Report.

In SuiteScript, this looks like:

```javascript
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, 'PERCENTAGE', null);
```

---

**nlobjReportForm**

Object used to encapsulate the report form and render the report in HTML.

Since

- Version 2012.2

---

**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

**getParent()**

Get the parent reference of this row hierarchy.

**Returns**

- Either the parent reference to the `nlobjReportRowHierarchy` object or null.

**Since**

- Version 2012.2

---

**nlobjRequest**

Primary object used to encapsulate an HTTP GET or POST request. When creating Suitelets you will 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**

```javascript
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)
- getItemCount(group)
- getItemCountValue(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

---

**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

```javascript
var params = request.getAllParameters();
for (param in params) {
    nlapiLogExecution('DEBUG', 'parameter: ' + param);
    nlapiLogExecution('DEBUG', 'value: ' + params[param]);
}
```

**getBody()**

Returns the body of the POST request

Returns

- The string value of the request body

Since

- Version 2008.1

**getFile(id)**

Returns a file added through the nlobjForm.addField(name, type, label, sourceOrRadio, tab) method. When adding a file field type, you will set the `type` parameter of 'file'.

Returns

- nlobjFile

Since

- Version 2010.1

Example

See Uploading Files to the File Cabinet Using SuiteScript.
**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

**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](https://help.netsuite.com) 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

**getLineItemValue(group, name, line)**

Returns the value of a sublist line item.

**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.
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

---

**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

```javascript
function demoSimpleForm(request, response) {
  if ( request.getMethod() == 'GET' ) {
    var form = nlapiCreateForm('Simple Form');

    //remainder of code...

    response.writePage(form);
  }
}
```
**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

---

**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

---

**getURL()**

Returns the full URL of the request

**Returns**
- The string value of the request URL

**Since**
- Version 2008.1
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.

Example

```javascript
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)`
- `setContentType(type, name, disposition)`
- `setHeader(name, value)`
- `sendRedirect(type, identifier, id, editmode, parameters)`
- `write(output)`
- `writeLine(output)`
- `writePage(pageobject)`
**addHeader(name, value)**

Adds a header to the response. If this header has already been set, this will add a new header to the response. Note that all user-defined headers must be prefixed with `Custom-Header` otherwise an SSS_INVALID_ARG error will be thrown.

**Parameters**

- `name` {string} [required] - The name of the header
- `value` {string} [required] - The value used to set header

**Returns**

- void

**Since**

- Version 2008.2

**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

**getBody()**

Returns the body 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 body
**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

**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

**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

**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.

**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
```javascript
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');
}
```

**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} [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.

   **Important:** The values for this parameter can be *true* or *false*, not T or F.

• **parameters** {hashtable} [optional] - An associative array of additional URL parameters as name/value pairs

**Returns**

• void

**Since**

• Version 2008.2

---

**write(output)**

Write information (text/xml/html) to the response

**Parameters**

• **output** {string | document} [required] - String or Document being written

**Returns**

• void

**Example**

```javascript
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

---

**writeLine(output)**

Write line information (text/xml/html) to the response

**Parameters**

• **output** {string} [required] - String being written
Returns
- void

Since
- Version 2008.2

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);
    }
}

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).
Once 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 once the object is loaded.

For general information on executing NetSuite searches using SuiteScript, see Searching Overview in the NetSuite Help Center.

**Methods**

- `addColumn(column)`
- `addColumnns(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()`
**addColumn(column)**

Adds a single return column to the search. Note that existing columns on the search are not changed.

**Parameters**

- `column` {nlobjSearchColumn} [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.

```javascript
// Define search filters
var filters = new Array();
filters[0] = new nlobjSearchFilter( 'trandate', null, 'onOrAfter', 'daysAgo90' );
filters[1] = new nlobjSearchFilter( 'projectedamount', null, 'between', 1000, 100000 );
filters[2] = new nlobjSearchFilter( 'salesrep', 'customer', 'anyOf', -5, null );
// Define return columns
var columns = new Array();
columns[0] = new nlobjSearchColumn( 'salesrep' );
columns[1] = new nlobjSearchColumn( 'expectedclosedate' );
columns[2] = new nlobjSearchColumn( 'entity' );
// Create the saved search
var search = nlapiCreateSearch( 'opportunity', filters, columns );
var searchId = search.saveSearch('My Opportunities in Last 90 Days', 'customsearch_kr');
// Load the existing search and add a new column to the search
var newSearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
// Call addColumn to add additional column to the existing search
var newColumn = new nlobjSearchColumn('somecolumn');
newSearch.addColumn(newColumn);
var newId = newSearch.saveSearch('My New Search', 'customsearch_kr2');
```

**addColumn(columns)**

Adds multiple return columns to the search. Note that existing columns on the search are not changed.
Parameters

- **columns** {nlobjSearchColumn[]} [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 additional columns. After the new columns are added, a new script ID is assigned to the search.

```javascript
// 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 additional columns to the existing search
newSearch.addColumns(newColumns);
var newId = newSearch.saveSearch('My Opportunities in Last 90 Days', 'customsearch_kr');
```

**addFilter(filter)**

Adds a single search filter. Note that existing filters on the search are not changed.

**Note:** This method does not accept a search filter expression (Object[]) as parameter. Only a single search filter (nlobjSearchFilter) is accepted.
Parameters

- `filter {nlobjSearchFilter} [required]` - The nlobjSearchFilter you want added to the search.

Returns

- `void`

Since

- Version 2012.1

Example

This example shows how to create a saved search and then load the search to add an additional filter. After the new filter is added, a new script ID is assigned to the search.

```javascript
// 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');
```

---

**addFilters(filters)**

Adds a search filter list. Note that existing filters on the search are not changed.

**Note:** This method does not accept a search filter expression (Object[]) as parameter. Only a search filter list (nlobjSearchFilter[]) is accepted.

Parameters

- `filters {nlobjSearchFilter[]} [required]` - The list (array) of zero or more nlobjSearchFilter you want added to the search.
Returns

- void

Since

- Version 2012.1

Example

This example shows how to create a saved search and then load the search to add additional 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 additional filters to the existing search
newSearch.addFilters( newFilters );
var newId = newSearch.saveSearch( 'My New Search', 'customsearch_kr2' );
```
Example

This example shows how to load an existing saved search and delete it.

```javascript
// Load the existing search and then delete it
var mySearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
mySearch.deleteSearch();
```

getColumns()

Gets the search return columns for the search.

Returns

- `nlobjSearchColumn[]`

Since

- Version 2012.1

Example

This example shows how to load an existing saved search and get its search return columns and its filters.

```javascript
var s = nlapiLoadSearch('opportunity', 'customsearch_kr');
var columns = s.getColumns();
var filters = s.getFilters();
```

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.

```javascript
var s = nlapiLoadSearch('opportunity', 'customsearch_kr');
```
var filterExpression = s.getFilterExpression();

**getFilters()**

Gets the filters for the search.

**Note:** This method does not return a search filter expression (Object[]). Only a search filter list (nlobjSearchFilter[]) is returned. If you want to get a search filter expression, see getFilterExpression().

**Returns**

- nlobjSearchFilter[]

**Since**

- Version 2012.1

**Example**

This example shows how to load an existing saved search and get its search return columns and its filters.

```javascript
var s = nlapiLoadSearch('opportunity', 'customsearch_kr');
var columns = s.getColumns();
var filters = s.getFilters();
```

**getIId()**

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.

```javascript
// Load the existing search and then get its internal ID assigned by NetSuite
var mySearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
var internalId = mySearch.getId();
```

**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.

```javascript
// 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…
}
```

**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_myssearch` 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.

```javascript
// Load the existing search and then get its developer-assigned script ID
var mySearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
var scriptId = mySearch.getScriptId();
```

### 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

```javascript
var searchId = ...;
var s = nlapiLoadSearch(null, searchId);  // load a search with an unknown type
var t = s.getSearchType();
```

### 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)` allows you to create and run ad-hoc searches that are never saved to the database, much like `nlapiSearchRecord(...)`. 
Note that this method returns the \texttt{nlobjSearchResultSet} object, which provides you with more flexibility when working with or iterating through your search results. Therefore, you may also want to use \texttt{runSearch()} in conjunction with \texttt{nlapiLoadSearch(...)}. By doing so you can load an existing saved search, call \texttt{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**

- \texttt{nlobjSearchResultSet}

**Since**

- Version 2012.1

**Example 1**

This example shows how to load an existing saved search and re-run the search using \texttt{runSearch()}. After \texttt{runSearch()} executes, the search's entire result set is returned in an \texttt{nlobjSearchResultSet} object. You can then use the \texttt{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 \texttt{true} or \texttt{false}. True causes iteration to continue. False causes iteration to stop.

**Note:** The work done in the context of the callback function counts towards the governance of the script that called it. For example, if the callback function is running in the context of a scheduled script, which has a 10,000 unit governance limit, you must be sure the amount of processing within the callback function does not put the entire script at risk of exceeding scheduled script governance limits.

```javascript
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.

```javascript
// 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;
}

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 string.

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.

```javascript
// 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');
```

**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[]}` [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.

```javascript
// Define search filters
var filters = new Array();
filters[0] = new nlobjSearchFilter( 'trandate', null, 'onOrAfter', 'daysAgo90' );
```
filters[1] = new nllobjSearchFilter( 'projectedamount', null, 'between', 1000, 100000 );
filters[2] = new nllobjSearchFilter( 'salesrep', 'customer', 'anyOf', -5, null );
// Define search columns
var columns = new Array();
columns[0] = new nllobjSearchColumn( 'salesrep' );
columns[1] = new nllobjSearchColumn( 'expectedclosedate' );
columns[2] = new nllobjSearchColumn( '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 nllobjSearchColumn( 'email' );
newcolumns[1] = new nllobjSearchColumn( '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.

```javascript
// 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');
```

**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.

```javascript
var s = nlapiCreateSearch('Opportunity', filters, columns);
s.setIsPublic(true);
var searchId = s.saveSearch('My public opp search', 'customsearch_opp_public');
```

**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`, `addColumn`, 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 and in 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.

```javascript
// Load the search and redirect user to search definition page in the UI
var oppSearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
oppSearch.setRedirectURLToSearch();
```
**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 and in 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.

```javascript
// Load the search
var oppSearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
oppSearch.setRedirectURLToSearchResults();
```

**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 just 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]]]]

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.

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 just the same as 'AND', 'OR', and 'NOT'.
- 'NOT' must be followed by a filter term or a search filter expression.
- 'AND' or 'OR' must be preceded and followed by a filter term or search filter expression.

**Filter term**

A filter term is a JavaScript array that is composed of three or more elements, as follows:

- Filter identifier - a JavaScript string of the form: 
  - filter_name (such as amount) - This is equivalent to new nlobjSearchFilter('amount', null, ...) where 'amount' is the internal ID of the search field.
  - join_id.filter_name (such as customer.salesrep) - This is equivalent to new nlobjSearchFilter('salesrep', 'customer', ...) where 'customer' is the search join id used for the search field specified as filter name 'salesrep'. The filter name in this case may not be a formula filter like “formulatext: ...”.

For a list of search join ids and filter names associated to a record, see the SuiteScript Records Browser.

- formula_type: formula_text (such as formulatext: SUBSTR({custentity_myfield}, 3))
• **aggregate_function(filter_identifier)** (such as `max(amount)` ) - The filter_identifier itself can contain a joined record, or can be a formula filter. However, it cannot be both a joined record and a formula filter.

• Operator - a JavaScript string
• Operand - a JavaScript string or integer
• (Optional) Additional operands

**nlobjSearchColumn**

Primary object used to encapsulate search return columns. For information on executing NetSuite searches using SuiteScript, see Searching Overview in the NetSuite Help Center.

Note that 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 following nlobjSearchColumn methods against your search column results.

**Methods**

• `constructor(name, join, summary)`
• `getFormula()`
• `getFunction()`
• `getJoin()`
• `getLabel()`
• `getName()`
• `getSort()`
• `getSummary()`
• `setFormula(formula)`
• `setFunction(functionid)`
• `setLabel(label)`
• `setSort(order)`
• `setWhenOrderedBy(name, join)`

**Standard Objects | UI Objects | SuiteScript Functions**

**constructor(name, join, summary)**

Constructor used to create a search return column
Parameters

- **name** {string} - The search return column name
- **join** {string} - The join id for this search return column
- **summary** {string} - The summary type for this column
  - group
  - sum
  - count
  - avg
  - min
  - max

Returns

- nlobjSearchColumn

Since

- Version 2007.0

---

**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, will return null.

```javascript
// 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 2 shows the search results after the search is run.

Figure 3 shows the values for label and formula for the Customer Names (Reverse) column.
Figure 4 shows the values for **label** and **functionName** for the **Customer Balance** column.

**getFunction()**

**Returns**

- The function used in this search column as a string

**Since**

- Version 2009.1

**Example**

- See the sample in getFormula()

**getJoin()**

**Returns** join id for this search column
Returns

- The join id as a string

Since

- Version 2008.1

---

**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()`

---

**getName()**

**Returns**

- The name of the search column as a string

**Since**

- Version 2008.1

---

**getSort()**

Returns the sort direction for this column

**Returns**

- string
Since
• Version 2011.1

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

setFormula(formula)

Set the formula used for this column. Name of the column can either be formulatext, formulanumeric, formuladatetme, formulapercen, 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.

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:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Date Function</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>percentOfTotal</td>
<td>% of Total</td>
<td>No</td>
<td>percent</td>
</tr>
<tr>
<td>absoluteValue</td>
<td>Absolute Value</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>ageInDays</td>
<td>Age In Days</td>
<td>Yes</td>
<td>integer</td>
</tr>
<tr>
<td>ageInHours</td>
<td>Age In Hours</td>
<td>Yes</td>
<td>integer</td>
</tr>
<tr>
<td>ageInMonths</td>
<td>Age In Months</td>
<td>Yes</td>
<td>integer</td>
</tr>
<tr>
<td>ageInWeeks</td>
<td>Age In Weeks</td>
<td>Yes</td>
<td>integer</td>
</tr>
<tr>
<td>ageInYears</td>
<td>Age In Years</td>
<td>Yes</td>
<td>integer</td>
</tr>
<tr>
<td>calendarWeek</td>
<td>Calendar Week</td>
<td>Yes</td>
<td>date</td>
</tr>
<tr>
<td>day</td>
<td>Day</td>
<td>Yes</td>
<td>date</td>
</tr>
<tr>
<td>month</td>
<td>Month</td>
<td>Yes</td>
<td>text</td>
</tr>
<tr>
<td>negate</td>
<td>Negate</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>numberAsTime</td>
<td>Number as Time</td>
<td>No</td>
<td>text</td>
</tr>
<tr>
<td>quarter</td>
<td>Quarter</td>
<td>Yes</td>
<td>text</td>
</tr>
<tr>
<td>rank</td>
<td>Rank</td>
<td>No</td>
<td>integer</td>
</tr>
<tr>
<td>round</td>
<td>Round</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>roundToHundredths</td>
<td>Round to Hundredths</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>roundToTenths</td>
<td>Round to Tenths</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>weekOfYear</td>
<td>Week of Year</td>
<td>Yes</td>
<td>text</td>
</tr>
<tr>
<td>year</td>
<td>Year</td>
<td>Yes</td>
<td>text</td>
</tr>
</tbody>
</table>

Returns

- **nlobjSearchColumn**

Since

- Version 2011.1

**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

Example 1

Execute a customer search with the customer internal ID in the results. Set the internal ID column to sort in ascending order.

```javascript
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.

```javascript
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);
nlobjSearchFilter

Primary object used to encapsulate search filters. For information on executing NetSuite searches using SuiteScript, see Searching Overview in the NetSuite Help Center.

**Note:** By default, search filter list (nlobjSearchFilter[]) makes use only of an implicit ‘AND’ operator for filters. This is contrary to search filter expression that can explicitly use either ‘AND’ or ‘OR’ operators.

When searching on checkbox 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,

```javascript
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. See the list of possible operator values:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>after</td>
<td>isempty</td>
</tr>
<tr>
<td>anyof</td>
<td>isnot</td>
</tr>
<tr>
<td>before</td>
<td>isnotempty</td>
</tr>
<tr>
<td>between</td>
<td>lessthan</td>
</tr>
<tr>
<td>contains</td>
<td>lessthanorequalto</td>
</tr>
<tr>
<td>doesnotcontain</td>
<td>noneq</td>
</tr>
<tr>
<td>doesnotstartwith</td>
<td>notafter</td>
</tr>
<tr>
<td>equalto</td>
<td>notbefore</td>
</tr>
<tr>
<td>greaterthan</td>
<td>notbetween</td>
</tr>
<tr>
<td>greaterthanorequalto</td>
<td>notequalto</td>
</tr>
<tr>
<td>haskeywords</td>
<td>notgreaterthan</td>
</tr>
<tr>
<td>is</td>
<td>notgreaterthanorequalto</td>
</tr>
</tbody>
</table>

• **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. See the following list of possible values:
lastfiscalquarter
lastfiscalquartertyodate
lastfiscalyear
lastfiscalyeartodate
lastmonth
lastmonthdoctype
lastrollingquarter
lastrollingyear
lastweek
lastweekdoctype
nextbusinessweek
nextfiscalquarter
nextfiscalyear
nextfourweeks
nextmonth
nextonemonth
nextonequarter
nextoneweek
nextoneyear
nextweek
previousoneday
previousonemonth
previousonequarter
previousoneweek
previousoneyear
previousrollingquarter
previousrollingyear
samemonthlastfiscalquarter
samemonthlastfiscalquartertyodate
samemonthlastfiscalyear
samemonthlastfiscalyeartodate
samequarterlastfiscalyear
samequarterlastfiscalyeartodate
thisbusinessweek
thisfiscalquarter
thisfiscalquartertyodate
thisfiscalyear
thisfiscalyeartodate
thismonth
thismonthdoctype
thisrollingquarter
thisrollingyear
thisweek
thisweekdoctype
thisyear
today
tomorrow
yesterday
daysagoxx
weeksagoxx
monthsagoxx
quartersagoxx
yearsagoxx
daysfromnowxx
weeksfromnowxx
monthsfromnowxx
quartersfromnowxx
yearsfromnowxx
daysagoxx
weeksagoxx
monthsagoxx
quartersagoxx
yearsagoxx
daysfromnowxx
weeksfromnowxx
monthsfromnowxx
quartersfromnowxx
yearsfromnowxx

**Returns**

- nlobjSearchFilter

**Since**

- Version 2007.0

---

**getFormula()**

Returns the formula used for this filter

**Returns**

- The formula used for this filter

**Since**

- Version 2011.1

---
**getJoin()**

Returns the join id for this search filter

**Returns**
- The string value of the search join

**Since**
- Version 2008.1

---

**getName()**

Returns the name for this search filter

**Returns**
- The string value of the search filter

**Since**
- Version 2007.0

---

**getSummaryType()**

Returns the summary type used for this filter

**Returns**
- The summary type used for this filter

**Since**
- Version 2011.1

---

**getOperator()**

Returns the filter operator that was used
Returns

- The string value of the search operator

Since

- Version 2008.2

**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**

```javascript
var filters = new Array();
filters[0] = new nlobjSearchFilter('formulatext', null, 'startswith', 'a');
filters[0].setFormula('SUBSTR({custbody_stringfield}, 3)');
```

**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:

<table>
<thead>
<tr>
<th>Summary type ID (used in script)</th>
<th>Summary Label (as seen in UI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>Maximum</td>
</tr>
<tr>
<td>min</td>
<td>Minimum</td>
</tr>
</tbody>
</table>
nlobjSearchFilter

Returns

• nlobjSearchFilter

Since

• Version 2011.1

Example

See the sample in Using Summary Filters in Search.

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)
**getAllColumns()**

Returns an array of nlobjSearchColumn objects containing all the columns returned in a specified search

Returns

- nlobjSearchColumn[]

Since

- Version 2009.2

**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*

**getText(column)**

Returns the text value for this nlobjSearchColumn if it is a select field

Parameters

- *column* [nlobjSearchColumn] [required] - The name of the search result column.

Returns

- string
Since

- Version 2009.2

**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

**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
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 column

**Parameters**

- `column {nlobjColumn} [required] - Column object used in search whose value you want to return`

**Returns**

- String value of the 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 column.

```javascript
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)

// execute the campaign search
```
var searchresults = nlapiSearchRecord( 'campaign', null, filters, columns );

// get the value of the Email column
var val = searchresults[0].getValue(columns[2]);

### `nlobjSearchResultSet`

Primary object used to encapsulate a set of search results. The `nlobjSearchResultSet` object provides both an iterator interface, which allows you to process each result of the search, and stop at any time, and a slice interface, which allows you to retrieve 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:

```javascript
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:

```javascript
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().

---

`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[]`

Since

- Version 2012.1

---

`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.

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

```javascript
// Load a search and get the first three results.
var search = nlapiLoadSearch('opportunity', 'customsearch_al');
var resultSet = search.runSearch();
var firstThreeResults = resultSet.getResults(0, 3);
```

---

**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**

```javascript
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() );
```

**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**

```javascript
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() );
```
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.

Important: Subrecords are currently supported only in the context of the new Advanced Bin / Numbered Inventory Management feature. For sample scripts, see Using SuiteScript with Advanced Bin / Numbered Inventory Management.

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 a 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.

```javascript
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

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 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)
- getLastError()
- 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.

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 will appear as the Company Information group. Next, two text fields (Company Name and Legal Name) are added to the Company Information field group. Also notice that 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. Once returned, help text is added to each of these fields.

```javascript
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");

assistant.getField("legalname").setHelpText("Enter a Legal Name if it differs from your company name")
assistant.getField("shipatoattention").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")

### 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. Once 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.");

![Small Business Setup Assistant](image)

**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");
}
**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

**getAllSteps()**

Use this method to return an array of all the assistant steps for this assistant.

**Returns**
- nlobjAssistantStep[]

**Since**
- Version 2009.2

**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
**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.

```javascript
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. Once 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.
```javascript
var fields = assistant.getAllFields();
for (var i = 0; i < fields.length; i++) {
    assistant.getField(fields[i]).setDefaultValue(nlapiGetContext().getSessionObject(fields[i]))
}
```

### 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

### 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.

```javascript
//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
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

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

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
**sendRedirect(response)**

Use this method to manage redirects in an assistant. In most cases, an assistant redirects to itself as in:

```javascript
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](#).

---

**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.

```javascript
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...
**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.

```javascript
assistant.addField("companyname", "text", "Company Name", null, "companyinfo");
assistant.addField("address1", "text", "Address 1", null, "companyinfo");
assistant.setFieldValues({companyname: "Wolfe Electronics", address1: "123 Main St., Anytown, USA");
```

**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 once 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.

**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

**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`:

  ![Ordered Steps](image1)

  ordered parameter set to `false`:

  ![Unordered Steps](image2)

**Returns**

- `void`

**Since**

- Version 2009.2

---

**setScript(script)**

Use this method to set the scriptId for a global client script you want to run on an assistant page.

**Parameters**

- `script {string | int} [required]` - The scriptId of the global client script
Returns

- void

Since

- Version 2009.2

**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.

```javascript
var assistant = nlapiCreateAssistant("Small Business Setup Assistant", true, false);
assistant.setOrdered(true);
assistant.setShortcut(false);
```

**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.

```javascript
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.");
```

---

**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(...).

```javascript
function showAssistant(request, response)
{
    /* first create assistant object and define its steps. */
    var assistant = nlapiCreateAssistant("Small Business Setup Assistant");
    assistant.setTitle("Sample Title");

    // 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

### 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
**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

---

**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

---

**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
**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

---

**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

---

**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

**setHelpText(help)**

Use this method to set help text for an assistant step.

**Parameters**
- *help {string} [required] - The help text for the step*

**Returns**
- nlobjAssistantStep

Since
- Version 2009.2

**Example**

See the sample provided in nlobjAssistant.addStep(name, label).

**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
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.

If you add a custom button to a record, it will appear before the printer icon (see figure).

![Custom Button Example](image)

**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 Configuring Buttons and Actions in the NetSuite Help Center for details.

**Methods**

- `setDisabled(disabled)`
- `setLabel(label)`
- `setVisible(visible)`

**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 simply 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

```javascript
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 `$lobjForm.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**

- `$lobjButton`

**Since**

- Version 2008.2

**Example**

```javascript
function relabelUpdateOrderButton(type, form)
{
    // Get the button
    var button = form.getButton('custpage_updateorderbutton');

    // Relabel the button's UI label
    button.setLabel('Modify Order');
}
```
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

```javascript
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);
}
```

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.

Once 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)

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

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
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
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. Once 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, fldnammm, linenenum)` (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)`

**addSelectOption(value, text, selected)**

Adds a select option to a SELECT field

**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.

```javascript
// 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

```javascript
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 contain '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.

```javascript
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.

```javascript
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

**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

**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
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)

Returns

- nlobjField

Since

- Version 2008.2

Example

This sample shows a simple user event script, which specifies the hidden parameter to hide a checkbox field on a beforeLoad event. When the record is loaded (for example, an Estimate or Customer record), the checkbox referenced in this script will be hidden from the user.

```javascript
function beforeLoad(type, form)
{
    form.getField('custbody_myspecialcheckbox').setDisplayType('hidden');
}
```

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.

```javascript
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

**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

---

**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
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:

```javascript
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:

```javascript
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:

```javascript
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.

---

**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
**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).

**setSingleColumn(column)**

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**

- **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:

```javascript
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);
}
```

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'.
• scriptId {string} [optional] - The scriptId of the script that is allowed to use this credential field. For example, 'customscript_my_script'.
• 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 nlapiRequestUrlWithCredentials with this credential is restricted to the same entity that originally entered the credential. An example where you would not want this (you would set to false) is with a credit card processor, where the credential represents the company an employee is working for and multiple entities will be expected to make secure calls out to the processor (clerks, for example). An example where you might want to set entityMatch to true is when each user of the remote call has his or her own credentials.
• tab {string} [optional] - The tab parameter can be used to specify either a tab or a field group (if you have added nlobjFieldGroup objects to your form). If tab is empty, then the field is added to the “main” section of the form.

Returns

• nlobjField object

Since

• Version 2012.1

Example 1

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. Once the user submits the form, the credentials will be sent to a website called www.mysite.com. Additionally, only a script with the script ID ‘customscript_a’ can use this credential field. Finally, the entityMatch parameter is set
to false. This means that when `nlapiRequestURLWithCredentials(credentials, url, postdata, headers, httpMethod)` is used with this credential there is no restriction to the same entity that originally entered the credential.

```javascript
function demoSimpleForm(request, response)
{
  if (request.getMethod() == "GET")
  {
    var form = nlapiCreateForm('Simple Form');
    form.addCredentialField('username', 'Username', 'www.mysite.com', null, , 'customscript_a', false, null);
    response.addSubmitButton('Save');
    response.writePage( form );
  }
}
```

This code shows how to retrieve the credential in a Suitelet.

```javascript
function demoSimpleForm(request, response)
{
  if (request.getMethod() == "POST")
  {
    var handle = request.getParameter("username");
  }
}
```

This code shows how to use the credential.

```javascript
var creds= [record.getFieldValue("username")];
var sUrl = "https://www.mysite.com/serviceA?user=" +"{" + record.getFieldValue("username") +"}";
response = nlapiRequestURLWithCredentials(creds, sUrl, null, null, null);
```

**addField(name, type, label, sourceOrRadio, tab)**

Adds an nlobjField object to a form and returns a reference to it

**Parameters**

- `name` [string] [required] - The internal ID name of the field. The internal ID must be in lowercase, contain no spaces, and include the prefix `custpage` if you are adding the field to an existing page. For example, if you add a field that appears as Purchase Details, the field internal ID should be something similar to `custpage_purchasedetails` or `custpage_purchase_details`.

- `type` [string] [required] - The field type for this field. Use any of the following enumerated field types:
• text
• radio - See Working with Radio Buttons for details on adding this field type.
• label - This is a field type that has no values. It is used for placing a label next to another field. In Working with Radio Buttons, see the first code sample that shows how to set this field type and how it will render in the UI.
• email
• phone
• date
• datetimetz - This field type lets you combine date and time values in one field. For example, you may want a single field to contain date and time “timestamp” data. After a user enters a date/time value, the data is rendered in the user’s preferred date and time format, as well as the user’s time zone. Also note that time values are stored in NetSuite down to the second.
• currency
• float
• integer
• checkbox
• select
• url - See 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.

- **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.

```javascript
//Create a form called Interns.
var newForm = nlapiCreateForm('Interns');

//Add a tab to the Intern form.
var firstTab = newForm.addTab('custpage_academichistorytab', 'Academic History');

//Add a text field to the first tab.
newForm.addField('custpage_universityname', 'text', 'University Name', null, 'custpage_academichistorytab');

//Add a second tab to the Intern form.
var secondTab = newForm.addTab('custpage_studentcontacttab', 'Student Contact');

//Add an email field to the second tab.
newForm.addField('custpage_studentemail', 'email', 'Student Email', null, 'custpage_studentcontacttab');
```
## 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 simply 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 SuiteBundler Overview in the NetSuite Help Center).

<table>
<thead>
<tr>
<th>Record Type</th>
<th>Internal ID</th>
<th>Internal ID (number)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account</td>
<td>account</td>
<td>-112</td>
</tr>
<tr>
<td>Accounting Period</td>
<td>accountingperiod</td>
<td>-105</td>
</tr>
<tr>
<td>Call</td>
<td>phonecall</td>
<td>-22</td>
</tr>
<tr>
<td>Campaign</td>
<td>campaign</td>
<td>-24</td>
</tr>
<tr>
<td>Campaign Event</td>
<td>campaignevent</td>
<td>-107</td>
</tr>
<tr>
<td>Case</td>
<td>supportcase</td>
<td>-23</td>
</tr>
<tr>
<td>Class</td>
<td>classification</td>
<td>-101</td>
</tr>
<tr>
<td>Competitor</td>
<td>competitor</td>
<td>-108</td>
</tr>
<tr>
<td>Contact</td>
<td>contact</td>
<td>-6</td>
</tr>
<tr>
<td>Currency</td>
<td>customer</td>
<td>-122</td>
</tr>
<tr>
<td>Customer</td>
<td>customer</td>
<td>-2</td>
</tr>
<tr>
<td>Customer Category</td>
<td>customercategory</td>
<td>-109</td>
</tr>
<tr>
<td>Department</td>
<td>department</td>
<td>-102</td>
</tr>
<tr>
<td>Email Template</td>
<td></td>
<td>-120</td>
</tr>
<tr>
<td>Employee</td>
<td>employee</td>
<td>-4</td>
</tr>
<tr>
<td>Employee Type</td>
<td>employeetype</td>
<td>-111</td>
</tr>
<tr>
<td>Entity Status</td>
<td>customerstatus</td>
<td>-104</td>
</tr>
<tr>
<td>Event</td>
<td>calendarevent</td>
<td>-20</td>
</tr>
<tr>
<td>Field</td>
<td>customfield</td>
<td>-124</td>
</tr>
</tbody>
</table>
SuiteScript Developer and Reference Guide

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.

```javascript
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 );
}
```

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.

```javascript
form.assistant.getField('orgtype','b2c');
```

Setting Radio Fields

You can set the value of a radio button using either of these approaches:

```javascript
form.getField('orgtype', 'b2c').setDefaultValue('b2c');
```

- or -

```javascript
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.

```javascript
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

```javascript
function formWithFieldGroups(request, response) {
    if ( request.getMethod() == 'GET' ) {
        var form = nlapiCreateForm('Simple Form');
        var group = form.addFieldGroup( 'myfieldgroup', 'My Field Group');
        form.addField('companyname', 'text', 'Company Name', null,'myfieldgroup');
        form.addField('legalname', 'text', 'Legal Name', null,'myfieldgroup');
        form.addField('datefield', 'date', 'Date', null,'myfieldgroup' );
        form.addField('currencyfield', 'currency', 'Currency', null,'myfieldgroup');
        form.addField('textareafield', 'textarea', 'Textarea', null,'myfieldgroup');
        group setShowBorder(true);

        //Add a tab to the form.
        var firstTab = form.addTab('academichistorytab', 'Academic History');

        //Add a second tab to the form.
        var secondTab = form.addTab('studentcontacttab', 'Student Contact');

        //Add a field group to the first tab and align all field group fields in a single column
        var fieldGroupUniv = form.addFieldGroup("universityinfo", "Univeristy Information", 'academichistorytab');
        fieldGroupUniv.setSingleColumn(true);

        //Add fields to the University Information field group.
        form.addField('universityname', 'text', 'University Name', null, "universityinfo");
        form.addField('universityaddr', 'text', 'University Address', null, "universityinfo");

        //Add a field group to the second tab.
        form.addFieldGroup('studentinfogroup', "Student Information", 'studentcontacttab');

        //Add fields to the Student Information field group.
        form.addField('studentemail', 'email', 'Student Email', null, "studentcontacttab");
        form.addField('studentphone1', 'text', 'Student Phone 1', null, "studentcontacttab");
        form.addField('studentphone2', 'text', 'Student Phone 2', null, "studentcontacttab");
        form.addField('studentphone3', 'text', 'Student Phone 3', null, "studentcontacttab");
        form.addField('studentphone4', 'text', 'Student Phone 4', null, "studentcontacttab");

        form.addSubmitButton('Submit');
        response.writePage( form );
    }
}
```
else
dumpResponse(request,response);
}

Standard Objects | UI Objects | SuiteScript Functions

**addPageLink(type, title, url)**

Adds a navigation cross-link to the form

**Parameters**

- `type {string} [required]` - The type of navbar link to add. Possible values include:
  - `breadcrumb` - appears on top left corner after system bread crumbs
  - `crosslink` - appears on top right corner
- `title {string} [required]` - The text displayed in the link
- `url {string} [required]` - The URL used for this link

**Returns**

- `void`

**Since**

- Version 2008.2

Standard Objects | UI Objects | SuiteScript Functions
**addResetButton(label)**

Adds a reset button to a form

**Parameters**

- `label` {string} [optional] - The UI label used for this button. If no label is provided, the label defaults to *Reset*.

**Returns**

- `nlobjButton`

**Since**

- Version 2008.2

-----

**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.

```javascript
//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.
n newObj.addField('custpage_universityname', 'text', 'University Name', null, 'custpage_academichistorytab');

//Add a second tab to the Intern form.
var secondTab = newForm.addTab('custpage_studentcontacttab', 'Student Contact');

//Add an email field to the second tab.
n newObj.addField('custpage_studentemail', 'email', 'Student Email', null, 'custpage_studentcontacttab');
```
**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**

```javascript
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);
}
```

---

**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

---

**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

---

**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
**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

**getTabs()**

Returns an array of `nlobjTab` objects containing all the tabs in a form.

**Returns**

- `nlobjTab[]`

**Since**

- Version 2012.2

**Example**

This sample shows how to create a new form using `nlapiCreateForm`, add two tabs to the form using `nlobjForm.addTab`, add one field to each new tab using `nlobjForm.addField`, and then get all tabs in the new form using `nlobjForm.getTabs`.

```javascript
//Create a form called Interns.
var newForm = nlapiCreateForm('Interns');

//Add a tab to the Interns form.
var firstTab = newForm.addTab('custpage_academichistorytab', 'Academic History');

//Add a text field to the first tab.
newForm.addField('custpage_universityname', 'text', 'University Name', null, 'custpage_academichistorytab');

//Add a second tab to the Interns form.
var secondTab = newForm.addTab('custpage_studentcontacttab', 'Student Contact');

//Add an email field to the second tab.
newForm.addField('custpage_studentemail', 'email', 'Student Email', null, 'custpage_studentcontacttab');

//Get all tabs in the Interns form.
var tabs = newForm.getTabs();
```
**insertField(field, nextfld)**

Inserts a field (nlobjField) in front of another field and returns a reference to it

**Parameters**

- `field` {nlobjField} [required] - nlobjField object to insert
- `nextfield` {string} [required] - The name of the field you are inserting in front of

**Returns**

- nlobjField

**Since**

- Version 2008.2

**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

**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

---

**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

---

**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

```javascript
function removeUpdateOrderButton(type, form)
{
    form.removeButton('custpage_updateorderbutton');
}
```

**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

```javascript
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' });
```

**setScript(script)**

Sets the Client SuiteScript file used for this form

**Parameters**

- `script` {string | int} [required] - The scriptId or internal ID for the global client script used to enable Client SuiteScript on this form
Returns

- void

Since

- Version 2008.2

Standard Objects | UI Objects | SuiteScript Functions

setTitle(title)

Sets the title for this form

Parameters

- title {string} [required] - The title used for this form

Returns

- void

Since

- Version 2008.2

Standard Objects | UI Objects | SuiteScript Functions

nlobjList

Primary object used to encapsulate a list page. Note that the nlapiCreateList(title, hideNavbar) function returns a reference to this object.

Methods

- addButton(name, label, script)
- addColumn(name, type, label, align)
- addEditColumn(column, showView, showHrefCol)
- addPageLink(type, title, url)
- addRow(row)
- addRows(rows)
- setScript(script)
- setStyle(style)
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

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
- **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
SuiteScript 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.

**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 2008.2

---

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

---

### 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

---

### 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

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:

```javascript
function myPortlet(portlet, column)
{
    portlet.setTitle('Portlet Title');
    portlet.addLine('This is my SuiteScript portlet', null, 1);
}
```

Note: The column 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:

```javascript
function myPortlet(portlet)
{
    portlet.setTitle('Portlet Title');
    portlet.writeLine('This is my SuiteScript portlet', null, 1);
}
```

Once 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
• 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

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.
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

• 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.

Returns
• nlobjField

Since
• Version 2008.2

addLine(text, url, indent)

Adds a line (containing text or simple 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 simple 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 {hashable<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

---

**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

---

**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

- $n \{\text{int}\} [\text{required}]$ - 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

---

**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

- $scriptid \{\text{int | string}\} [\text{required}]$ - 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()**.

---

**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:

```javascript
portlet.setSubmitButton(nlapiResolveURL('SUITELET', 'customscript_suitelet', 'customdeploy_suitelet'), 'Save', '_hidden');
```

**Note:** The **target** parameter was added as of Version 2011.1.

**Returns**

- **nlobjButton**

**Since**

- Version 2008.2
setTitle(title)

Sets the portlet title

Parameters

- title {string} [required] - The title used for this portlet

Returns

- void

Since

- Version 2008.2

Example

```javascript
function demoSimpleFormPortlet(portlet, column)
{
    portlet.setTitle('Simple Form Portlet')
    //remainder of code...
}
```
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.

Once the form object is instantiated, you can add a new sublist to the form using the `nlobjForm.addSubList(name, type, label, tab)` method, which returns a reference to `nlobSubList`.

**nlobjSubList Methods**

- `addButton(name, label, script)`
- `addField(name, type, label, source)`
- `addMarkAllButtons()`
- `addRefreshButton()`
- `getLineItemCount(group)`
- `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 - **Important**: 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.

**Returns**

• nlobjField

**Since**

• Version 2008.2

---

**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

---

**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
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](https://help.netsuite.com/ncd/nt/2023-1/ScriptableSublists.html) 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

**Example**

```javascript
function request(request, response) {
    var form = nlapiCreateForm('myform');
    var list = form.addSubList('results', 'staticlist', 'My Sublist', 'resultsTab');
    list.addField("rownum","text","#");

    var i=1;
    for(var i=1; i< 10; i++)
        list.setLineItemValue("rownum", i, "val"+i);

    var count = list.getLineItemCount();
    form.addSubmitButton("Count#:"+count);

    response.writePage( form );
}
```

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

---

**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**

---

**setDisplayType(type)**

Sets the display style for this sublist. This method is only supported on scripted or static list 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, linenumber, value)**

Sets the value of a cell in a sublist field.
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
**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.

```javascript
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");
```

---

**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.

Once 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

---

**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

---
Chapter 58 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)
- nlapiDateToString(d, format)
- nlapiDeleteFile(id)
• nlapiDeleteRecord(type, id)
• 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()
• nlapiGetCurrentLineItemIndex(type)
• nlapiGetCurrentLineItemMatrixValue(type, fldnam, column)
• nlapiGetCurrentLineItemText(type, fldnam)
• nlapiGetCurrentLineItemValue(type, fldnam)
• nlapiGetCurrentLineItemValues(type, fldnam)
• nlapiGetDepartment()
• nlapiGetField(fldnam)
• nlapiGetFieldText(fldnam)
• nlapiGetFieldTexts(fldnam)
• nlapiGetFieldValue(fldnam)
• nlapiGetFieldValues(fldnam)
• nlapiGetLineItemCount(type)
• nlapiGetLineItemField(type, fldnamm, linenum)
• nlapiGetLineItemMatrixField(type, fldnam, linenum, column)
• nlapiGetLineItemMatrixValue(type, fldnam, linenum, column)
• nlapiGetLineItemText(type, fldnam, linenum)
• nlapiGetLineItemValue(type, fldnam, linenum)
- `nlapiGetLineItemValues(type, fldname, linenum)`
- `nlapiGetLocation()`
- `nlapiGetLogin()`
- `nlapiGetMatrixCount(type, fldnam)`
- `nlapiGetMatrixField(type, fldnam, column)`
- `nlapiGetMatrixValue(type, fldnam, column)`
- `nlapiGetNewRecord()`
- `nlapiGetOldRecord()`
- `nlapiGetRecordId()`
- `nlapiGetRecordType()`
- `nlapiGetRole()`
- `nlapiGetSubsidiary()`
- `nlapiGetUser()`
- `nlapiInitiateWorkflow(recordtype, id, workflowid)`
- `nlapiInsertLineItem(type, line)`
- `nlapiInsertLineItemOption(type, fldnam, value, text, selected)`
- `nlapiInsertSelectOption(fldnam, value, text, selected)`
- `nlapiIsLineItemChanged(type)`
- `nlapiLoadFile(id)`
- `nlapiLoadRecord(type, id, initializeValues)`
- `nlapiLoadSearch(type, id)`
- `nlapiLogExecution(type, title, details)`
- `nlapiLookupField(type, id, fields, text)`
- `nlapiMergeRecord(id, baseType, baseId, altType, altId, fields)`
- `nlapiMergeTemplate(id, baseType, baseId, altType, altId, fields)`
- `nlapiOutboundSSO(id)`
- `nlapiPrintRecord(type, id, mode, properties)`
- `nlapiRefreshLineItems(type)`
- `nlapiRefreshPortlet()`
- `nlapiRemoveCurrentLineItemSubrecord(sublist, fldname)`
• nlapiRemoveLineItem(type, line)
• nlapiRemoveLineItemOption(type, fldnam, value)
• nlapiRemoveSelectOption(fldnam, value)
• nlapiRemoveSubrecord(fldname)
• nlapiRequestURL(url, postdata, headers, callback, httpMethod)
• nlapiRequestURLWithCredentials(credentials, url, postdata, headers, httpMethod)
• 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)
• nlapiSendFax(author, recipient, subject, body, records, attachments)
• 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)
• nlapiSetFieldText(fldname, txt, firefieldchanged, synchronous)
• nlapiSetFieldTexts(fldname, txts, firefieldchanged, synchronous)
• nlapiSetFieldValue(fldnam, value, firefieldchanged, synchronous)
• nlapiSetFieldValues(fldnam, value, firefieldchanged, synchronous)
• 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)`
- `nlapiViewCurrentLineItemSubrecord(sublist, fldname)`
- `nlapiViewLineItemSubrecord(sublist, fldname, linenum)`
- `nlapiViewSubrecord(fldname)`
- `nlapiXMLToPDF(xmlstring)`
- `nlapiXMLToString(xml)`
- `nlapiYieldScript()`

**Objects**

- `nlobjAssistant`
- `nlobjAssistantStep`
- `nlobjButton`
- `nlobjColumn`
- `nlobjConfiguration`
- `nlobjContext`
- `nlobjCSVImport`
- `nlobjError`
- `nlobjField`
- `nlobjFieldGroup`
- `nlobjFile`
- `nlobjForm`
- nlobjList
- nlobjLogin
- nlobjPivotColumn
- nlobjPivotRow
- nlobjPivotTable
- nlobjPivotTableHandle
- nlobjPortlet
- nlobjRecord
- nlobjReportColumn
- nlobjReportColumnHierarchy
- nlobjReportDefinition
- nlobjReportForm
- nlobjReportRowHierarchy
- nlobjResponse
- nlobjRequest
- nlobjSearch
- nlobjSearchColumn
- nlobjSearchFilter
- nlobjSearchResult
- nlobjSelectOption
- nlobjSubList
- nlobjSubrecord
- nlobjTab
Part 11  SuiteScript Reference
Chapter 59 SuiteScript Reference

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

- SuiteScript References Overview
- Using the SuiteScript Records Browser

**SuiteScript References Overview**

The NetSuite Help Center provides all the internal IDs you will need when referencing NetSuite records, fields, sublists, search filters, permissions, features, 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 Using 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.

- Scriptable Sublists
- Transaction Type IDs
- Permission Names and IDs
- Feature Names and IDs
- Preference Names and IDs
- Supported File Types
- Supported Tasklinks
- SuiteScript Errors
Using the SuiteScript Records Browser

The SuiteScript Records Browser provides a Web-based view of all records, fields, sublists, search joins, search filters, search columns, and record transformations that are supported in SuiteScript.

To use the SuiteScript Records Browser:

1. If you are logged in to NetSuite, click any link to the SuiteScript Records Browser from within the NetSuite Help Center. The SuiteScript Records Browser opens in a new browser window.

   Note: If you want to use the SuiteScript Records Browser when you are not logged in to NetSuite, see Using the SuiteScript Records Browser Offline.

2. Click the record you want to reference in SuiteScript (see figure). You can jump to the bottom of the table by clicking a letter from the alphabetized list at the top.

3. The figure below shows an example of the record data that appears. You will see:
   - Name: The name of the record as it appears in the UI, for example Assembly Build.
   - Internal ID: The internal ID for the record, for example assemblybuild (this is the ID you will reference in SuiteScript when using Record APIs).
   - The internal IDs for all SuiteScript-supported fields, tabs, sublists, search joins, search filters, search columns, and transformation types (if the record can be transformed into another record type).
   - The Supported in nlapiSubmitField column.
     - true: This means if you reference the field in nlapiSubmitField(type, id, fields, values, doSourcing), the amount of SuiteScript governance units consumed will only be as high as 10 units.
     - false: This means if you reference the field in nlapiSubmitField(type, id, fields, values, doSourcing), the single nlapiSubmitField call could consume 30 SuiteScript governance units. The nlapiSubmitField function is meant for only inline editable fields. If you pass in a non-inline editable field to
nlapiSubmitField, your code will still work, however, the call will be charged more units, as NetSuite must do a complete load record, set field, submit record process. See Consequences of Using nlapiSubmitField on Non Inline Editable Fields for complete details.

**Record**

**Name:** Assembly Build  
**Internal ID:** assemblybuild

<table>
<thead>
<tr>
<th>Fields</th>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Type</th>
<th>Required</th>
<th>Supported in nlapiSubmitField</th>
</tr>
</thead>
<tbody>
<tr>
<td>buildable</td>
<td>Buildable Quantity</td>
<td>float</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>class</td>
<td>Class</td>
<td>select</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>createddate</td>
<td>Created Date</td>
<td>date</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>customform</td>
<td>Custom Form</td>
<td>select</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Department</td>
<td>Department</td>
<td>select</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>externalid</td>
<td>External ID</td>
<td>text</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>inventorydetail</td>
<td>Inventory Detail</td>
<td>summary</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>item</td>
<td>Assembly</td>
<td>select</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

**Note:** For information on governance applied to individual SuiteScript APIs, as well as to various SuiteScript script types, see [SuiteScript Governance](#).

**Using the SuiteScript Records Browser Offline**

If you want to use the SuiteScript Records Browser without having to log in to your NetSuite account, you must download the SuiteScript Records Browser .zip file.

After downloading the .zip file to your local machine, extract the .zip and click the `index.html` file.
# Chapter 60 SuiteScript Supported Records

The following table lists all NetSuite records that support SuiteScript. Also provided are record internal 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.

<table>
<thead>
<tr>
<th>Record Name</th>
<th>Record Internal ID</th>
<th>Record Category</th>
<th>Scripting Level</th>
<th>Scriptable in Client SuiteScript</th>
<th>Scriptable in Server SuiteScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account</td>
<td>account</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Activity</td>
<td>activity</td>
<td>Activity</td>
<td>Search Only</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>Assembly Build</td>
<td>assemblybuild</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Assembly Unbuild</td>
<td>assemblyunbuild</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Bin</td>
<td>bin</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Bin Putaway Worksheet</td>
<td>binworksheet</td>
<td>Transaction</td>
<td>Copy and Update Not Allowed</td>
<td>Dynamic Mode Only</td>
<td></td>
</tr>
<tr>
<td>Bin Transfer</td>
<td>bintransfer</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Build/Assembly</td>
<td>assemblyitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Campaign</td>
<td>campaign</td>
<td>Marketing</td>
<td>Full</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>Case</td>
<td>supportcase</td>
<td>Support</td>
<td>Full</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>Cash Refund</td>
<td>cashrefund</td>
<td>Transaction</td>
<td>Full</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>Cash Sale</td>
<td>cashsale</td>
<td>Transaction</td>
<td>Full</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>Check</td>
<td>check</td>
<td>Transaction</td>
<td>Full</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>classification</td>
<td>List</td>
<td>Full</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>Competitor</td>
<td>competitor</td>
<td>Entity</td>
<td>Full</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>Contact</td>
<td>contact</td>
<td>Entity</td>
<td>Full</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>Coupon Code</td>
<td>couponcode</td>
<td>Marketing</td>
<td>Full</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>Credit Memo</td>
<td>creditmemo</td>
<td>Transaction</td>
<td>Full</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>Currency</td>
<td>currency</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Customer</td>
<td>customer</td>
<td>Entity</td>
<td>Full</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>Customer Category</td>
<td>customercategory</td>
<td>List</td>
<td>Search Not Available</td>
<td>X X</td>
<td></td>
</tr>
<tr>
<td>Record Name</td>
<td>Record Internal ID</td>
<td>Record Category</td>
<td>Scripting Level</td>
<td>Scriptable in Client SuiteScript</td>
<td>Scriptable in Server SuiteScript</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------</td>
<td>-----------------</td>
<td>-----------------</td>
<td>----------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Customer Deposit</td>
<td>customerdeposit</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Customer Payment</td>
<td>customerpayment</td>
<td>Transaction</td>
<td>Copy and Create Not Allowed</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Customer Refund</td>
<td>customerrefund</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Custom List</td>
<td>customlist*</td>
<td>Custom</td>
<td>Search Only</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Custom Record</td>
<td>customrecord*</td>
<td>Custom</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Department</td>
<td>department</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Deposit Application</td>
<td>depositapplication</td>
<td>Transaction</td>
<td>Create Not Allowed</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Description</td>
<td>descriptionitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Discount</td>
<td>discountitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Download Item</td>
<td>downloaditem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Employee</td>
<td>employee</td>
<td>Entity</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Entity</td>
<td>entity</td>
<td>Entity</td>
<td>Search Only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimate / Quote</td>
<td>estimate</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Event</td>
<td>calendarevent</td>
<td>Activity</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Expense Category</td>
<td>expensecategory</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Expense Report</td>
<td>expensereport</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Folder</td>
<td>folder</td>
<td>File Cabinet</td>
<td>Full</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Gift Certificate</td>
<td>giftcertificate</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Gift Certificate Item</td>
<td>giftcertificateitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Intercompany Journal Entry</td>
<td>intercompanyjournale</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Inventory Adjustment</td>
<td>inventoryadjustment</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Inventory Detail</td>
<td>inventorydetail</td>
<td>Subrecord</td>
<td>Must be created in the context of a parent record. See Scripting the Inventory Detail Subrecord for more information.</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Inventory Item</td>
<td>inventoryitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>(Also referred to in the UI as Inventory Part)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record Name</td>
<td>Record Internal ID</td>
<td>Record Category</td>
<td>Scripting Level</td>
<td>Scriptable in Client SuiteScript</td>
<td>Scriptable in Server SuiteScript</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------------------------</td>
<td>-----------------</td>
<td>------------------------------------------------------</td>
<td>----------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Inventory Number</td>
<td>inventorynumber</td>
<td>List</td>
<td>Copy, Create, and Delete Not Allowed</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Inventory Transfer</td>
<td>inventorytransfer</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Invoice</td>
<td>invoice</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Issue</td>
<td>issue</td>
<td>Support</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Item Search</td>
<td>item</td>
<td>Item</td>
<td>Search Only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item Demand Plan</td>
<td>itemdemandplan</td>
<td>Transaction</td>
<td>Copy Not Allowed, No Available Transforms</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Item Fulfillment</td>
<td>itemfulfillment</td>
<td>Transaction</td>
<td>Copy and Create Not Allowed</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Item Group</td>
<td>itemgroup</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Item Receipt</td>
<td>itemreceipt</td>
<td>Transaction</td>
<td>Copy and Create Not Allowed</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Item Revision</td>
<td>itemrevision</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Item Supply Plan</td>
<td>itemsupplyplan</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Journal Entry</td>
<td>journalentry</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Kit</td>
<td>kititem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Lead</td>
<td>lead</td>
<td>Entity</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Location</td>
<td>location</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Lot Numbered Assembly Item</td>
<td>lotnumberedassemblyitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Lot Numbered Inventory Item</td>
<td>lotnumberedinventoryitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Markup</td>
<td>markupitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Message</td>
<td>message</td>
<td>Communication</td>
<td>Full</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Nexus</td>
<td>nexus</td>
<td>List</td>
<td>Copy Not Allowed</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Non-Inventory Part</td>
<td>noninventoryitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Note</td>
<td>note</td>
<td>Communication</td>
<td>Full</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Opportunity</td>
<td>opportunity</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Other Charge Item</td>
<td>otherchargeitem</td>
<td>Item</td>
<td>Full</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other Name</td>
<td>othername</td>
<td>Entity</td>
<td>Full</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Record Name</td>
<td>Record Internal ID</td>
<td>Record Category</td>
<td>Scripting Level</td>
<td>Scriptable in Client SuiteScript</td>
<td>Scriptable in Server SuiteScript</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------</td>
<td>-----------------</td>
<td>----------------</td>
<td>----------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Partner</td>
<td>partner</td>
<td>Entity</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Paycheck Journal</td>
<td>paycheckjournal</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Payment</td>
<td>paymentitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Payroll Item</td>
<td>payrollitem</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Phone Call</td>
<td>phonecall</td>
<td>Activity</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Price Level</td>
<td>pricelevel</td>
<td>Lists</td>
<td>Search Not Available</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Project</td>
<td>job</td>
<td>Entity</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Project Task</td>
<td>projecttask</td>
<td>Event</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Promotion</td>
<td>promotioncode</td>
<td>Marketing</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Prospect</td>
<td>prospect</td>
<td>Entity</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Purchase Order</td>
<td>purchaseorder</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Reallocate Items</td>
<td>reallocateitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Return Authorization</td>
<td>returnauthorization</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Revenue Commitment</td>
<td>revenuecommitment</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Revenue Commitment Reversal</td>
<td>revenuecommitmenr</td>
<td>Transaction</td>
<td>Create Not Allowed. See Usage Notes.</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Revenue Recognition Schedule</td>
<td>revrecschedule</td>
<td>List</td>
<td>Copy, Create, and Delete Not Allowed</td>
<td>X</td>
<td>(User Event Scripts Not Supported)</td>
</tr>
<tr>
<td>Revenue Recognition Template</td>
<td>revrectemplate</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td>(User Event Scripts Not Supported)</td>
</tr>
<tr>
<td>Sales Order</td>
<td>salesorder</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Sales Tax Item</td>
<td>salestaxitem</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Serialized Assembly Item</td>
<td>serializedassemblyitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Serialized Inventory Item</td>
<td>serializedinventoryitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Service</td>
<td>serviceitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Solution</td>
<td>solution</td>
<td>Support</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Subsidiary</td>
<td>subsidiary</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Record Name</td>
<td>Record Internal ID</td>
<td>Record Category</td>
<td>Scripting Level</td>
<td>Scriptable in Client SuiteScript</td>
<td>Scriptable in Server SuiteScript</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------</td>
<td>-----------------</td>
<td>-----------------</td>
<td>----------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>Subtotal</td>
<td>subtotalitem</td>
<td>Item</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Task</td>
<td>task</td>
<td>Activity</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Tax Group</td>
<td>taxgroup</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Tax Period</td>
<td>taxperiod</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Tax Type</td>
<td>taxtype</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Term</td>
<td>term</td>
<td>List</td>
<td>Search Not Available</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Time</td>
<td>timebill</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Topic</td>
<td>topic</td>
<td>Support</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Transaction Search</td>
<td>transaction</td>
<td>Transaction</td>
<td>Search Only</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Transfer Order</td>
<td>transferorder</td>
<td>Transaction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unit of Measure</td>
<td>unitstype</td>
<td>List</td>
<td>Full</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Vendor</td>
<td>vendor</td>
<td>Entity</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Vendor Bill</td>
<td>vendorbill</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Vendor Category</td>
<td>vendorcategory</td>
<td>List</td>
<td>Search Not Available</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Vendor Credit</td>
<td>vendorcredit</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Vendor Payment</td>
<td>vendorpayment</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Vendor Return</td>
<td>vendorreturnauthorization</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Authorization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Work Order</td>
<td>workorder</td>
<td>Transaction</td>
<td>Full</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**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 Setup > Customization > Record Types. For custom list IDs, go to Setup > Customization > 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

The following activity records are scriptable in SuiteScript:

- Activity
- Event
- Phone Call
- Project Task
- Task
- Work Calendar

Activity

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Event

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Phone Call

The internal ID for this record is `phonecall`.

See the SuiteScript Records Browser for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Project Task

The internal ID for this record is `projecttask`.

See the SuiteScript Records Browser for all internal IDs associated with this record.
**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

**Usage Notes**

When creating new Project Tasks you must set the Project (company) field to the project/job ID. Project Tasks are not stand-alone records, and therefore must be associated with a specific project/job.

**Task**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Work Calendar**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.
Entities

The following entity records are scriptable in SuiteScript:

- Competitor
- Contact
- Customer
- Employee
- Entity
- Project
- Lead
- Other Name
- Partner
- Prospect
- Vendor

**Competitor**

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 [Using the SuiteScript Records Browser](#) in the NetSuite Help Center.

**Contact**

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 [Using the SuiteScript Records Browser](#) in the NetSuite Help Center.

**Customer**

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 Using 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.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>datecreated</td>
<td>Date Created</td>
<td>This is a system-generated field that marks the date the record was created in NetSuite. You cannot change or override this field.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>password</td>
<td>Password</td>
<td>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).</td>
</tr>
<tr>
<td>password2</td>
<td>Confirm Password</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Search Filters and Search Columns**

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccnumber</td>
<td>Credit Card Number</td>
<td>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).</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Target Record Type</th>
<th>Target Record Internal ID</th>
<th>Field Defaults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cash Sale</td>
<td>cashsale</td>
<td>billdate</td>
</tr>
<tr>
<td>Customer Payment</td>
<td>customerpayment</td>
<td></td>
</tr>
<tr>
<td>Quote</td>
<td>estimate</td>
<td></td>
</tr>
<tr>
<td>Invoice</td>
<td>invoice</td>
<td>billdate</td>
</tr>
<tr>
<td>Opportunity</td>
<td>opportunity</td>
<td></td>
</tr>
<tr>
<td>Sales Order</td>
<td>salesorder</td>
<td></td>
</tr>
</tbody>
</table>

Employee

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Usage Notes

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>Password</td>
<td>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).</td>
</tr>
<tr>
<td>password2</td>
<td>Confirm Password</td>
<td>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).</td>
</tr>
</tbody>
</table>

Transform Types

In the NetSuite Help Center, see `nlapiTransformRecord(type, id, transformType, transformValues)` for examples on how to transform records.
Entity

The internal ID for this record is `entity`.

See the SuiteScript Records Browser for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Project

The internal ID for this record is `job`.

See the SuiteScript Records Browser for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Usage Notes

To use the Project record you must have the Projects feature enabled. Go to Setup > Company > Enable Features. Click the Company tab, and select **Projects**.

If you plan to do advanced project tracking, you must enable both **Projects** and **Advanced Projects**. If you do not see the Advanced Projects check box, your company must first purchase the Advanced Projects add-on from NetSuite.

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>datecreated</td>
<td>Date Created</td>
<td>This is a system-generated field that marks the date the record was created in NetSuite. You cannot change or override this field. 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.</td>
</tr>
</tbody>
</table>

Transform Types

In the NetSuite Help Center, see `nlapiTransformRecord(type, id, transformType, transformValues)` for examples on how to transform records.
Lead

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Usage Notes**

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>datecreated</td>
<td>Date Created</td>
<td>This is a system-generated field that marks the date the record was created in NetSuite. You cannot change or override this field.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>password</td>
<td>Password</td>
<td>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).</td>
</tr>
<tr>
<td>password2</td>
<td>Confirm Password</td>
<td>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).</td>
</tr>
</tbody>
</table>
**Transform Types**

In the NetSuite Help Center, see `nlapiTransformRecord(type, id, transformType, transformValues)` for examples on how to transform records.

<table>
<thead>
<tr>
<th>Target Record Name</th>
<th>Target Record Internal ID</th>
<th>Field Defaults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opportunity</td>
<td>opportunity</td>
<td></td>
</tr>
</tbody>
</table>

**Other Name**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Partner**

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 Using 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.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>Password</td>
<td>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).</td>
</tr>
<tr>
<td>password2</td>
<td>Confirm Password</td>
<td>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).</td>
</tr>
</tbody>
</table>

**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.

**Prospect**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Usage Notes**

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>Password</td>
<td>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).</td>
</tr>
<tr>
<td>password2</td>
<td>Confirm Password</td>
<td>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).</td>
</tr>
<tr>
<td>ccnumber</td>
<td>Credit Card Number</td>
<td>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).</td>
</tr>
</tbody>
</table>
Transform Types

In the NetSuite Help Center, see `nlapiTransformRecord(type, id, transformType, transformValues)` for examples on how to transform records.

<table>
<thead>
<tr>
<th>Target Record Name</th>
<th>Target Record Internal ID</th>
<th>Field Defaults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimate/Quote</td>
<td>estimate</td>
<td></td>
</tr>
<tr>
<td>Opportunity</td>
<td>opportunity</td>
<td></td>
</tr>
<tr>
<td>Sales Order</td>
<td>salesorder</td>
<td></td>
</tr>
</tbody>
</table>

Vendor

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Usage Notes

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>Password</td>
<td>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).</td>
</tr>
<tr>
<td>password2</td>
<td>Confirm Password</td>
<td>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).</td>
</tr>
</tbody>
</table>

Transform Types

In the NetSuite Help Center, see `nlapiTransformRecord(type, id, transformType, transformValues)` for examples on how to transform records.

<table>
<thead>
<tr>
<th>Target Record Name</th>
<th>Target Record Internal ID</th>
<th>Field Defaults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purchase Order</td>
<td>purchaseorder</td>
<td></td>
</tr>
<tr>
<td>Target Record Name</td>
<td>Target Record Internal ID</td>
<td>Field Defaults</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Vendor Bill</td>
<td>vendorbill</td>
<td></td>
</tr>
<tr>
<td>Vendor Payment</td>
<td>vendorpayment</td>
<td></td>
</tr>
</tbody>
</table>
Communications

The following communication records are scriptable in SuiteScript:

- Message
- Note

**Message**

The internal ID for this record is *message*.

See the SuiteScript Records Browser for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

**User Notes**

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:

```javascript
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 as a result of responding to an emailed case.

**Note**

The internal ID for this record is *note*.

See the SuiteScript Records Browser for all internal IDs associated with this record.
Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.
**Items**

The following item records are scriptable in SuiteScript:

- Build/Assembly
- 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
- Subtotal

**Using Item Records in SuiteScript**

This section includes the following topics:

- Loading Item Types
- Filtering Items by Type
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.

<table>
<thead>
<tr>
<th>Item Type IDs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly</td>
<td>Markup</td>
</tr>
<tr>
<td>Description</td>
<td>NonInvtPart</td>
</tr>
<tr>
<td>Discount</td>
<td>OthCharge</td>
</tr>
<tr>
<td>DwnLdItem</td>
<td>Payment</td>
</tr>
<tr>
<td>EndGroup</td>
<td>Service</td>
</tr>
<tr>
<td>GiftCert</td>
<td>Shiptem</td>
</tr>
<tr>
<td>Group</td>
<td>Subtotal</td>
</tr>
<tr>
<td>InvtPart</td>
<td>TaxGroup</td>
</tr>
<tr>
<td>Kit</td>
<td>TaxItem</td>
</tr>
</tbody>
</table>

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

```javascript
//Create a script that will search for all non-inventory part items
function searchnoninventorypart()
{
    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);
}
```

Build/Assembly

The internal ID for this record is `assemblyitem`. 
See the SuiteScript Records Browser for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see Using 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.

### Description

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

### Discount

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

### Download Item

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

### Gift Certificate Item

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

### Inventory Item

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 Using the SuiteScript Records Browser in the NetSuite Help Center.
Item Group

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Item Search

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Kit

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Lot Numbered Assembly Item

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Lot Numbered Inventory Item

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Markup

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 *Using the SuiteScript Records Browser* in the NetSuite Help Center.

**Non-Inventory Part**

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 *Using the SuiteScript Records Browser* in the NetSuite Help Center.

**Other Charge Item**

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 *Using the SuiteScript Records Browser* in the NetSuite Help Center.

**Payment**

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 *Using the SuiteScript Records Browser* in the NetSuite Help Center.

**Reallocate Items**

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 *Using the SuiteScript Records Browser* in the NetSuite Help Center.

**Usage Notes**

This record does not support client scripts.

The record supports only user event scripts. You can execute beforeLoad, beforeSubmit, and afterSubmit user event scripts on this record.

**Serialized Assembly Item**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

### Serialized Inventory Item

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

### Service

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

### Subtotal

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 Using the SuiteScript Records Browser in the NetSuite Help Center.
Transactions

The following transaction records are scriptable in SuiteScript.

- Assembly Build
- Assembly Unbuild
- Bin Putaway Worksheet
- Bin Transfer
- Cash Refund
- Cash Sale
- Check
- Credit Memo
- Customer Deposit
- Customer Payment
- Customer Refund
- Deposit Application
- Estimate / Quote
- Expense Report
- Intercompany Journal Entry
- Inventory Adjustment
- Inventory Detail
- Inventory Transfer
- Invoice
- Item Demand Plan
- Item Fulfillment
- Item Receipt
- Item Supply Plan
- Journal Entry
- Opportunity
- Paycheck Journal
- Purchase Order
- Return Authorization
- Revenue Commitment
- Revenue Commitment Reversal
- Sales Order
- Time
- Transaction Search
- Transfer Order
- Vendor Bill
- Vendor Credit
- Vendor Payment
- Vendor Return Authorization
- Work Order

**Assembly Build**

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 Using 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**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Bin Putaway Worksheet**

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 Using 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
- Do I have to 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

The internal ID for this record is \textit{bintransfer}.

See the SuiteScript Records Browser for all internal IDs associated with this record.

\textbf{Note:} For information on using the SuiteScript Records Browser, see Using 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.

Cash Refund

The internal ID for this record is \textit{cashrefund}.

See the SuiteScript Records Browser for all internal IDs associated with this record.

\textbf{Note:} For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Usage Notes

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{ccnumber}</td>
<td>Credit Card #</td>
<td>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).</td>
</tr>
</tbody>
</table>
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 Using the SuiteScript Records Browser in the NetSuite Help Center.

### Usage Notes

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccnumber</td>
<td>Credit Card #</td>
<td>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).</td>
</tr>
<tr>
<td>estgrossprofit</td>
<td>Est. Gross Profit</td>
<td>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.</td>
</tr>
</tbody>
</table>

---

**Cash Sale**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

### Usage Notes

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccnumber</td>
<td>Credit Card #</td>
<td>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).</td>
</tr>
<tr>
<td>estgrossprofit</td>
<td>Est. Gross Profit</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Check

The internal ID for this record is `check`.

See the SuiteScript Records Browser for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

### Usage Notes

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>estgrossprofitpercent</td>
<td>Est. Gross Profit Percent</td>
<td>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.</td>
</tr>
<tr>
<td>totalcostestimate</td>
<td>Est. Extended Cost</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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 Memo

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 Using the SuiteScript Records Browser in the NetSuite Help Center.
**Customer Deposit**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Usage Notes**

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccnumber</td>
<td>Credit Card #</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Customer Payment**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Usage Notes**

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccnumber</td>
<td>Credit Card #</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Customer Refund**

The internal ID for this record is `customerrefund`.

See the SuiteScript Records Browser for all internal IDs associated with this record.
For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

**Usage Notes**

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body Fields</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ccnumber</td>
<td>Credit Card #</td>
<td>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).</td>
</tr>
<tr>
<td>estgrossprofit</td>
<td>Est. Gross Profit</td>
<td>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.</td>
</tr>
<tr>
<td>estgrossprofitpercent</td>
<td>Est. Gross Profit Percent</td>
<td>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.</td>
</tr>
<tr>
<td>totalcostestimate</td>
<td>Est. Extended Cost</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Deposit Application**

The internal ID for this record is `depositapplication`.

In SuiteScript, you do not use the `nlapiCreateRecord(...)` to create a Deposit Application record. Deposit applications are always created as a result of applying a Customer Deposit 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.

For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.
**Estimate / Quote**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Usage Notes**

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Body Fields</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>estgrossprofit</td>
<td>Est. Gross Profit</td>
<td>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.</td>
</tr>
<tr>
<td>estgrossprofitpercent</td>
<td>Est. Gross Profit Percent</td>
<td>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.</td>
</tr>
<tr>
<td>totalcostestimate</td>
<td>Est. Extended Cost</td>
<td>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.</td>
</tr>
</tbody>
</table>

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**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Intercompany Journal Entry**

The internal ID for this record is *intercompanyjournalentry*.

See the SuiteScript Records Browser for all internal IDs associated with this record.
Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Inventory Adjustment

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Inventory Detail

The internal ID for this record is `inventorydetail`.

See the SuiteScript Records Browser for all internal IDs associated with this record.

Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Usage Notes

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

The internal ID for this record is `inventorytransfer`.

See the SuiteScript Records Browser for all internal IDs associated with this record.

Note: For information on using the SuiteScript Records Browser, see Using 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

The internal ID for this record is `invoice`.

See the SuiteScript Records Browser for all internal IDs associated with this record.
**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

**Usage Notes**

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>estgrossprofit</td>
<td>Est. Gross Profit</td>
<td>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.</td>
</tr>
<tr>
<td>estgrossprofitpercent</td>
<td>Est. Gross Profit Percent</td>
<td>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.</td>
</tr>
<tr>
<td>totalcostestimate</td>
<td>Est. Extended Cost</td>
<td>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.</td>
</tr>
</tbody>
</table>

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**

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 Using 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.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Body Fields</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subsidiary</td>
<td>Subsidiary</td>
<td>Required in OneWorld accounts.</td>
</tr>
<tr>
<td>location</td>
<td>Location</td>
<td>Required when the Multi-Location Inventory feature enabled.</td>
</tr>
<tr>
<td>item</td>
<td>Item</td>
<td>Required. Can only use items with supplyreplenishment method set to Time Phased.</td>
</tr>
<tr>
<td>units</td>
<td>Unit of Measure</td>
<td>Optional. Available when the Multiple Units of Measure feature is enabled.</td>
</tr>
<tr>
<td>memo</td>
<td>Memo</td>
<td>Optional.</td>
</tr>
<tr>
<td>startdate</td>
<td>Start Date</td>
<td>Optional. Defaults to the first day of the current year, for example for 2011, defaults to 1/1/2011.</td>
</tr>
<tr>
<td>enddate</td>
<td>End Date</td>
<td>Optional. Defaults to the last day of the current year, for example for 2011, defaults to 12/31/2011.</td>
</tr>
<tr>
<td>demandplancalendartype</td>
<td>View</td>
<td>Required. Valid values are MONTHLY, WEEKLY, or DAILY. (Must use all capital letters.)</td>
</tr>
<tr>
<td><strong>Matrix Fields</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>quantity</td>
<td>Quantity</td>
<td>For monthly and weekly demand plans, each row has one quantity column.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For daily demand plans, each row has seven quantity columns.</td>
</tr>
</tbody>
</table>
Item Demand Plan

Note: It is recommended that you work with the Item Demand Plan record in dynamic mode. See Working with Records in Dynamic Mode.

Item Fulfillment

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 Using 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

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Usage Notes

The following usage notes are included for this record:

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
</table>
| startdate         | Start Date    | System-calculated, read-only values.  
|                   |               | • For monthly plans, the date of the first day of the month that the row represents.  
|                   |               | • 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 > Company > General Preferences.  |
| enddate           | End Date      | System-calculated, read-only values.  
|                   |               | • For monthly plans, the date of the last day of the month that the row represents.  
|                   |               | • 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 > Company > General Preferences.  |
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:

```javascript
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:

```javascript
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 given 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

The internal ID for this record is `itemsupplyplan`.

This record includes the Orders Sublist.

This record will be added soon to the SuiteScript Records Browser.
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.

```javascript
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.

```javascript
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

The internal ID for this record is `journalentry`.

See the SuiteScript Records Browser for all internal IDs associated with this record.
**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

**Opportunity**

The internal ID for this record is `opportunity`.

See the SuiteScript Records Browser for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

**Usage Notes**

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>estgrossprofit</code></td>
<td>Est. Gross Profit</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When this field appears on the sublist line level, this field is not scriptable.</td>
</tr>
<tr>
<td><code>estgrossprofitpercent</code></td>
<td>Est. Gross Profit Percent</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When this field appears on the sublist line level, this field is not scriptable.</td>
</tr>
<tr>
<td><code>totalcostestimate</code></td>
<td>Est. Extended Cost</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

**Paycheck Journal**

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 Using 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.

```javascript
//create Paycheck Journal with earnings and deduction sublist
// add 2 Earning and 1 deduction sublist

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.

```javascript
//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 update

  for (var lineNo=1; lineNo <= pj.getLineItemCount('deduction'); lineNo++)
    pj.removeLineItem('deduction', lineNo );

  nlapiSubmitRecord(pj);
}
```
Purchase Order

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 Using 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.

Return Authorization

The internal ID for this record is `returnauthorization`.

See the SuiteScript Records Browser for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see Using 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 Commitment

The internal ID for this record is `revenuecommitment`.

**Note:** For information on using the SuiteScript Records Browser, see Using 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

The internal ID for this record is `revenuecommitmentreversal`. 
**Note:** For information on using the SuiteScript Records Browser, see Using 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**

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 Using 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.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>estgrossprofit</td>
<td>Est. Gross Profit</td>
<td>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.</td>
</tr>
<tr>
<td>estgrossprofitpercent</td>
<td>Est. Gross Profit Percent</td>
<td>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.</td>
</tr>
<tr>
<td>totalcostestimate</td>
<td>Est. Extended Cost</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.
Time

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Transaction Search

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Usage Notes

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccnumber</td>
<td>Credit Card #</td>
<td>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).</td>
</tr>
<tr>
<td>entity</td>
<td>Name</td>
<td>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.</td>
</tr>
</tbody>
</table>

Transfer Order

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Vendor Bill

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Usage Notes**

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>usertotal</td>
<td>Amount</td>
<td>This field is not available via search or lookup for any transactions.</td>
</tr>
</tbody>
</table>

**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.

**Vendor Credit**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Usage Notes**

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>usertotal</td>
<td>Amount</td>
<td>This field is not available via search or lookup for any transactions.</td>
</tr>
</tbody>
</table>

**Vendor Payment**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.
Vendor Return Authorization

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Usage Notes

The following table provides usage notes for specific fields on this record.

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>usertotal</td>
<td>Amount</td>
<td>This field is not available via search or lookup for any transactions.</td>
</tr>
</tbody>
</table>

Work Order

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 Using the SuiteScript Records Browser in the NetSuite Help Center.
Support

The following records are scriptable in SuiteScript:

- Case
- Issue
- Solution
- Task
- Topic

Case

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Issue

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Solution

The internal ID for this record is `solution`.

See the SuiteScript Records Browser for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Task

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 Using the SuiteScript Records Browser in the NetSuite Help Center.
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 Using the SuiteScript Records Browser in the NetSuite Help Center.
File Cabinet

The following records are scriptable in SuiteScript:

- Folder

Folder

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 Using the SuiteScript Records Browser in the NetSuite Help Center.
The following list records are scriptable in SuiteScript:

- Account
- Bin
- Class
- Currency
- Customer Category
- Department
- Expense Category
- Gift Certificate
- Inventory Number
- Item Revision
- Location
- Nexus
- Payroll Item
- Price Level
- Revenue Recognition Schedule
- Revenue Recognition Template
- Sales Tax Item
- Subsidiary
- Tax Group
- Tax Period
- Tax Type
- Term
- Unit of Measure
- Vendor Category
Account

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Bin

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 Using 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

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Currency

The internal ID for this record is *currency*.

See the SuiteScript Records Browser for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Customer Category

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Department

The internal ID for this record is *department*. 
Expense Category

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Gift Certificate

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Inventory Number

The internal ID for this record is inventorynumber.

See the SuiteScript Records Browser for all internal IDs associated with this record.

**Note**: For information on using the SuiteScript Records Browser, see Using 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 Revision

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 Using 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:

```javascript
function afterSubmit(type) {
    var itemrev = nlapiCreateRecord("itemrevision");

    itemrev.setFieldValu("name", "revision name 222");
    itemrev.setFieldValu("item", "109");
    itemrev.setFieldValu("memo", "revision memo");
    itemrev.setFieldValu("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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Nexus

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Payroll Item

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 Using 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.

```javascript
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.

```javascript
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.

```javascript
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.

```javascript
function deletePayrollItem()
{
    var savedSearchInternalId = 2
    var searchresults = nlapiSearchRecord('payrollitem', savedSearchInternalId, null, null);
    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**

The internal ID for this record is `pricelevel`. Search is not available on this record type.

See the [SuiteScript Records Browser](https://app.netsuite.com/app/help) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

**Revenue Recognition Schedule**

The internal ID for this record is `revrecschedule`.

See the [SuiteScript Records Browser](https://app.netsuite.com/app/help) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see Using 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

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 Using 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.

Sales Tax Item

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Subsidiary

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Tax Group

The internal ID for this record is taxgroup.

See the SuiteScript Records Browser for all internal IDs associated with this record.

Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Tax Period

The internal ID for this record is taxperiod. The SuiteScript Records Browser will soon be updated to include the internal field IDs for this record.
Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

**Tax Type**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Term**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Unit of Measure**

The internal ID for this record is `unitstype`. The SuiteScript Records Browser will soon be updated to include the internal field IDs for this record.

Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

**Vendor Category**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.
Customization

The following customization records are scriptable in SuiteScript:

- Custom List
- Custom Record

Custom List

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 Setup > Customization > Lists. If you have the Show Internal IDs preference enabled, all list internal IDs will be appear in the ID column.

- Search Filters
- Search Columns

Search Filters

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Field Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>internalid</td>
<td>Internal ID</td>
<td>select</td>
</tr>
<tr>
<td>internalidnumber</td>
<td>Internal ID (Number)</td>
<td>integer</td>
</tr>
<tr>
<td>isinactive</td>
<td>Inactive</td>
<td>checkbox</td>
</tr>
<tr>
<td>name</td>
<td>Name</td>
<td>text</td>
</tr>
</tbody>
</table>

Search Columns

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Field Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>internalid</td>
<td>Internal ID</td>
<td>select</td>
</tr>
<tr>
<td>isinactive</td>
<td>Inactive</td>
<td>checkbox</td>
</tr>
<tr>
<td>name</td>
<td>Name</td>
<td>text</td>
</tr>
</tbody>
</table>

Custom Record

Each custom record will have a unique internal ID. For example, a custom record's internal ID might be customrecord22, or customrecord5, or customrecord_holidaySurvey, depending on
whether you have accepted the default ID assigned to the custom record or you have created
your own ID.

See the SuiteScript Records Browser for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript
Records Browser in the NetSuite Help Center.

To see a list of IDs for all your custom records, in the UI, go to Setup > Customization > Record
Types. If you have the Show Internal IDs preference enabled, all record internal IDs will be
listed in the ID column.

**Usage Notes**

When searching on an auto-generated record ID in a custom record, you will need to filter
using the *idtext* identifier. Here is an example using SuiteScript. Note that this also applies to
searches executed in the NetSuite user interface.

**Example**

**Before**

```javascript
var filters = new Array();
filters[0] = new nlobjSearchFilter("name", null, "is", 'CRA0001', null);

var results = nlapiSearchRecord("customrecord13", null, filters, null);
results;
```

**After**

```javascript
var filters = new Array();
filters[0] = new nlobjSearchFilter("idtext", null, "is", 'CRA0001', null);

var results = nlapiSearchRecord("customrecord13", null, filters, null);
results;
```
Marketing

The following marketing records are scriptable in SuiteScript:

- Campaign
- Coupon Code
- Promotion

**Campaign**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Coupon Code**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Promotion**

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

**Usage Notes**

The following table provides usage notes for specific fields on this record.
<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body Fields</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discounttype</td>
<td>radio buttons</td>
<td>Valid values in scripts are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• percent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• flat</td>
</tr>
</tbody>
</table>
Chapter 71 Scriptable Sublists

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.

<table>
<thead>
<tr>
<th>Sublist</th>
<th>Sublist Internal ID</th>
<th>Sublist Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Sublist (contact roles)</td>
<td>contactroles</td>
<td>list</td>
</tr>
<tr>
<td>Address Sublist</td>
<td>addressbook</td>
<td>editor</td>
</tr>
<tr>
<td>Adjustments Sublist</td>
<td>inventory</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Apply Sublist</td>
<td>apply</td>
<td>list</td>
</tr>
<tr>
<td>Assignees Sublist</td>
<td>assignee</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Attendees Sublist</td>
<td>attendee</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Billable Expenses Sublist</td>
<td>expcost</td>
<td>list</td>
</tr>
<tr>
<td>Billable Items Sublist</td>
<td>itemcost</td>
<td>list</td>
</tr>
<tr>
<td>Billable Time Sublist</td>
<td>time</td>
<td>list</td>
</tr>
<tr>
<td>Bin Numbers Sublist</td>
<td>binnumber</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Sublist</td>
<td>Sublist Internal ID</td>
<td>Sublist Type</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Company Contributions Sublist</td>
<td>companycontribution</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Company Contributions Sublist</td>
<td>companytax</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Competitors Sublist</td>
<td>competitors</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Credits Sublist</td>
<td>credit</td>
<td>list</td>
</tr>
<tr>
<td>Currencies Sublist</td>
<td>currency</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Custom Child Record Sublists</td>
<td>recmachcustrecord</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Deductions Sublist</td>
<td>deduction</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Demand Plan Detail Sublist</td>
<td>demandplandetail</td>
<td>list</td>
</tr>
<tr>
<td>Deposits Sublist</td>
<td>deposit</td>
<td>list</td>
</tr>
<tr>
<td>Direct Mail Sublist</td>
<td>campaigndirectmail</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Download Sublist</td>
<td>download</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Earnings Sublist</td>
<td>earning</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>E-mail Sublist</td>
<td>campaignemail</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Employee Taxes Sublist</td>
<td>employeetax</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Escalate To Sublist</td>
<td>escalateto</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Expenses Sublist</td>
<td>expense</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Group Pricing Sublist</td>
<td>grouppricing</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Item Fulfillment/Receipt Sublist</td>
<td>item</td>
<td>list</td>
</tr>
<tr>
<td>Items Sublist</td>
<td>item</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Item Pricing Sublist</td>
<td>itempricing</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Lead Nurturing Sublist</td>
<td>campaigndrip</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Line Sublist</td>
<td>line</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Members Sublist</td>
<td>member</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Orders Sublist</td>
<td>order</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Other Events Sublist</td>
<td>campaignevent</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Partners Sublist</td>
<td>partners</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Pricing Sublist</td>
<td>price</td>
<td>list</td>
</tr>
<tr>
<td>Predecessors Sublist</td>
<td>predecessor</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Related Solutions Sublist</td>
<td>solutions</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Resources Sublist</td>
<td>resource</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Sales Team Sublist</td>
<td>salesteam</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Shipping Sublist</td>
<td>shipgroup</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Site Category</td>
<td>sitecategory</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Time Tracking Sublist</td>
<td>timeitem</td>
<td>inlineeditor</td>
</tr>
<tr>
<td>Topics Sublist</td>
<td>topics</td>
<td>inlineeditor</td>
</tr>
</tbody>
</table>
Access Sublist (contact roles)

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.)

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>contact</td>
<td>Contact</td>
<td>select</td>
<td>false</td>
</tr>
<tr>
<td>email</td>
<td>Email</td>
<td>email</td>
<td>false</td>
</tr>
<tr>
<td>giveaccess</td>
<td>Access</td>
<td>checkbox</td>
<td>false</td>
</tr>
<tr>
<td>passwordconfirm</td>
<td>Confirm Password</td>
<td>password</td>
<td>false</td>
</tr>
<tr>
<td>role</td>
<td>Role</td>
<td>select</td>
<td>false</td>
</tr>
<tr>
<td>sendemail</td>
<td>Notify</td>
<td>checkbox</td>
<td>false</td>
</tr>
</tbody>
</table>

Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Address Sublist

The internal ID for this sublist is addressbook. It is an editor sublist. (In the NetSuite Help Center, see Editor Sublists for information on this sublist type.)

The Address sublist appears on the following records: Contact, Customer, Partner, Vendor, Employee. To see the internal IDs associated with the Address sublist, open the SuiteScript Records Browser and click on one of the records that includes this sublist.

Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Adjustments Sublist

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, open the SuiteScript Records Browser and click on the Inventory Adjustment record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

### Apply Sublist

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 click on one of the records that includes this sublist.

**Note:** For information on using the SuiteScript Records Browser, see Using 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

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, open the SuiteScript Records Browser and click on the Project Task record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

### Attendees Sublist

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, open the SuiteScript Records Browser and click on the Event record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.
Billable Expenses Sublist

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 click on one of the records that includes this sublist.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Billable Items Sublist

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 click on one of the records that includes this sublist.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Billable Time Sublist

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 click on one of the records that includes this sublist.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Bin Numbers Sublist

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, open the SuiteScript Records Browser and click on the Inventory Item record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.
Company Contributions Sublist

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. To see the internal IDs associated with the Company Contributions sublist, open the SuiteScript Records Browser and click on the Paycheck Journal record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Company Taxes Sublist

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, open the SuiteScript Records Browser and click on the Paycheck Journal record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Competitors Sublist

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, open the SuiteScript Records Browser and click on the Opportunity record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Credits Sublist

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 SuiteScript Records Browser. To get the internal IDs for the Credits sublist, click the Customer Payments record in the SuiteScript Records Browser.)

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.
Currencies Sublist

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 click on any of these records.

Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Custom Child Record Sublists

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.
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 Understanding Parent - Child Record Relationships in the SuiteBuilder Guide.)

The Fixed Assets fields that will appear in the sublist are the mandatory fields (those 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.
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.)
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.
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.
   a. Add a custom field to your new custom record type.
   b. On the field definition page for the new field, set the field Type to List/Record (see figure below).
   c. 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.
   d. Select the Record is Parent checkbox. 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.

![Edit Fixed Assets Field](image)

If you choose, you can have your custom child record sublist appear on a standard or custom subtab of the parent record.

4. 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.
5. 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**.

![Fixed Assets sublist](image)

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.
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`. 
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).
Click to create a new (child) Fixed Assets record.
2. 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`.

3. 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 Setup > Customization > Record Types > 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.
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);
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.

```javascript
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', 'Fixed asset 3');

//Set the value for the Cost field
rec.setCurrentLineItemValue('recmachcustrecord102', 'custrecord1', 4000);

//Set the value for the Useful Life in Years field
rec.setCurrentLineItemValue('recmachcustrecord102', 'custrecord4', '2');

//Commit your sublist changes
rec.commitLineItem('recmachcustrecord102');

//Submit the updated Customer record
var id = nlapiSubmitRecord(rec, true);

---

**Deductions Sublist**

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. To see the internal IDs associated with the Deductions sublist, open the **SuiteScript Records Browser** and click on the Paycheck Journal record.

**Note:** For information on using the **SuiteScript Records Browser**, see [Using the SuiteScript Records Browser](#) in the NetSuite Help Center.

**Demand Plan Detail Sublist**

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, **demandplancalendarType**, 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 Working with Records in Dynamic Mode.

Be aware of the following requirements:

- In order 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.

- 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:

```javascript
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', '1/1/2011');
record.setFieldValue('enddate', '12/31/2011');
```
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.

- 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:

```javascript
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', '1/1/2011');
record.setFieldValue('enddate', '12/31/2011');

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.

- 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

```javascript
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

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 click on one of the records that includes this sublist.
Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Direct Mail Sublist

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, open the SuiteScript Records Browser and click on the Campaign record.

Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Download Sublist

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, open the SuiteScript Records Browser and click on the Customer record.

Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Earnings Sublist

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. To see the internal IDs associated with the Earning sublist, open the SuiteScript Records Browser and click on the Paycheck Journal record.

Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

E-mail Sublist

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, open the SuiteScript Records Browser and click on the Campaign record.

Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.
Employee Taxes Sublist

The internal ID for this sublist is `employetax`. 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, open the SuiteScript Records Browser and click on the Paycheck Journal record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Escalate To Sublist

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, open the SuiteScript Records Browser and click on the Case record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Expenses Sublist

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 click on one of the records that includes this sublist.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Group Pricing Sublist

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, open the SuiteScript Records Browser and click on the Customer record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.
**Item Fulfillment/Receipt Sublist**

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 click on one of the records that includes this sublist.

**Note:** For information on using the SuiteScript Records Browser, see Using 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**

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 click on one of the records that includes this sublist.

**Note:** For information on using the SuiteScript Records Browser, see Using 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.

<table>
<thead>
<tr>
<th>Sublist Field Internal ID</th>
<th>Sublist Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>altsalesamt</td>
<td>Alt. Sales Amount</td>
<td>This sublist field is NOT scriptable on the Cash Sale and Invoice records.</td>
</tr>
<tr>
<td>costestimatetype</td>
<td>Cost Estimate Type</td>
<td>This field supports both client and server scripting. It is also available in search.</td>
</tr>
<tr>
<td>costestimate</td>
<td>Est. Extended Cost</td>
<td>This field supports both client and server scripting. It is also available in search.</td>
</tr>
<tr>
<td>estgrossprofit</td>
<td>Est. Gross Profit</td>
<td>This field cannot be scripted with client/server SuiteScript. It is, however, available in search.</td>
</tr>
<tr>
<td>estgrossprofitpercent</td>
<td>Est. Gross Profit Percent</td>
<td>This field cannot be scripted with client/server SuiteScript. It is, however, available in search.</td>
</tr>
</tbody>
</table>

**Item Pricing Sublist**

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, open the SuiteScript Records Browser and click on the customer record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.
Lead Nurturing Sublist

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, open the SuiteScript Records Browser and click on the Campaign record.

Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Line Sublist

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 click on one of the records that includes this sublist.

Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Members Sublist

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 click on one of the records that includes this sublist.

Note: For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Usage Notes

The following table provides usage notes for specific sublist fields on this sublist.

<table>
<thead>
<tr>
<th>Sublist Field Internal ID</th>
<th>Sublist Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>taxschedule</td>
<td>Scheduled</td>
<td>This sublist field is visible only in the UI when the Advanced Taxes feature is enabled.</td>
</tr>
</tbody>
</table>

Orders Sublist

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, open the SuiteScript Records Browser and click on the Item Supply Plan record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

### Other Events Sublist

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, open the SuiteScript Records Browser and click on the Campaign record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

### Partners Sublist

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 click on one of the records that includes this sublist.

**Note:** For information on using the SuiteScript Records Browser, see Using 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

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

To see which records the Pricing sublist appears on, see Records that Include the Pricing Sublist.

**Note:** For general information on item pricing, see these topics in the NetSuite Help Center:

- Item Pricing
- Setting Up Item Pricing
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.

```
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).

**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 is 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/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.

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 checkbox 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:

```javascript
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.

By comparison, this figure shows the Pricing sublist with the Multiple Prices feature disabled. You can set only one price for the item.

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 checkbox 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:

```javascript
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.

By comparison, this figure shows the Pricing sublist with the Quantity Pricing feature disabled. Item prices are not determined by the quantities specified.

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** checkbox 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:

```javascript
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).

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.

Based on the internal ID in the figure above, you will set the type parameter in APIs such as `nlapiGetMatrixField` as follows:

```javascript
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.

<table>
<thead>
<tr>
<th>Field UI Label</th>
<th>Field Internal ID</th>
<th>Field Type</th>
<th>Mandatory</th>
<th>Field Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Maxtrix Fields</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base Price</td>
<td>price</td>
<td>string</td>
<td>true</td>
<td>The price for that level and quantity. See Figure 1 - Matrix Fields. Important: When using matrix APIs (ie., any API that has the word Matrix in its name), <code>price</code> will generally be the value specified for the <code>fldnam</code> parameter. The exception to this is if the Multiple Currencies feature is enabled in your account.</td>
</tr>
<tr>
<td>Qty</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sublist Fields</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default Discount %</td>
<td>discount</td>
<td></td>
<td></td>
<td>See Figure 2 - Sublist Fields. See also Standard Sublist APIs for a code sample that references this internal ID.</td>
</tr>
<tr>
<td></td>
<td>currency</td>
<td></td>
<td></td>
<td>The currency field is a hidden field. It is not visible in the UI. See Figure 2 - Sublist Fields. This field is only scriptable when the Multiple Currencies feature is enabled. The internal IDs for the <code>fldname</code> parameter in matrix APIs will be <code>price1currency</code>, <code>price2currency</code>, <code>pricecurrency3</code>, etc., to reflect the currency internal ID.</td>
</tr>
<tr>
<td>Price Level</td>
<td>pricelevel</td>
<td></td>
<td></td>
<td>The pricelevel for this price. See Figure 2 - Sublist Fields. See also Standard Sublist APIs for a code sample that references this internal ID.</td>
</tr>
<tr>
<td>Field UI Label</td>
<td>Field Internal ID</td>
<td>Field Type</td>
<td>Mandatory</td>
<td>Field Notes</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------</td>
<td>------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Quantity Pricing Schedule</td>
<td>quantitypricingschedule</td>
<td>select</td>
<td>false</td>
<td>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 Figure 3 - Body Fields. Also note that the value of the quantity pricing schedule sets the value of the Calculate Quantity Discounts (overallquantitypricingtype) field. Use Field APIs to access this field.</td>
</tr>
<tr>
<td>Calculate Quantity Discounts</td>
<td>overallquantitypricingtype</td>
<td>select</td>
<td>false</td>
<td>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 Figure 3 - Body Fields. Use Field APIs to access this field.</td>
</tr>
<tr>
<td>Use Marginal Rates</td>
<td>usemarginalrates</td>
<td>checkbox</td>
<td>false</td>
<td>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 Figure 3 - Body Fields. Use Field APIs to access this field.</td>
</tr>
<tr>
<td>Pricing Group</td>
<td>pricinggroup</td>
<td>select</td>
<td>false</td>
<td>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 Figure 3 - Body Fields. Use Field APIs to access this field.</td>
</tr>
</tbody>
</table>
Figure 1 - Matrix Fields

Pricing sublist matrix field.

Figure 2 - Sublist Fields

Pricing sublist sublist fields. Use non-matrix Sublist APIs to get/set values for these fields. Note: The currency field is hidden in the UI but appears in the source code between the Price Level and Default Discount % fields.

Figure 3 - Body Fields

Pricing sublist body fields. Use standard Field APIs to get/set values for these 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.

```javascript
// 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

// 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 = "pricelevel";
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>

    // this Matrix API obtains the value of the Quantity level
    strName += itemRecord.getMatrixValue( priceID, 'price', j);
    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)`

On matrix “header” fields, use: `nlapiGetMatrixField, nlapiGetMatrixValue, nlapiGetMatrixCount`

When on an existing line in the matrix, use: `nlapiGet([Set]CurrentLineItemMatrixValue to get|set the price on that line in a specific column)`

For all other lines in the matrix, use: `nlapiGetLineItemMatrixValue, nlapiGetLineItemMatrixField, nlapiFindLineItemValue`
**Standard Sublist APIs**

If you want to reference the other fields in the Pricing sublist, for example *currency*, *name*, *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:

```javascript
// 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

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, open the SuiteScript Records Browser and click on the Project Task record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Related Solutions Sublist

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, open the SuiteScript Records Browser and click on the Solution record.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Resources Sublist

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 Using the SuiteScript Records Browser in the NetSuite Help Center.

Sales Team Sublist

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 click on one of the records that includes this sublist.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.
Shipping Sublist

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, in the SuiteScript Records Browser, click any of the other record types that support this sublist.)

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Usage Notes

<table>
<thead>
<tr>
<th>Sublist Field Internal ID</th>
<th>Sublist Field UI Label</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>shippingtaxcode</td>
<td>Shipping Tax Code</td>
<td>This sublist field appears only if per-line taxes have been set on the Item sublist.</td>
</tr>
</tbody>
</table>

Site Category

The internal ID for this sublist is `sitecategory`. 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 click on one of the records that includes this sublist.

**Note:** For information on using the SuiteScript Records Browser, see Using the SuiteScript Records Browser in the NetSuite Help Center.

Time Tracking Sublist

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, in the SuiteScript Records Browser, click any of the other record types that support this sublist.)
**Note:** For information on using the SuiteScript Records Browser, see *Using the SuiteScript Records Browser* in the NetSuite Help Center.

### Topics Sublist

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, open the SuiteScript Records Browser and click on the solution record.

**Note:** For information on using the SuiteScript Records Browser, see *Using the SuiteScript Records Browser* in the NetSuite Help Center.

### Units

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, open the SuiteScript Records Browser and click the Unit of Measure record.

**Note:** For information on using the SuiteScript Records Browser, see *Using the SuiteScript Records Browser* in the NetSuite Help Center.

### Vendors

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 click on either of these records.

**Note:** For information on using the SuiteScript Records Browser, see *Using the SuiteScript Records Browser* in the NetSuite Help Center.
Chapter 72 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)`
- `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.

<table>
<thead>
<tr>
<th>Record</th>
<th>Initialization Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>All SuiteScript-supported records.</td>
<td>recordmode</td>
<td>dynamic</td>
</tr>
<tr>
<td>For a list of records, see SuiteScript Supported Records.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>For information on scripting a record in dynamic mode, see Working with Records in Dynamic Mode.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All SuiteScript-supported records that support form customization.</td>
<td>customform</td>
<td>&lt;customformid&gt;</td>
</tr>
<tr>
<td>Assembly Build</td>
<td>assemblyitem</td>
<td>&lt;assemblyitemid&gt;</td>
</tr>
<tr>
<td>Cash Refund</td>
<td>entity</td>
<td>&lt;entityid&gt;</td>
</tr>
<tr>
<td>Cash Sale</td>
<td>entity</td>
<td>&lt;entityid&gt;</td>
</tr>
<tr>
<td>Check</td>
<td>entity</td>
<td>&lt;entityid&gt;</td>
</tr>
<tr>
<td>Credit Memo</td>
<td>entity</td>
<td>&lt;entityid&gt;</td>
</tr>
<tr>
<td>Customer Payment</td>
<td>entity</td>
<td>&lt;entityid&gt;</td>
</tr>
<tr>
<td>Customer Refund</td>
<td>entity</td>
<td>&lt;entityid&gt;</td>
</tr>
<tr>
<td>Estimate</td>
<td>entity</td>
<td>&lt;entityid&gt;</td>
</tr>
<tr>
<td>Expense Report</td>
<td>entity</td>
<td>&lt;entityid&gt;</td>
</tr>
</tbody>
</table>
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**

// 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();

---

**Record Initialization Defaults**

Record | Initialization Type | Values
---|---|---
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>
Service | subtype | sale | resale | purchase
Tax Group | nexuscountry | <countrycode>
Tax Type | country | <countrycode>
Topic | parenttopic | <parenttopicid>
Vendor Bill | entity | <entityid>
Vendor Payment | entity | <entityid>
Work Order | assemblyitem | <assemblyitemid>

See Country Codes Used for Initialization Parameters.
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);

**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'});
```

<table>
<thead>
<tr>
<th>Country Code</th>
<th>Country Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD</td>
<td>Andorra</td>
</tr>
<tr>
<td>AE</td>
<td>United Arab Emirates</td>
</tr>
<tr>
<td>AF</td>
<td>Afghanistan</td>
</tr>
<tr>
<td>AG</td>
<td>Antigua and Barbuda</td>
</tr>
<tr>
<td>AI</td>
<td>Anguilla</td>
</tr>
<tr>
<td>AL</td>
<td>Albania</td>
</tr>
<tr>
<td>AM</td>
<td>Armenia</td>
</tr>
<tr>
<td>AN</td>
<td>Netherlands Antilles</td>
</tr>
<tr>
<td>AO</td>
<td>Angola</td>
</tr>
<tr>
<td>AQ</td>
<td>Antarctica</td>
</tr>
<tr>
<td>AR</td>
<td>Argentina</td>
</tr>
<tr>
<td>AS</td>
<td>American Samoa</td>
</tr>
<tr>
<td>AT</td>
<td>Austria</td>
</tr>
<tr>
<td>AU</td>
<td>Australia</td>
</tr>
<tr>
<td>AW</td>
<td>Aruba</td>
</tr>
<tr>
<td>AZ</td>
<td>Azerbaijan</td>
</tr>
<tr>
<td>BA</td>
<td>Bosnia and Herzegovina</td>
</tr>
<tr>
<td>BB</td>
<td>Barbados</td>
</tr>
<tr>
<td>BD</td>
<td>Bangladesh</td>
</tr>
<tr>
<td>BE</td>
<td>Belgium</td>
</tr>
<tr>
<td>BF</td>
<td>Burkina Faso</td>
</tr>
<tr>
<td>BG</td>
<td>Bulgaria</td>
</tr>
<tr>
<td>BH</td>
<td>Bahrain</td>
</tr>
<tr>
<td>BI</td>
<td>Burundi</td>
</tr>
<tr>
<td>Country Code</td>
<td>Country Name</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>BJ</td>
<td>Benin</td>
</tr>
<tr>
<td>BL</td>
<td>Saint Barthélemy</td>
</tr>
<tr>
<td>BM</td>
<td>Bermuda</td>
</tr>
<tr>
<td>BN</td>
<td>Brunei Darussalam</td>
</tr>
<tr>
<td>BO</td>
<td>Bolivia</td>
</tr>
<tr>
<td>BR</td>
<td>Brazil</td>
</tr>
<tr>
<td>BS</td>
<td>Bahamas</td>
</tr>
<tr>
<td>BT</td>
<td>Bhutan</td>
</tr>
<tr>
<td>BV</td>
<td>Bouvet Island</td>
</tr>
<tr>
<td>BW</td>
<td>Botswana</td>
</tr>
<tr>
<td>BY</td>
<td>Belarus</td>
</tr>
<tr>
<td>BZ</td>
<td>Belize</td>
</tr>
<tr>
<td>CA</td>
<td>Canada</td>
</tr>
<tr>
<td>CC</td>
<td>Cocos (Keeling) Islands</td>
</tr>
<tr>
<td>CD</td>
<td>Congo, Democratic People's Republic</td>
</tr>
<tr>
<td>CF</td>
<td>Central African Republic</td>
</tr>
<tr>
<td>CG</td>
<td>Congo, Republic of</td>
</tr>
<tr>
<td>CH</td>
<td>Switzerland</td>
</tr>
<tr>
<td>CI</td>
<td>Cote d'Ivoire</td>
</tr>
<tr>
<td>CK</td>
<td>Cook Islands</td>
</tr>
<tr>
<td>CL</td>
<td>Chile</td>
</tr>
<tr>
<td>CM</td>
<td>Cameroon</td>
</tr>
<tr>
<td>CN</td>
<td>China</td>
</tr>
<tr>
<td>CO</td>
<td>Colombia</td>
</tr>
<tr>
<td>CR</td>
<td>Costa Rica</td>
</tr>
<tr>
<td>CU</td>
<td>Cuba</td>
</tr>
<tr>
<td>CV</td>
<td>Cap Verde</td>
</tr>
<tr>
<td>CX</td>
<td>Christmas Island</td>
</tr>
<tr>
<td>CY</td>
<td>Cyprus</td>
</tr>
<tr>
<td>CZ</td>
<td>Czech Republic</td>
</tr>
<tr>
<td>DE</td>
<td>Germany</td>
</tr>
<tr>
<td>DJ</td>
<td>Djibouti</td>
</tr>
<tr>
<td>DK</td>
<td>Denmark</td>
</tr>
<tr>
<td>DM</td>
<td>Dominica</td>
</tr>
<tr>
<td>DO</td>
<td>Dominican Republic</td>
</tr>
<tr>
<td>DZ</td>
<td>Algeria</td>
</tr>
<tr>
<td>Country Code</td>
<td>Country Name</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>EC</td>
<td>Ecuador</td>
</tr>
<tr>
<td>EE</td>
<td>Estonia</td>
</tr>
<tr>
<td>EG</td>
<td>Egypt</td>
</tr>
<tr>
<td>EH</td>
<td>Western Sahara</td>
</tr>
<tr>
<td>ER</td>
<td>Eritrea</td>
</tr>
<tr>
<td>ES</td>
<td>Spain</td>
</tr>
<tr>
<td>ET</td>
<td>Ethiopia</td>
</tr>
<tr>
<td>FI</td>
<td>Finland</td>
</tr>
<tr>
<td>FJ</td>
<td>Fiji</td>
</tr>
<tr>
<td>FK</td>
<td>Falkland Islands (Malvina)</td>
</tr>
<tr>
<td>FM</td>
<td>Micronesia, Federal State of</td>
</tr>
<tr>
<td>FO</td>
<td>Faroe Islands</td>
</tr>
<tr>
<td>FR</td>
<td>France</td>
</tr>
<tr>
<td>GA</td>
<td>Gabon</td>
</tr>
<tr>
<td>GB</td>
<td>United Kingdom (GB)</td>
</tr>
<tr>
<td>GD</td>
<td>Grenada</td>
</tr>
<tr>
<td>GE</td>
<td>Georgia</td>
</tr>
<tr>
<td>GF</td>
<td>French Guiana</td>
</tr>
<tr>
<td>GG</td>
<td>Guernsey</td>
</tr>
<tr>
<td>GH</td>
<td>Ghana</td>
</tr>
<tr>
<td>GI</td>
<td>Gibraltar</td>
</tr>
<tr>
<td>GL</td>
<td>Greenland</td>
</tr>
<tr>
<td>GM</td>
<td>Gambia</td>
</tr>
<tr>
<td>GN</td>
<td>Guinea</td>
</tr>
<tr>
<td>GP</td>
<td>Guadeloupe</td>
</tr>
<tr>
<td>GQ</td>
<td>Equatorial Guinea</td>
</tr>
<tr>
<td>GR</td>
<td>Greece</td>
</tr>
<tr>
<td>GS</td>
<td>South Georgia</td>
</tr>
<tr>
<td>GT</td>
<td>Guatemala</td>
</tr>
<tr>
<td>GU</td>
<td>Guam</td>
</tr>
<tr>
<td>GW</td>
<td>Guinea-Bissau</td>
</tr>
<tr>
<td>GY</td>
<td>Guyana</td>
</tr>
<tr>
<td>HK</td>
<td>Hong Kong</td>
</tr>
<tr>
<td>HM</td>
<td>Heard and McDonald Islands</td>
</tr>
<tr>
<td>HN</td>
<td>Honduras</td>
</tr>
<tr>
<td>HR</td>
<td>Croatia/Hrvatska</td>
</tr>
<tr>
<td>Country Code</td>
<td>Country Name</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>HT</td>
<td>Haiti</td>
</tr>
<tr>
<td>HU</td>
<td>Hungary</td>
</tr>
<tr>
<td>ID</td>
<td>Indonesia</td>
</tr>
<tr>
<td>IE</td>
<td>Ireland</td>
</tr>
<tr>
<td>IL</td>
<td>Israel</td>
</tr>
<tr>
<td>IM</td>
<td>Isle of Man</td>
</tr>
<tr>
<td>IN</td>
<td>India</td>
</tr>
<tr>
<td>IO</td>
<td>British Indian Ocean Territory</td>
</tr>
<tr>
<td>IQ</td>
<td>Iraq</td>
</tr>
<tr>
<td>IR</td>
<td>Iran (Islamic Republic of)</td>
</tr>
<tr>
<td>IS</td>
<td>Iceland</td>
</tr>
<tr>
<td>IT</td>
<td>Italy</td>
</tr>
<tr>
<td>JE</td>
<td>Jersey</td>
</tr>
<tr>
<td>JM</td>
<td>Jamaica</td>
</tr>
<tr>
<td>JO</td>
<td>Jordan</td>
</tr>
<tr>
<td>JP</td>
<td>Japan</td>
</tr>
<tr>
<td>KE</td>
<td>Kenya</td>
</tr>
<tr>
<td>KG</td>
<td>Kyrgyzstan</td>
</tr>
<tr>
<td>KH</td>
<td>Cambodia</td>
</tr>
<tr>
<td>KI</td>
<td>Kiribati</td>
</tr>
<tr>
<td>KM</td>
<td>Comoros</td>
</tr>
<tr>
<td>KN</td>
<td>Saint Kitts and Nevis</td>
</tr>
<tr>
<td>KP</td>
<td>Korea, Democratic People's Republic</td>
</tr>
<tr>
<td>KR</td>
<td>Korea, Republic of</td>
</tr>
<tr>
<td>KW</td>
<td>Kuwait</td>
</tr>
<tr>
<td>KY</td>
<td>Cayman Islands</td>
</tr>
<tr>
<td>KZ</td>
<td>Kazakhstan</td>
</tr>
<tr>
<td>LA</td>
<td>Lao People’s Democratic Republic</td>
</tr>
<tr>
<td>LB</td>
<td>Lebanon</td>
</tr>
<tr>
<td>LC</td>
<td>Saint Lucia</td>
</tr>
<tr>
<td>LI</td>
<td>Liechtenstein</td>
</tr>
<tr>
<td>LK</td>
<td>Sri Lanka</td>
</tr>
<tr>
<td>LR</td>
<td>Liberia</td>
</tr>
<tr>
<td>LS</td>
<td>Lesotho</td>
</tr>
<tr>
<td>LT</td>
<td>Lithuania</td>
</tr>
<tr>
<td>LU</td>
<td>Luxembourg</td>
</tr>
<tr>
<td>Country Code</td>
<td>Country Name</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>LV</td>
<td>Latvia</td>
</tr>
<tr>
<td>LY</td>
<td>Libyan Arab Jamahiriya</td>
</tr>
<tr>
<td>MA</td>
<td>Morocco</td>
</tr>
<tr>
<td>MC</td>
<td>Monaco</td>
</tr>
<tr>
<td>MD</td>
<td>Moldova, Republic of</td>
</tr>
<tr>
<td>ME</td>
<td>Montenegro</td>
</tr>
<tr>
<td>MF</td>
<td>Saint Martin</td>
</tr>
<tr>
<td>MG</td>
<td>Madagascar</td>
</tr>
<tr>
<td>MH</td>
<td>Marshall Islands</td>
</tr>
<tr>
<td>MK</td>
<td>Macedonia</td>
</tr>
<tr>
<td>ML</td>
<td>Mali</td>
</tr>
<tr>
<td>MM</td>
<td>Myanmar</td>
</tr>
<tr>
<td>MN</td>
<td>Mongolia</td>
</tr>
<tr>
<td>MO</td>
<td>Macau</td>
</tr>
<tr>
<td>MP</td>
<td>Northern Mariana Islands</td>
</tr>
<tr>
<td>MQ</td>
<td>Martinique</td>
</tr>
<tr>
<td>MR</td>
<td>Mauritania</td>
</tr>
<tr>
<td>MS</td>
<td>Montserrat</td>
</tr>
<tr>
<td>MT</td>
<td>Malta</td>
</tr>
<tr>
<td>MU</td>
<td>Mauritius</td>
</tr>
<tr>
<td>MV</td>
<td>Maldives</td>
</tr>
<tr>
<td>MW</td>
<td>Malawi</td>
</tr>
<tr>
<td>MX</td>
<td>Mexico</td>
</tr>
<tr>
<td>MY</td>
<td>Malaysia</td>
</tr>
<tr>
<td>MZ</td>
<td>Mozambique</td>
</tr>
<tr>
<td>NA</td>
<td>Namibia</td>
</tr>
<tr>
<td>NC</td>
<td>New Caledonia</td>
</tr>
<tr>
<td>NE</td>
<td>Niger</td>
</tr>
<tr>
<td>NF</td>
<td>Norfolk Island</td>
</tr>
<tr>
<td>NG</td>
<td>Nigeria</td>
</tr>
<tr>
<td>NI</td>
<td>Nicaragua</td>
</tr>
<tr>
<td>NL</td>
<td>Netherlands</td>
</tr>
<tr>
<td>NO</td>
<td>Norway</td>
</tr>
<tr>
<td>NP</td>
<td>Nepal</td>
</tr>
<tr>
<td>NR</td>
<td>Nauru</td>
</tr>
<tr>
<td>NU</td>
<td>Niue</td>
</tr>
<tr>
<td>Country Code</td>
<td>Country Name</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>NZ</td>
<td>New Zealand</td>
</tr>
<tr>
<td>OM</td>
<td>Oman</td>
</tr>
<tr>
<td>PA</td>
<td>Panama</td>
</tr>
<tr>
<td>PE</td>
<td>Peru</td>
</tr>
<tr>
<td>PF</td>
<td>French Polynesia</td>
</tr>
<tr>
<td>PG</td>
<td>Papua New Guinea</td>
</tr>
<tr>
<td>PH</td>
<td>Philippines</td>
</tr>
<tr>
<td>PK</td>
<td>Pakistan</td>
</tr>
<tr>
<td>PL</td>
<td>Poland</td>
</tr>
<tr>
<td>PM</td>
<td>St. Pierre and Miquelon</td>
</tr>
<tr>
<td>PN</td>
<td>Pitcairn Island</td>
</tr>
<tr>
<td>PR</td>
<td>Puerto Rico</td>
</tr>
<tr>
<td>PS</td>
<td>Palestinian Territories</td>
</tr>
<tr>
<td>PT</td>
<td>Portugal</td>
</tr>
<tr>
<td>PW</td>
<td>Palau</td>
</tr>
<tr>
<td>PY</td>
<td>Paraguay</td>
</tr>
<tr>
<td>QA</td>
<td>Qatar</td>
</tr>
<tr>
<td>RE</td>
<td>Reunion Island</td>
</tr>
<tr>
<td>RO</td>
<td>Romania</td>
</tr>
<tr>
<td>RS</td>
<td>Serbia</td>
</tr>
<tr>
<td>RU</td>
<td>Russian Federation</td>
</tr>
<tr>
<td>RW</td>
<td>Rwanda</td>
</tr>
<tr>
<td>SA</td>
<td>Saudi Arabia</td>
</tr>
<tr>
<td>SB</td>
<td>Solomon Islands</td>
</tr>
<tr>
<td>SC</td>
<td>Seychelles</td>
</tr>
<tr>
<td>SD</td>
<td>Sudan</td>
</tr>
<tr>
<td>SE</td>
<td>Sweden</td>
</tr>
<tr>
<td>SG</td>
<td>Singapore</td>
</tr>
<tr>
<td>SH</td>
<td>St. Helena</td>
</tr>
<tr>
<td>SI</td>
<td>Slovenia</td>
</tr>
<tr>
<td>SJ</td>
<td>Svalbard and Jan Mayen Islands</td>
</tr>
<tr>
<td>SK</td>
<td>Slovak Republic</td>
</tr>
<tr>
<td>SL</td>
<td>Sierra Leone</td>
</tr>
<tr>
<td>SM</td>
<td>San Marino</td>
</tr>
<tr>
<td>SN</td>
<td>Senegal</td>
</tr>
<tr>
<td>SO</td>
<td>Somalia</td>
</tr>
<tr>
<td>Country Code</td>
<td>Country Name</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>SR</td>
<td>Suriname</td>
</tr>
<tr>
<td>ST</td>
<td>Sao Tome and Principe</td>
</tr>
<tr>
<td>SV</td>
<td>El Salvador</td>
</tr>
<tr>
<td>SY</td>
<td>Syrian Arab Republic</td>
</tr>
<tr>
<td>SZ</td>
<td>Swaziland</td>
</tr>
<tr>
<td>TC</td>
<td>Turks and Caicos Islands</td>
</tr>
<tr>
<td>TD</td>
<td>Chad</td>
</tr>
<tr>
<td>TF</td>
<td>French Southern Territories</td>
</tr>
<tr>
<td>TG</td>
<td>Togo</td>
</tr>
<tr>
<td>TH</td>
<td>Thailand</td>
</tr>
<tr>
<td>TJ</td>
<td>Tajikistan</td>
</tr>
<tr>
<td>TK</td>
<td>Tokelau</td>
</tr>
<tr>
<td>TM</td>
<td>Turkmenistan</td>
</tr>
<tr>
<td>TN</td>
<td>Tunisia</td>
</tr>
<tr>
<td>TO</td>
<td>Tonga</td>
</tr>
<tr>
<td>TP</td>
<td>East Timor</td>
</tr>
<tr>
<td>TR</td>
<td>Turkey</td>
</tr>
<tr>
<td>TT</td>
<td>Trinidad and Tobago</td>
</tr>
<tr>
<td>TV</td>
<td>Tuvalu</td>
</tr>
<tr>
<td>TW</td>
<td>Taiwan</td>
</tr>
<tr>
<td>TZ</td>
<td>Tanzania</td>
</tr>
<tr>
<td>UA</td>
<td>Ukraine</td>
</tr>
<tr>
<td>UG</td>
<td>Uganda</td>
</tr>
<tr>
<td>UM</td>
<td>US Minor Outlying Islands</td>
</tr>
<tr>
<td>US</td>
<td>United States</td>
</tr>
<tr>
<td>UY</td>
<td>Uruguay</td>
</tr>
<tr>
<td>UZ</td>
<td>Uzbekistan</td>
</tr>
<tr>
<td>VA</td>
<td>Holy See (City Vatican State)</td>
</tr>
<tr>
<td>VC</td>
<td>Saint Vincent and the Grenadines</td>
</tr>
<tr>
<td>VE</td>
<td>Venezuela</td>
</tr>
<tr>
<td>VG</td>
<td>Virgin Islands (British)</td>
</tr>
<tr>
<td>VI</td>
<td>Virgin Islands (USA)</td>
</tr>
<tr>
<td>VN</td>
<td>Vietnam</td>
</tr>
<tr>
<td>VU</td>
<td>Vanuatu</td>
</tr>
<tr>
<td>WF</td>
<td>Wallis and Futuna Islands</td>
</tr>
<tr>
<td>WS</td>
<td>Western Samoa</td>
</tr>
<tr>
<td>Country Code</td>
<td>Country Name</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>YE</td>
<td>Yemen</td>
</tr>
<tr>
<td>YT</td>
<td>Mayotte</td>
</tr>
<tr>
<td>ZA</td>
<td>South Africa</td>
</tr>
<tr>
<td>ZM</td>
<td>Zambia</td>
</tr>
<tr>
<td>ZW</td>
<td>Zimbabwe</td>
</tr>
</tbody>
</table>
Chapter 73 Transaction Type IDs

When calling `nlapiSearchRecord(type, id, filters, columns)` for transactions, you can pass the 'type' as a transaction search column filter. The following table lists all NetSuite transaction types and their corresponding transaction type IDs. When scripting, use the **Type ID** to filter the transaction type.

Note that the transaction type IDs are case-sensitive. Be sure to type the IDs as they appear in this table.

<table>
<thead>
<tr>
<th>Transaction Type</th>
<th>Transaction Type ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Build</td>
<td>Build</td>
</tr>
<tr>
<td>Assembly Unbuild</td>
<td>Unbuild</td>
</tr>
<tr>
<td>Bill</td>
<td>VendBill</td>
</tr>
<tr>
<td>Bill CCard</td>
<td>VendCard</td>
</tr>
<tr>
<td>Bill Credit</td>
<td>VendCred</td>
</tr>
<tr>
<td>Bill Payment</td>
<td>VendPymt</td>
</tr>
<tr>
<td>Bin Putaway Worksheet</td>
<td>BinWksht</td>
</tr>
<tr>
<td>Bin Transfer</td>
<td>BinTrnfr</td>
</tr>
<tr>
<td>CCard Refund</td>
<td>CardRfnd</td>
</tr>
<tr>
<td>Cash Refund</td>
<td>CashRfnd</td>
</tr>
<tr>
<td>Cash Sale</td>
<td>CashSale</td>
</tr>
<tr>
<td>Check</td>
<td>Check</td>
</tr>
<tr>
<td>Commission</td>
<td>Commissn</td>
</tr>
<tr>
<td>Credit Card</td>
<td>CardChrg</td>
</tr>
<tr>
<td>Credit Memo</td>
<td>CustCred</td>
</tr>
<tr>
<td>Currency Revaluation</td>
<td>FxReval</td>
</tr>
<tr>
<td>Customer Deposit</td>
<td>CustDep</td>
</tr>
<tr>
<td>Customer Refund</td>
<td>CustRfnd</td>
</tr>
<tr>
<td>Deposit</td>
<td>Deposit</td>
</tr>
<tr>
<td>Deposit Application</td>
<td>DepAppl</td>
</tr>
<tr>
<td>Estimate</td>
<td>Estimate</td>
</tr>
<tr>
<td>Expense Report</td>
<td>ExpRept</td>
</tr>
<tr>
<td>Inventory Adjustment</td>
<td>InvAdjst</td>
</tr>
<tr>
<td>Transaction Type</td>
<td>Transaction Type ID</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Inventory Distribution</td>
<td>InvDistr</td>
</tr>
<tr>
<td>Inventory Transfer</td>
<td>InvTrnfr</td>
</tr>
<tr>
<td>Inventory Worksheet</td>
<td>InvWksht</td>
</tr>
<tr>
<td>Invoice</td>
<td>CustInvc</td>
</tr>
<tr>
<td>Item Fulfillment</td>
<td>ItemShip</td>
</tr>
<tr>
<td>Item Receipt</td>
<td>ItemRcpt</td>
</tr>
<tr>
<td>Journal</td>
<td>Journal</td>
</tr>
<tr>
<td>Liability Adjustment</td>
<td>LiaAdjst</td>
</tr>
<tr>
<td>Opportunity</td>
<td>Opprnty</td>
</tr>
<tr>
<td>Paycheck</td>
<td>Paycheck</td>
</tr>
<tr>
<td>Payment</td>
<td>CustPymt</td>
</tr>
<tr>
<td>Payroll Adjustment</td>
<td>YtdAdjst</td>
</tr>
<tr>
<td>Payroll Liability Check</td>
<td>LiabPymt</td>
</tr>
<tr>
<td>Purchase Order</td>
<td>PurchOrd</td>
</tr>
<tr>
<td>Return Authorization</td>
<td>RtnAuth</td>
</tr>
<tr>
<td>Sales Order</td>
<td>SalesOrd</td>
</tr>
<tr>
<td>Sales Tax Payment</td>
<td>TaxPymt</td>
</tr>
<tr>
<td>Statement Charge</td>
<td>CustChrg</td>
</tr>
<tr>
<td>Tax Liability Cheque</td>
<td>TaxLiab</td>
</tr>
<tr>
<td>Transfer</td>
<td>Transfer</td>
</tr>
<tr>
<td>Vendor Return Authorization</td>
<td>VendAuth</td>
</tr>
<tr>
<td>Work Order</td>
<td>WorkOrd</td>
</tr>
</tbody>
</table>

**Related Topics**
- Transaction Search
- SuiteScript Functions
Chapter 74 Permission Names and IDs

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.

<table>
<thead>
<tr>
<th>Permission ID</th>
<th>Permission Name</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMI_ACCOUNTING</td>
<td>Set Up Accounting</td>
<td>Accounting</td>
</tr>
<tr>
<td>ADMI_ACCTPERIODS</td>
<td>Manage Accounting Periods</td>
<td>Accounting Periods</td>
</tr>
<tr>
<td>ADMI_ACCTSETUP</td>
<td>Set Up Accounting</td>
<td>Accounting</td>
</tr>
<tr>
<td>ADMI_ADVANCED_ANALYTICS</td>
<td>Advanced Analytics</td>
<td>ODBC Connections for Advanced Reporting</td>
</tr>
<tr>
<td>ADMI_BILLPAYSETUP</td>
<td>Set Up Bill Pay</td>
<td>Online Bill Pay</td>
</tr>
<tr>
<td>ADMI_CAMPAIGNEMAIL</td>
<td>Set Up Campaign Email Addresses</td>
<td>Marketing Automation</td>
</tr>
<tr>
<td>ADMI_CAMPAIGNSETUP</td>
<td>Setup Campaigns</td>
<td>Marketing Automation</td>
</tr>
<tr>
<td>ADMI_CASEFORM</td>
<td>Online Case Form</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>ADMI_CASEISSUE</td>
<td>Support Case Issue</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>ADMI_CASEORIGIN</td>
<td>Support Case Origin</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>ADMI_CASEPRIORITY</td>
<td>Support Case Priority</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>ADMI_CASERULE</td>
<td>Support Case Territory Rule</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>ADMI_CASESTATUS</td>
<td>Support Case Status</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>ADMI_CASETERITORY</td>
<td>Support Case Territory</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>ADMI_CASETYPE</td>
<td>Support Case Type</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>ADMI_CLASSESTOLOCS</td>
<td>Convert Classes to Locations</td>
<td>Locations</td>
</tr>
<tr>
<td>ADMI_CLOSEPERIOD</td>
<td>Lock Transactions</td>
<td>Accounting Periods</td>
</tr>
<tr>
<td>ADMI_CONSOLIDATED</td>
<td>Create Consolidation Company</td>
<td>Accounting</td>
</tr>
<tr>
<td>ADMI_CONVERTCLASSES</td>
<td>Convert Classes to Departments</td>
<td>Departments</td>
</tr>
<tr>
<td>ADMI_CREATEPEER</td>
<td>Copy Chart of Accounts to New Company</td>
<td>Accounting</td>
</tr>
<tr>
<td>ADMI_CREDITCARD</td>
<td>Credit Card Processing</td>
<td>Credit Card Payments</td>
</tr>
<tr>
<td>ADMI_CSVIMPORTPREF</td>
<td>Set Up CSV Preferences</td>
<td>Accounting</td>
</tr>
<tr>
<td>ADMI_CUSTCENTER</td>
<td>Custom Centers</td>
<td>Custom Records</td>
</tr>
<tr>
<td>ADMI_CUSTITEMNUMBERFIELD</td>
<td>Custom Item Number Fields</td>
<td>Inventory</td>
</tr>
<tr>
<td>ADMI_CUSTOMERFORM</td>
<td>Online Customer Form</td>
<td>Sales Force Automation</td>
</tr>
<tr>
<td>Permission ID</td>
<td>Permission Name</td>
<td>Feature</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>ADMI_CUSTOMERRULE</td>
<td>Sales Territory Rule</td>
<td>Sales Force Automation</td>
</tr>
<tr>
<td>ADMI_CUSTOMSCRIPT</td>
<td>SuiteScript</td>
<td>Client SuiteScript</td>
</tr>
<tr>
<td>ADMI_CUSTRECORD</td>
<td>Custom Record Types</td>
<td>Custom Records</td>
</tr>
<tr>
<td>ADMI_CUSTSECTION</td>
<td>Custom Center Tabs</td>
<td>Custom Records</td>
</tr>
<tr>
<td>ADMI_CUSTTASKS</td>
<td>Custom Center Links</td>
<td>Custom Records</td>
</tr>
<tr>
<td>ADMI_DUPLICATESETUP</td>
<td>Duplicate Detection Setup</td>
<td>Duplicate Detection &amp; Merge</td>
</tr>
<tr>
<td>ADMIEMPLCATEGORY</td>
<td>Publish Employee List</td>
<td>Intranet</td>
</tr>
<tr>
<td>ADMI_ENTITYSTATUS</td>
<td>Customer Status</td>
<td>Customer Relationship Management</td>
</tr>
<tr>
<td>ADMI_ESCALATIONRULE</td>
<td>Escalation Assignment Rule</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>ADMI_ESCALATIONTERRITORY</td>
<td>Escalation Assignment</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>ADMI_EXPORTIIF</td>
<td>Export as IIF</td>
<td>Accounting</td>
</tr>
<tr>
<td>ADMI_EXPORTIMPORTEBAY</td>
<td>eBay Export/Import</td>
<td>eBay Integration</td>
</tr>
<tr>
<td>ADMI_FINCHARGEPREF</td>
<td>Finance Charge Preferences</td>
<td>A/R</td>
</tr>
<tr>
<td>ADMI_IMPORTQIFFILE</td>
<td>Import Quicken® QIF File</td>
<td>Accounting</td>
</tr>
<tr>
<td>ADMI_ISSUESHOWSTOPPER</td>
<td>Mark Issue As Showstopper</td>
<td>Issue Management</td>
</tr>
<tr>
<td>ADMI_KNOWLEDGEBASE</td>
<td>Publish Knowledge Base</td>
<td>Knowledge Base</td>
</tr>
<tr>
<td>ADMI_KPIREPORT</td>
<td>KPI Scorecards</td>
<td>KPI Scorecards</td>
</tr>
<tr>
<td>ADMI_OFFLINECLIENT</td>
<td>Offline Client</td>
<td>Offline Client for Sales</td>
</tr>
<tr>
<td>ADMI_OUTLOOKINTEGRATION</td>
<td>Outlook Integration</td>
<td>Outlook Integration</td>
</tr>
<tr>
<td>ADMI_PAYROLL</td>
<td>Set Up Payroll</td>
<td>Payroll</td>
</tr>
<tr>
<td>ADMI_PERIODOVERRIDE</td>
<td>Override Period Restrictions</td>
<td>Accounting Periods</td>
</tr>
<tr>
<td>ADMI_REPOGROUPS</td>
<td>Financial Statement Sections</td>
<td>Accounting</td>
</tr>
<tr>
<td>ADMI_REPOLAYOUTS</td>
<td>Financial Statement Layouts</td>
<td>Accounting</td>
</tr>
<tr>
<td>ADMI_SALESTERRITORY</td>
<td>Sales Territory</td>
<td>Sales Force Automation</td>
</tr>
<tr>
<td>ADMI_SETUPEBAY</td>
<td>Set Up eBay</td>
<td>eBay Integration</td>
</tr>
<tr>
<td>ADMI_SFASETUP</td>
<td>Sales Force Automation Setup</td>
<td>Sales Force Automation</td>
</tr>
<tr>
<td>ADMI_STATETAXIMPORT</td>
<td>Import State Sales Tax</td>
<td>Accounting</td>
</tr>
<tr>
<td>ADMI_STORESETUP</td>
<td>Set Up Web Site</td>
<td>Web Site</td>
</tr>
<tr>
<td>ADMI_SUPPORTSETUP</td>
<td>Support Setup</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>ADMI_SWAPPRICES</td>
<td>Swap Prices Between Price Levels</td>
<td>Multiple Prices</td>
</tr>
<tr>
<td>ADMI_SYNCHRONIZATIONSETUP</td>
<td>Set up synchronization</td>
<td>Synchronization</td>
</tr>
<tr>
<td>ADMI_TEAMSELLINGCONTRIBUTION</td>
<td>Team Selling Contribution</td>
<td>Team Selling</td>
</tr>
<tr>
<td>ADMI_TELEPHONY_SETUP</td>
<td>Telephony Integration</td>
<td>Telephony Integration</td>
</tr>
<tr>
<td>ADMI_TRANSLITEMTXT</td>
<td>Translation</td>
<td>Multi-Language</td>
</tr>
<tr>
<td>Permission ID</td>
<td>Permission Name</td>
<td>Feature</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>ADMI_UNCATSITEITEMS</td>
<td>Uncategorized Presentation Items</td>
<td>Web Store</td>
</tr>
<tr>
<td>ADMI_UPSELLSETUP</td>
<td>Upsell Setup</td>
<td>Upsell Manager</td>
</tr>
<tr>
<td>GRAP_AP</td>
<td>Accounts Payable Graphing</td>
<td>A/P</td>
</tr>
<tr>
<td>GRAP_AR</td>
<td>Accounts Receivable Graphing</td>
<td>A/R</td>
</tr>
<tr>
<td>GRAP_EXP</td>
<td>Expenses</td>
<td>Accounting</td>
</tr>
<tr>
<td>GRAP_INC</td>
<td>Income</td>
<td>Accounting</td>
</tr>
<tr>
<td>LIST_NETWORTH</td>
<td>Net Worth</td>
<td>Accounting</td>
</tr>
<tr>
<td>LIST_ACCOUNT</td>
<td>Accounts</td>
<td>Accounting</td>
</tr>
<tr>
<td>LIST_ALLOCSCHEDULE</td>
<td>Allocation Schedules</td>
<td>Expense Allocation</td>
</tr>
<tr>
<td>LIST_AMORTIZATION</td>
<td>Amortization Schedules</td>
<td>Amortization</td>
</tr>
<tr>
<td>LIST_BILLINGSCHEDULE</td>
<td>Billing Schedules</td>
<td>Advanced Billing</td>
</tr>
<tr>
<td>LIST_BIN</td>
<td>Bins</td>
<td>Accounting</td>
</tr>
<tr>
<td>LIST_CAMPAIGN</td>
<td>Marketing Campaigns</td>
<td>Marketing Automation</td>
</tr>
<tr>
<td>LIST_CAMPAIGNHISTORY</td>
<td>Campaign History</td>
<td>Marketing Automation</td>
</tr>
<tr>
<td>LIST_CASE</td>
<td>Cases</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>LIST_CATEGORY</td>
<td>Expense Categories</td>
<td>Expense Reports</td>
</tr>
<tr>
<td>LIST_CLASS</td>
<td>Classes</td>
<td>Classes</td>
</tr>
<tr>
<td>LIST_COMMISSIONRULES</td>
<td>Employee Commission Schedules/Plans</td>
<td>Employee Commissions</td>
</tr>
<tr>
<td>LIST_COMPANY</td>
<td>Companies</td>
<td>Customer Relationship Management</td>
</tr>
<tr>
<td>LIST_COMPETITOR</td>
<td>Competitors</td>
<td>Sales Force Automation</td>
</tr>
<tr>
<td>LIST_CRMTEMPLATE</td>
<td>Marketing Template</td>
<td>Marketing Automation</td>
</tr>
<tr>
<td>LIST_CURRENCY</td>
<td>Currency</td>
<td>Multiple Currencies</td>
</tr>
<tr>
<td>LIST_DEPARTMENT</td>
<td>Departments</td>
<td>Departments</td>
</tr>
<tr>
<td>LIST_DUPLICATES</td>
<td>Duplicate Record Management</td>
<td>Duplicate Detection &amp; Merge</td>
</tr>
<tr>
<td>LIST_FAXTEMPLATE</td>
<td>Fax Template</td>
<td>Mail Merge</td>
</tr>
<tr>
<td>LIST_FILECABINET</td>
<td>Documents and Files</td>
<td>File Cabinet</td>
</tr>
<tr>
<td>LIST_INFOCATEGORY</td>
<td>Store Content Categories</td>
<td>Web Store</td>
</tr>
<tr>
<td>LIST_ISSUE</td>
<td>Issues</td>
<td>Issue Management</td>
</tr>
<tr>
<td>LIST_JOB</td>
<td>Jobs</td>
<td>Projects</td>
</tr>
<tr>
<td>LIST_KNOWLEDGEBASE</td>
<td>Knowledge Base</td>
<td>Knowledge Base</td>
</tr>
<tr>
<td>LIST_LOCATION</td>
<td>Locations</td>
<td>Locations</td>
</tr>
<tr>
<td>LIST_MAILMERGE</td>
<td>Mail Merge</td>
<td>Mail Merge</td>
</tr>
<tr>
<td>LIST_MAILTEMPLATE</td>
<td>Letter Template</td>
<td>Mail Merge</td>
</tr>
<tr>
<td>LIST_MEDIAITEMFOLDER</td>
<td>Media Folders</td>
<td>File Cabinet</td>
</tr>
<tr>
<td>LIST_PARTNER</td>
<td>Partners</td>
<td>Partner Relationship Management</td>
</tr>
<tr>
<td>Permission ID</td>
<td>Permission Name</td>
<td>Feature</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>LIST_PARTNERCOMMISNRULES</td>
<td>Partner Commission Schedules/Plans</td>
<td>Partner Commissions/Royalties</td>
</tr>
<tr>
<td>LIST_PAYCHECK</td>
<td>Paychecks</td>
<td>Payroll</td>
</tr>
<tr>
<td>LIST_PAYROLLITEM</td>
<td>Payroll Items</td>
<td>Payroll</td>
</tr>
<tr>
<td>LIST_PDFTEMPLATE</td>
<td>PDF Template</td>
<td>Mail Merge</td>
</tr>
<tr>
<td>LIST_PROJECTTASK</td>
<td>Project Tasks</td>
<td>Advanced Projects</td>
</tr>
<tr>
<td>LIST_PROMOTIONCODE</td>
<td>Promotion Code</td>
<td>Sales Force Automation</td>
</tr>
<tr>
<td>LIST_QUANTITYPRICINGSCHEDULE</td>
<td>Quantity pricing Schedules</td>
<td>Quantity Pricing</td>
</tr>
<tr>
<td>LIST_RELATEDITEMS</td>
<td>Related Items</td>
<td>Web Site</td>
</tr>
<tr>
<td>LIST_REVRECSCHEDULE</td>
<td>Revenue Recognition Schedules</td>
<td>Revenue Recognition</td>
</tr>
<tr>
<td>LIST_REVRECVSOE</td>
<td>Revenue Management VSOE</td>
<td>VSOE</td>
</tr>
<tr>
<td>LIST_SALESCAMPAIGN</td>
<td>Sales Campaigns</td>
<td>Sales Campaigns</td>
</tr>
<tr>
<td>LIST_SALESROLE</td>
<td>Sales Roles</td>
<td>Team Selling</td>
</tr>
<tr>
<td>LIST_SHIPITEM</td>
<td>Shipping Items</td>
<td>Accounting</td>
</tr>
<tr>
<td>LIST_STORECATEGORY</td>
<td>Store Categories</td>
<td>Web Store</td>
</tr>
<tr>
<td>LIST_SUBSIDIARY</td>
<td>Subsidiaries</td>
<td>Subsidiaries</td>
</tr>
<tr>
<td>LIST_SYSTEMEMAILTEMPLATE</td>
<td>System Email Template</td>
<td>Customer Relationship Management</td>
</tr>
<tr>
<td>LIST_TAXITEM</td>
<td>Tax Items</td>
<td>Accounting</td>
</tr>
<tr>
<td>LIST_TAXSCHEDULE</td>
<td>Tax Schedules</td>
<td>Advanced Taxes</td>
</tr>
<tr>
<td>LIST_TEGERATAACCOUNT</td>
<td>Tegata Accounts</td>
<td>Accounting</td>
</tr>
<tr>
<td>LIST_UNIT</td>
<td>Units</td>
<td>Accounting</td>
</tr>
<tr>
<td>LIST_UPSELL</td>
<td>Upsell Assistant</td>
<td>Upsell Manager</td>
</tr>
<tr>
<td>LIST_UPSELLWIZARD</td>
<td>Upsell Wizard</td>
<td>Upsell Manager</td>
</tr>
<tr>
<td>LIST_VENDOR</td>
<td>Vendors</td>
<td>Accounting</td>
</tr>
<tr>
<td>LIST_WEBSITE</td>
<td>Website (External) publisher</td>
<td>Web Site</td>
</tr>
<tr>
<td>LIST_WORKPLACE</td>
<td>Workplaces</td>
<td>Payroll</td>
</tr>
<tr>
<td>REGT_ACCTPAY</td>
<td>Accounts Payable Register</td>
<td>A/P</td>
</tr>
<tr>
<td>REGT_ACCTREC</td>
<td>Accounts Receivable Register</td>
<td>A/R</td>
</tr>
<tr>
<td>REGT_BANK</td>
<td>Bank Account Registers</td>
<td>Accounting</td>
</tr>
<tr>
<td>REGT_COGS</td>
<td>Cost of Goods Sold Registers</td>
<td>Accounting</td>
</tr>
<tr>
<td>REGT_CRED CARD</td>
<td>Credit Card Registers</td>
<td>Accounting</td>
</tr>
<tr>
<td>REGT_DEFEREXPENSE</td>
<td>Deferred Expense Registers</td>
<td>Amortization</td>
</tr>
<tr>
<td>REGT_DEFERREVENUE</td>
<td>Deferred Revenue Registers</td>
<td>Revenue Recognition</td>
</tr>
<tr>
<td>REGT_EQUITY</td>
<td>Equity Registers</td>
<td>Accounting</td>
</tr>
<tr>
<td>REGT_EXPENSE</td>
<td>Expense Registers</td>
<td>Accounting</td>
</tr>
<tr>
<td>REGT_FIXEDASSET</td>
<td>Fixed Asset Registers</td>
<td>Accounting</td>
</tr>
<tr>
<td>Permission ID</td>
<td>Permission Name</td>
<td>Feature</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>REGT_INCOME</td>
<td>Income Registers</td>
<td>Accounting</td>
</tr>
<tr>
<td>REGT_LONGTERMLIAB</td>
<td>Long Term Liability Registers</td>
<td>Accounting</td>
</tr>
<tr>
<td>REGT_NONPOSTING</td>
<td>Non Posting Registers</td>
<td>Accounting</td>
</tr>
<tr>
<td>REGT_OTHASSET</td>
<td>Other Asset Registers</td>
<td>Accounting</td>
</tr>
<tr>
<td>REGT_OTHCURRASSET</td>
<td>Other Current Asset Registers</td>
<td>Accounting</td>
</tr>
<tr>
<td>REGT_OTHCURRLIAB</td>
<td>Other Current Liability Registers</td>
<td>Accounting</td>
</tr>
<tr>
<td>REGT_OTHEXPENSE</td>
<td>Other Expense Registers</td>
<td>Accounting</td>
</tr>
<tr>
<td>REGT_OTHINCOME</td>
<td>Other Income Registers</td>
<td>Accounting</td>
</tr>
<tr>
<td>REGT_PAYROLL</td>
<td>Run Payroll</td>
<td>Payroll</td>
</tr>
<tr>
<td>REGT_UNBILLEDREC</td>
<td>Unbilled Receivable Registers</td>
<td>Revenue Commitments</td>
</tr>
<tr>
<td>REPO_1099</td>
<td>Form 1099 - Federal Miscellaneous</td>
<td>A/P</td>
</tr>
<tr>
<td>REPO_940</td>
<td>Form 940 - Employer's Annual</td>
<td>Payroll</td>
</tr>
<tr>
<td></td>
<td>Federal Unemployment Tax Return</td>
<td></td>
</tr>
<tr>
<td>REPO_941</td>
<td>Form 941 - Employer's Quarterly</td>
<td>Payroll</td>
</tr>
<tr>
<td></td>
<td>Federal Tax Return</td>
<td></td>
</tr>
<tr>
<td>REPO_AMORTIZATION</td>
<td>Amortization Reports</td>
<td>Amortization</td>
</tr>
<tr>
<td>REPO_AP</td>
<td>Accounts Payable</td>
<td>A/P</td>
</tr>
<tr>
<td>REPO_AR</td>
<td>Accounts Receivable</td>
<td>A/R</td>
</tr>
<tr>
<td>REPO_AUTHPARTNERCOMMISSION</td>
<td>Partner Authorized Commission Reports</td>
<td>Partner Commissions/Royalties</td>
</tr>
<tr>
<td>REPO_BALANCESHEET</td>
<td>Balance Sheet</td>
<td>Accounting</td>
</tr>
<tr>
<td>REPO_BUDGET</td>
<td>Budget</td>
<td>Accounting</td>
</tr>
<tr>
<td>REPO_CASHFLOW</td>
<td>Cash Flow Statement</td>
<td>Accounting</td>
</tr>
<tr>
<td>REPO_COMMISSION</td>
<td>Commission Reports</td>
<td>Employee Commissions</td>
</tr>
<tr>
<td>REPO_FINANCIALS</td>
<td>Financial Statements</td>
<td>Accounting</td>
</tr>
<tr>
<td>REPO_GL</td>
<td>General Ledger</td>
<td>Accounting</td>
</tr>
<tr>
<td>REPO_INVENTORY</td>
<td>Inventory</td>
<td></td>
</tr>
<tr>
<td>REPO_ISSUE</td>
<td>Issue Reports</td>
<td>Issue Management</td>
</tr>
<tr>
<td>REPO_MARKETING</td>
<td>Marketing Campaign Reports</td>
<td>Marketing Automation</td>
</tr>
<tr>
<td>REPO_PANDL</td>
<td>Income Statement</td>
<td>Accounting</td>
</tr>
<tr>
<td>REPO_PARTNERCOMMISSION</td>
<td>Partner Commission Reports</td>
<td>Partner Commissions/Royalties</td>
</tr>
<tr>
<td>REPO_PAYCHECKDETAIL</td>
<td>Payroll Check Register</td>
<td>Payroll</td>
</tr>
<tr>
<td>REPO_PAYROLL</td>
<td>Payroll Summary &amp; Detail Reports</td>
<td>Payroll</td>
</tr>
<tr>
<td>REPO_PAYROLLHOURSEARNING</td>
<td>Payroll Hours &amp; Earnings</td>
<td>Payroll</td>
</tr>
<tr>
<td>REPO_PAYROLLJOURNAL</td>
<td>Payroll Journal Report</td>
<td>Payroll</td>
</tr>
<tr>
<td>REPO_PAYROLLLIAB</td>
<td>Payroll Liability Report</td>
<td>Payroll</td>
</tr>
<tr>
<td>Permission ID</td>
<td>Permission Name</td>
<td>Feature</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>REPO_PAYROLLSTATEWITHHOLD</td>
<td>Payroll State Withholding</td>
<td>Payroll</td>
</tr>
<tr>
<td>REPO_PAYROLLW2</td>
<td>Form W-2 - Wage and Tax Statement</td>
<td>Payroll</td>
</tr>
<tr>
<td>REPO_PURCHASEORDER</td>
<td>Purchase Order Reports</td>
<td>Purchase Orders</td>
</tr>
<tr>
<td>REPO_PURCHASES</td>
<td>Purchases</td>
<td>Accounting</td>
</tr>
<tr>
<td>REPO_QUOTA</td>
<td>Quota Reports</td>
<td>Sales Force Automation</td>
</tr>
<tr>
<td>REPO_RECONCILE</td>
<td>Reconcile Reporting</td>
<td>Accounting</td>
</tr>
<tr>
<td>REPO_RETURNAUTH</td>
<td>Return Authorization Reports</td>
<td>Return Authorizations</td>
</tr>
<tr>
<td>REPO_REVREC</td>
<td>Revenue Recognition Reports</td>
<td>Revenue Recognition</td>
</tr>
<tr>
<td>REPO_SALESORDER</td>
<td>Sales Order Fulfillment Reports</td>
<td>Sales Orders</td>
</tr>
<tr>
<td>REPO_SALES_PARTNER</td>
<td>Sales By Partner</td>
<td>Partner Relationship Management</td>
</tr>
<tr>
<td>REPO_SALES_PROMO</td>
<td>Sales By Promotion Code</td>
<td>Sales Force Automation</td>
</tr>
<tr>
<td>REPO_SFA</td>
<td>Sales Force Automation</td>
<td>Customer Relationship Management</td>
</tr>
<tr>
<td>REPO_SNAPSHOTCASE</td>
<td>Support Case Snapshot/Reminders</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>REPO_SUPPORT</td>
<td>Support</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>REPO_TAX</td>
<td>Tax</td>
<td>Accounting</td>
</tr>
<tr>
<td>REPO_TIME</td>
<td>Time Tracking</td>
<td>Time Tracking</td>
</tr>
<tr>
<td>REPO_TRIALBALANCE</td>
<td>Trial Balance</td>
<td>Accounting</td>
</tr>
<tr>
<td>REPO_UNBILLED</td>
<td>Accounts Receivable Un-Billed</td>
<td>Accounting</td>
</tr>
<tr>
<td>REPO_W4</td>
<td>Form W4 - Employee’s Withholding Allowance Certificate</td>
<td>Payroll</td>
</tr>
<tr>
<td>REPO_WEBSITE</td>
<td>Web Site Report</td>
<td>Web Site</td>
</tr>
<tr>
<td>REPO_WEBSTORE</td>
<td>Web Store Report</td>
<td>Web Store</td>
</tr>
<tr>
<td>TRAN_ALLOCOSCHEDULE</td>
<td>Create Allocation Schedules</td>
<td>Expense Allocation</td>
</tr>
<tr>
<td>TRAN_APPROPECOMMISSN</td>
<td>Employee Commission Transaction Approval</td>
<td>Employee Commissions</td>
</tr>
<tr>
<td>TRAN_APPROVEDD</td>
<td>Approve Direct Deposit</td>
<td>Direct Deposit</td>
</tr>
<tr>
<td>TRAN_APPROVEEFT</td>
<td>Approve EFT</td>
<td>Electronic Funds Transfer</td>
</tr>
<tr>
<td>TRAN_APPROVEPARTNERCOMM</td>
<td>Partner Commission Transaction Approval</td>
<td>Partner Commissions/Royalties</td>
</tr>
<tr>
<td>TRAN_APPROVEVP</td>
<td>Approve Vendor Payments</td>
<td>ACH Vendor Payments</td>
</tr>
<tr>
<td>TRAN_BILLPAY_APPROVE</td>
<td>Approve Online Bill Payments</td>
<td>Online Bill Pay</td>
</tr>
<tr>
<td>TRAN_BILLPAY_STATUS</td>
<td>View Online Bill Pay Status</td>
<td>Online Bill Pay</td>
</tr>
<tr>
<td>TRAN_BINTRNFR</td>
<td>Bin Transfer</td>
<td>Bin Management</td>
</tr>
<tr>
<td>TRAN_BINWKSHT</td>
<td>Bin Putaway Worksheet</td>
<td>Bin Management</td>
</tr>
<tr>
<td>TRAN_BUDGET</td>
<td>Set Up Budgets</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_BUILD</td>
<td>Build Assemblies</td>
<td>Assembly Items</td>
</tr>
<tr>
<td>Permission ID</td>
<td>Permission Name</td>
<td>Feature</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>TRAN_CARDCHRG</td>
<td>Credit Card</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_CARDRFND</td>
<td>Credit Card Refund</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_CASHRFND</td>
<td>Cash Sale Refund</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_CASHSALE</td>
<td>Cash Sale</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_CHECK</td>
<td>Check</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_COMMISSN</td>
<td>Employee Commission Transaction</td>
<td>Employee Commissions</td>
</tr>
<tr>
<td>TRAN_COMMITPAYROLL</td>
<td>Commit Payroll</td>
<td>Payroll</td>
</tr>
<tr>
<td>TRAN_COPY_BUDGET</td>
<td>Copy Budgets</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_CUSTCHRG</td>
<td>Statement Charge</td>
<td>A/R</td>
</tr>
<tr>
<td>TRAN_CUSTCRED</td>
<td>Credit Memo</td>
<td>A/R</td>
</tr>
<tr>
<td>TRAN_CUSTDEP</td>
<td>Customer Deposit</td>
<td>A/R</td>
</tr>
<tr>
<td>TRAN_CUSTINVC</td>
<td>Invoice</td>
<td>A/R</td>
</tr>
<tr>
<td>TRAN_CUSTPYMT</td>
<td>Customer Payment</td>
<td>A/R</td>
</tr>
<tr>
<td>TRAN_CUSTRFND</td>
<td>Customer Refund</td>
<td>A/R</td>
</tr>
<tr>
<td>TRAN_DEPAPPL</td>
<td>Deposit Application</td>
<td>A/R</td>
</tr>
<tr>
<td>TRAN_DEPOSIT</td>
<td>Deposit</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_ESTIMATE</td>
<td>Estimate</td>
<td>Estimates</td>
</tr>
<tr>
<td>TRAN_ESTIMATEDCOSTOVERRIDE</td>
<td>Override Estimated Cost on Transactions</td>
<td>Gross Profit</td>
</tr>
<tr>
<td>TRAN_EXPREPT</td>
<td>Expense Report</td>
<td>Expense Reports</td>
</tr>
<tr>
<td>TRAN_FINCHRG</td>
<td>Finance Charge</td>
<td>A/R</td>
</tr>
<tr>
<td>TRAN_FINDOLBMATCH</td>
<td>Find Matching Online Banking</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_FORECAST</td>
<td>Edit Forecast</td>
<td>Sales Force Automation</td>
</tr>
<tr>
<td>TRAN_FXREVAL</td>
<td>Currency Revaluation</td>
<td>Multiple Currencies</td>
</tr>
<tr>
<td>TRAN_IMPORTOLBFILE</td>
<td>Import Online Banking (QIF) File</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_INVAJDST</td>
<td>Adjust Inventory</td>
<td>Inventory</td>
</tr>
<tr>
<td>TRAN_INVDISTR</td>
<td>Distribute Inventory</td>
<td>Multi-Location Inventory</td>
</tr>
<tr>
<td>TRAN_INVTRNFR</td>
<td>Transfer Inventory</td>
<td>Multi-Location Inventory</td>
</tr>
<tr>
<td>TRAN_INVWKSHT</td>
<td>Adjust Inventory Worksheet</td>
<td>Inventory</td>
</tr>
<tr>
<td>TRAN_ITEMRCPT</td>
<td>Receive Items</td>
<td>Advanced Receiving</td>
</tr>
<tr>
<td>TRAN_ITEMSHIP</td>
<td>Ship Items</td>
<td>Advanced Shipping</td>
</tr>
<tr>
<td>TRAN_JOURNAL</td>
<td>Make Journal Entry</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_JOURNALAPPRV</td>
<td>Journal Approval</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_LIAPBYMT</td>
<td>Payroll Liability Payments</td>
<td>Payroll</td>
</tr>
<tr>
<td>TRAN_MANAGEPAYROLL</td>
<td>Manage Payroll</td>
<td>Payroll</td>
</tr>
<tr>
<td>Permission ID</td>
<td>Permission Name</td>
<td>Feature</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>TRAN_MGRFORECAST</td>
<td>Edit Manager Forecast</td>
<td>Sales Force Automation</td>
</tr>
<tr>
<td>TRAN_OLBSTATEMENT</td>
<td>Online Banking Statement</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_OPENBAL</td>
<td>Enter Opening Balances</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_OPPRTNTY</td>
<td>Opportunity</td>
<td>Opportunities</td>
</tr>
<tr>
<td>TRAN_PARTNERCOMMISSN</td>
<td>Partner Commission Transaction</td>
<td>Partner Commissions/Royalties</td>
</tr>
<tr>
<td>TRAN_PAYCHECK</td>
<td>Individual Paycheck</td>
<td>Payroll</td>
</tr>
<tr>
<td>TRAN_PAYROLLRUN</td>
<td>Process Payroll</td>
<td>Payroll</td>
</tr>
<tr>
<td>TRAN_POSTPERIODS</td>
<td>Posting Period on Transactions</td>
<td>Accounting Periods</td>
</tr>
<tr>
<td>TRAN_PURCHORD</td>
<td>Purchase Order</td>
<td>Purchase Orders</td>
</tr>
<tr>
<td>TRAN_PURCHORDBILL</td>
<td>Bill Purchase Orders</td>
<td>Advanced Receiving</td>
</tr>
<tr>
<td>TRAN_PURCHORDRECEIVE</td>
<td>Receive Purchase Orders</td>
<td>Purchase Orders</td>
</tr>
<tr>
<td>TRAN_QUOTA</td>
<td>Establish Quotas</td>
<td>Sales Force Automation</td>
</tr>
<tr>
<td>TRAN_RECONCILE</td>
<td>Reconcile</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_REVCOMM</td>
<td>Revenue Commitment</td>
<td>Revenue Commitments</td>
</tr>
<tr>
<td>TRAN_REVCOMRV</td>
<td>Revenue Commitment Reversal</td>
<td>Revenue Commitments</td>
</tr>
<tr>
<td>TRAN_RTNAUTH</td>
<td>Return Authorization</td>
<td>Return Authorizations</td>
</tr>
<tr>
<td>TRAN_RTNAUTHAPPRV</td>
<td>Return Auth. Approval</td>
<td>Return Authorizations</td>
</tr>
<tr>
<td>TRAN_RTNAUTHREVERSERCOMMIT</td>
<td>Generate Revenue Commitment Reversals</td>
<td>Revenue Commitments</td>
</tr>
<tr>
<td>TRAN_SALESORD</td>
<td>Sales Order</td>
<td>Sales Orders</td>
</tr>
<tr>
<td>TRAN_SALESORDAPPRV</td>
<td>Sales Order Approval</td>
<td>Sales Orders</td>
</tr>
<tr>
<td>TRAN_SALESORDCOMMITREVENUE</td>
<td>Generate Revenue Commitment</td>
<td>Revenue Commitments</td>
</tr>
<tr>
<td>TRAN_SALESORDINVOICE</td>
<td>Bill Sales Orders</td>
<td>Advanced Shipping</td>
</tr>
<tr>
<td>TRAN_STATEMENT</td>
<td>Generate Statements</td>
<td>A/R</td>
</tr>
<tr>
<td>TRAN_STATUSDD</td>
<td>Direct Deposit Status</td>
<td>Direct Deposit</td>
</tr>
<tr>
<td>TRAN_STATUSEFT</td>
<td>EFT Status</td>
<td>Electronic Funds Transfer</td>
</tr>
<tr>
<td>TRAN_STATUSVP</td>
<td>Vendor Payment Status</td>
<td>ACH Vendor Payments</td>
</tr>
<tr>
<td>TRAN_TAXLIAB</td>
<td>Pay Tax Liability</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_TAXPYMT</td>
<td>Pay Sales Tax</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_TEGPYBL</td>
<td>Tegata Payable</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_TEGRCVBL</td>
<td>Tegata Receivable</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_TIMEBILL</td>
<td>Track Time</td>
<td>Time Tracking</td>
</tr>
<tr>
<td>TRAN_TIMECALC</td>
<td>Calculate Time</td>
<td>Time Tracking</td>
</tr>
<tr>
<td>TRAN_TIMER</td>
<td>Timer</td>
<td>Time Tracking</td>
</tr>
<tr>
<td>TRAN_TRANSFER</td>
<td>Transfer Funds</td>
<td>Accounting</td>
</tr>
<tr>
<td>TRAN_UNBUILD</td>
<td>Unbuild Assemblies</td>
<td>Assembly Items</td>
</tr>
<tr>
<td>Permission ID</td>
<td>Permission Name</td>
<td>Feature</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>TRAN_VENDAUTH</td>
<td>Vendor Return Authorization</td>
<td>Vendor Return Authorizations</td>
</tr>
<tr>
<td>TRAN_VENDAUTHAPPRV</td>
<td>Vendor Return Auth. Approval</td>
<td>Vendor Return Authorizations</td>
</tr>
<tr>
<td>TRAN_VENDAUTHRETURN</td>
<td>Vendor Returns</td>
<td>Vendor Return Authorizations</td>
</tr>
<tr>
<td>TRAN_VENDBILL</td>
<td>Bills</td>
<td>A/P</td>
</tr>
<tr>
<td>TRAN_VENDCRED</td>
<td>Enter Vendor Credits</td>
<td>A/P</td>
</tr>
<tr>
<td>TRAN_VENDPYMT</td>
<td>Pay Bills</td>
<td>A/P</td>
</tr>
<tr>
<td>TRAN_WORKORD</td>
<td>Work Order</td>
<td>Work Orders</td>
</tr>
<tr>
<td>TRAN_WORKORDBUILD</td>
<td>Build Work Orders</td>
<td>Work Orders</td>
</tr>
<tr>
<td>TRAN_YTDADJST</td>
<td>Enter Year-To-Date Payroll</td>
<td>Payroll</td>
</tr>
</tbody>
</table>
Chapter 75 Preference Names and IDs

The following tables list the internal IDs for all NetSuite preference configuration pages that support SuiteScript. To set values on any of these pages, you must first load the page using `nlapiLoadConfiguration(type)`.

Once the page loads, you can then get/set all configuration values using the methods on the `nlobjConfiguration` object.

NetSuite configuration preference IDs are grouped into the following categories:

- General Preferences
- Company Information
- Accounting Preferences
- Accounting Periods
- Tax Setup

**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`.

<table>
<thead>
<tr>
<th>Preference UI Label</th>
<th>Preference Internal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use State Abbreviations in Addresses</td>
<td>abbreviatestates</td>
</tr>
<tr>
<td>Assign Tasks to Partners</td>
<td>assigntaskstopartners</td>
</tr>
<tr>
<td>Auto Name Customers</td>
<td>autonamecustomers</td>
</tr>
<tr>
<td>Calendar System</td>
<td>calendarsystem</td>
</tr>
<tr>
<td>Company Logo Folder</td>
<td>companylogofolder</td>
</tr>
<tr>
<td>Add Primary Contact to Bill To Address</td>
<td>contactonbillto</td>
</tr>
<tr>
<td>Default Role for New Customers</td>
<td>customerrole</td>
</tr>
<tr>
<td>Default Customer Type</td>
<td>customertype</td>
</tr>
<tr>
<td>Customer Center Welcome Message</td>
<td>customerwelcomemessage</td>
</tr>
<tr>
<td>Date Format</td>
<td>dateformat</td>
</tr>
<tr>
<td>Delay Loading of Sublists</td>
<td>delayloadingsublists</td>
</tr>
<tr>
<td>Default Partner Type</td>
<td>dfltpartnertype</td>
</tr>
<tr>
<td>Preference UI Label</td>
<td>Preference Internal ID</td>
</tr>
<tr>
<td>---------------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Default Vendor Type</td>
<td>dftvendortype</td>
</tr>
<tr>
<td>Email Employee on Approvals</td>
<td>emailemployeeonapproval</td>
</tr>
<tr>
<td>Maintenance Complete Email Notification</td>
<td>emaillmaintenencecomplete</td>
</tr>
<tr>
<td>First Day of Week</td>
<td>firstdayofweek</td>
</tr>
<tr>
<td>Screen Font</td>
<td>font</td>
</tr>
<tr>
<td>Allow Free-Form States in Addresses</td>
<td>freeformstates</td>
</tr>
<tr>
<td>Hide Attachment Folders</td>
<td>hideattachmentfolders</td>
</tr>
<tr>
<td>Internal Web Site</td>
<td>internalwebsite</td>
</tr>
<tr>
<td>Show Display Name with Item Codes</td>
<td>itemnumbering</td>
</tr>
<tr>
<td>Use Last Name First for Employees</td>
<td>lastnamefirst</td>
</tr>
<tr>
<td>Use Last Name First for Entities</td>
<td>lastnamefirstentities</td>
</tr>
<tr>
<td>Number of rows in List segments</td>
<td>listsegmentsize</td>
</tr>
<tr>
<td>Long Date Format</td>
<td>longdateformat</td>
</tr>
<tr>
<td>Maximum entries in Dropdown</td>
<td>maxdroppdownsize</td>
</tr>
<tr>
<td>Maximum number of dimension columns allowed in a report</td>
<td>maxreportdimensions</td>
</tr>
<tr>
<td>Maximum number of rows allowed in a report</td>
<td>maxreportrows</td>
</tr>
<tr>
<td>Minimum Password Length</td>
<td>minpasswordlength</td>
</tr>
<tr>
<td>Password Expiration in Days</td>
<td>passwordexpiredays</td>
</tr>
<tr>
<td>Phone Number Format</td>
<td>phoneformat</td>
</tr>
<tr>
<td>Pre-Populate Contact Address</td>
<td>prepopulatecontactaddress</td>
</tr>
<tr>
<td>Show Reports in Grid</td>
<td>reportgrid</td>
</tr>
<tr>
<td>Show Employees as Contacts</td>
<td>showemployeesascontacts</td>
</tr>
<tr>
<td>Show Individuals as Contacts</td>
<td>showindividualascontacts</td>
</tr>
<tr>
<td>Show List When Only One Results</td>
<td>showlistoneresult</td>
</tr>
<tr>
<td>Show Page Feedback Link</td>
<td>showpagefeedbacklink</td>
</tr>
<tr>
<td>Show Quick Add Row on Lists</td>
<td>showquickadd</td>
</tr>
<tr>
<td>Time Format</td>
<td>timeformat</td>
</tr>
</tbody>
</table>

**Company Information**

These are the account preferences that can be found by going to Setup > Company > Company Information.

The internal ID for the Company Preferences page is **companyinformation**.
<table>
<thead>
<tr>
<th>Preference UI Label</th>
<th>Preference Internal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company Name</td>
<td>companyname</td>
</tr>
<tr>
<td>Legal Name</td>
<td>legalname</td>
</tr>
<tr>
<td>Ship to Attention</td>
<td>attention</td>
</tr>
<tr>
<td>Address 1</td>
<td>address1</td>
</tr>
<tr>
<td>Address 2</td>
<td>address2</td>
</tr>
<tr>
<td>City</td>
<td>city</td>
</tr>
<tr>
<td>County/State/Province</td>
<td>state</td>
</tr>
<tr>
<td>Zip</td>
<td>zip</td>
</tr>
<tr>
<td>Country</td>
<td>country</td>
</tr>
<tr>
<td>Address</td>
<td>addresstext</td>
</tr>
<tr>
<td>Employer Identification Number (EIN)</td>
<td>employerid</td>
</tr>
<tr>
<td>SSN or TIN (Social Security Number, Tax ID Number)</td>
<td>taxid</td>
</tr>
<tr>
<td>Return Email Address</td>
<td>email</td>
</tr>
<tr>
<td>Phone</td>
<td>phone</td>
</tr>
<tr>
<td>Fax</td>
<td>fax</td>
</tr>
<tr>
<td>Web Site</td>
<td>url</td>
</tr>
<tr>
<td>Company Logo (Forms)</td>
<td>formlogo</td>
</tr>
<tr>
<td>Company Logo (Pages)</td>
<td>pagelogo</td>
</tr>
<tr>
<td>Display Logo Internally</td>
<td>displaylogointernal</td>
</tr>
<tr>
<td>First Fiscal Month</td>
<td>fiscalmonth</td>
</tr>
<tr>
<td>Time Zone</td>
<td>timezone</td>
</tr>
<tr>
<td>Currency</td>
<td>basecurrency</td>
</tr>
</tbody>
</table>

**On the Shipping Address tab**

| Address 1                                  | shippingaddress1       |
| Address 2                                  | shippingaddress2       |
| City                                       | shippingcity           |
| County/State/Province                      | shippingstate          |
| Zip                                        | shippingzip            |
| Country                                    | shippingcountry        |
| Address                                    | shippingaddresstext    |

**On the Return Address tab**

| Address 1                                  | returnaddress1         |
| Address 2                                  | returnaddress2         |
| City                                       | returncity             |
Accounting Preferences

These are the account preferences that can be found by going to Setup > Accounting > Accounting Preferences.

The internal ID for the Accounting Preferences page is `accountingpreferences`.

<table>
<thead>
<tr>
<th>Preference UI Label</th>
<th>Preference Internal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>County/State/Province</td>
<td>returnstate</td>
</tr>
<tr>
<td>Zip</td>
<td>returnzip</td>
</tr>
<tr>
<td>Country</td>
<td>returncountry</td>
</tr>
<tr>
<td>Address</td>
<td>returnaddresstext</td>
</tr>
</tbody>
</table>

### On the General tab

<table>
<thead>
<tr>
<th>Preference UI Label</th>
<th>Preference Internal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Account Numbers</td>
<td>accountnumbers</td>
</tr>
<tr>
<td>Aging Reports Use</td>
<td>agefrom</td>
</tr>
<tr>
<td>Allow cross-subsidiary billable time and expenses</td>
<td>allowcrosssubbillables</td>
</tr>
<tr>
<td>Allow manual entry of Gift Certificate Codes</td>
<td>allowmanualgcode</td>
</tr>
<tr>
<td>Allow subsidiary hierarchy to be modified</td>
<td>allowsubsidhierarchychange</td>
</tr>
<tr>
<td>Default Amortization Journal Date to</td>
<td>amorjournaldatedefault</td>
</tr>
<tr>
<td>Expand Account Lists</td>
<td>expandaccountlists</td>
</tr>
<tr>
<td>Revenue Recognition/Adv. Billing: Use Sales Order Amount</td>
<td>calcpctcompfroomsalesorderamt</td>
</tr>
<tr>
<td>Cash Basis Reporting</td>
<td>cashbasis</td>
</tr>
<tr>
<td>Always Allow Per-line Classifications on Journals</td>
<td>cdelperlineonje</td>
</tr>
<tr>
<td>Allow Per-Line Classes</td>
<td>classesperline</td>
</tr>
<tr>
<td>Make Classes Mandatory</td>
<td>classmandatory</td>
</tr>
<tr>
<td>Allow Admins to Post in Closed Periods</td>
<td>closedperiodoverride</td>
</tr>
<tr>
<td>Accept Payments through Top-level Customer</td>
<td>consolpayments</td>
</tr>
<tr>
<td>Days Overdue for Warning/Hold</td>
<td>credlimdays</td>
</tr>
<tr>
<td>Display Current Count on Adjustments</td>
<td>curcounetonadjustments</td>
</tr>
<tr>
<td>Display Current Count on Transfers</td>
<td>curcountontransfers</td>
</tr>
<tr>
<td>Display Current Count on Worksheets</td>
<td>curcounetonworksheets</td>
</tr>
<tr>
<td>Customer Credit Limit Handling</td>
<td>custcredlimhandling</td>
</tr>
<tr>
<td>Customer Credit Limit Includes Orders</td>
<td>custcredlimorders</td>
</tr>
<tr>
<td>Make Departments Mandatory</td>
<td>deptmandatory</td>
</tr>
<tr>
<td>Preference UI Label</td>
<td>Preference Internal ID</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>Allow Per-Line Departments</td>
<td>deptsperline</td>
</tr>
<tr>
<td>Require Approvals on Journal Entries</td>
<td>journalapprovals</td>
</tr>
<tr>
<td>Make Locations Mandatory</td>
<td>locmandatory</td>
</tr>
<tr>
<td>Allow Per-Line Locations</td>
<td>locsperline</td>
</tr>
<tr>
<td>Maximum number of MLJ locations</td>
<td>maxlocations</td>
</tr>
<tr>
<td>Maximum number of Subsidiaries</td>
<td>maxsubsidiaries</td>
</tr>
<tr>
<td>Allow Users to Modify Amortization Schedule</td>
<td>modifyamortotalamount</td>
</tr>
<tr>
<td>Allow Users to Modify Revenue Recognition Schedule</td>
<td>modifyrevrectotalamount</td>
</tr>
<tr>
<td>Allow Users to Modify VSOE Values on Transactions</td>
<td>modifyvsoevalsontran</td>
</tr>
<tr>
<td>Name for Tax Amount</td>
<td>namingtaxamount</td>
</tr>
<tr>
<td>Name for Tax Rate</td>
<td>namingtaxrate</td>
</tr>
<tr>
<td>Name for Tax Reg. Number</td>
<td>namingtaxregnumber</td>
</tr>
<tr>
<td>Allow Non-balancing Classifications on Journals</td>
<td>nonbalancingcdlonje</td>
</tr>
<tr>
<td>Allow Empty Classifications on Journals</td>
<td>nulcdlonje</td>
</tr>
<tr>
<td>Show Only Open Transactions on Statements</td>
<td>openonlystmts</td>
</tr>
<tr>
<td>Prorate Revenue Recognition Dates For Partially Billed Sales Orders</td>
<td>proraterevrecinfromso</td>
</tr>
<tr>
<td>Restrict Account Balance Viewing for Employees with Classification Restrictions</td>
<td>restrictbalanceviewing</td>
</tr>
<tr>
<td>Void Transactions Using Reversing Journals</td>
<td>reversalvoiding</td>
</tr>
<tr>
<td>Invoice Revenue Recognition Dates Source from</td>
<td>revrecdatesupdatemethod</td>
</tr>
<tr>
<td>Default Revenue Recognition Journal Date to</td>
<td>revrecjournaldatedefault</td>
</tr>
<tr>
<td>Show Journal Memos on Statements</td>
<td>statementjournalmemos</td>
</tr>
<tr>
<td>Include Shipping for Term Discounts</td>
<td>termdiscountsincludeshipping</td>
</tr>
<tr>
<td>Include Tax for Term Discounts</td>
<td>termdiscountsincludetax</td>
</tr>
<tr>
<td>Allow Revenue Commitment Reversals In Advance of Item Receipt</td>
<td>unreceivedrevenuecommitments</td>
</tr>
<tr>
<td>Allow Revenue Commitments In Advance of Fulfillment</td>
<td>unshippedrevenuecommitments</td>
</tr>
<tr>
<td>Use System Calculated Percentage of Completion For Revenue Recognition/Amortization</td>
<td>usesyscalcpct4revrec</td>
</tr>
<tr>
<td>Vendor Credit Limit Includes Orders</td>
<td>vendcredlimorders</td>
</tr>
<tr>
<td>Vendor Credit Limit Warnings</td>
<td>vendcredlimwarnings</td>
</tr>
<tr>
<td>Default Vendor Payments To Be Printed</td>
<td>vendpymtttoprint</td>
</tr>
</tbody>
</table>

**On the Items/Transactions tab**

- Allow Purchase of Assembly Items                                  | allowassemblypurchase       |
- Default Asset Account                                              | assetaccount                |
<table>
<thead>
<tr>
<th>Preference UI Label</th>
<th>Preference Internal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Credit Card Security Code for Credit Card Transactions</td>
<td>ccsecuritycode</td>
</tr>
<tr>
<td>Centralize Purchasing in a Single Location</td>
<td>centralizedpurchasing</td>
</tr>
<tr>
<td>Default COGS Account</td>
<td>cogsaccount</td>
</tr>
<tr>
<td>Consolidate Jobs on Sales Transactions</td>
<td>consolinvoices</td>
</tr>
<tr>
<td>Duplicate Number Warnings</td>
<td>duplicatewarnings</td>
</tr>
<tr>
<td>Anyone Can Set Item Accounts</td>
<td>edititemaccounts</td>
</tr>
<tr>
<td>Default Estimate Expiration (in days)</td>
<td>estimateexpiration</td>
</tr>
<tr>
<td>Use the Exact Cost for Linked Returns</td>
<td>exactcostonlinelinkedreturns</td>
</tr>
<tr>
<td>Default Expense Account</td>
<td>expenseaccount</td>
</tr>
<tr>
<td>Customers Can Pay Online</td>
<td>externalpayments</td>
</tr>
<tr>
<td>Include Reimbursements in Sales and Forecast Reports</td>
<td>forecastincludes_reimb_exp</td>
</tr>
<tr>
<td>Include Shipping in Sales and Forecast Reports</td>
<td>forecastincludes_shipping</td>
</tr>
<tr>
<td>Transaction Types to Exclude from Forecast Reports</td>
<td>forecasttrantypes</td>
</tr>
<tr>
<td>Gift Certificate Auth Code Generation</td>
<td>giftcertauthcodegeneration</td>
</tr>
<tr>
<td>Default Income Account</td>
<td>incomeaccount</td>
</tr>
<tr>
<td>Inventory Costing Method</td>
<td>invtcostmethod</td>
</tr>
<tr>
<td>Days Before Lot Expiration Warning</td>
<td>lotexpirationwarning</td>
</tr>
<tr>
<td>Default Payment Account</td>
<td>paymentaccount</td>
</tr>
<tr>
<td>Payment Due Preference</td>
<td>paymentduepreference</td>
</tr>
<tr>
<td>Purchase Discount Account</td>
<td>purchdisacct</td>
</tr>
<tr>
<td>Maximum # of Quantity-based Price Levels</td>
<td>qtypricecount</td>
</tr>
<tr>
<td>Allow Quantity Discounts per Price Level on Schedules</td>
<td>qtypriceschedulemultdiscounts</td>
</tr>
<tr>
<td>Sort Reconcile By</td>
<td>reconsortcol</td>
</tr>
<tr>
<td>Require Bins on All Transactions Except Item Receipts</td>
<td>requirebinsontrans</td>
</tr>
<tr>
<td>Sales Discount Account</td>
<td>salesdiscacct</td>
</tr>
<tr>
<td>Transaction Types to Exclude from Sales Reports</td>
<td>salestrantypes</td>
</tr>
<tr>
<td>Scan Individual Items</td>
<td>singleitembarcoding</td>
</tr>
<tr>
<td>Matrix Item Name/Number Separator</td>
<td>skuseparator</td>
</tr>
<tr>
<td>Tegata Maturity Date</td>
<td>tegatamaturity</td>
</tr>
<tr>
<td>Recalculate Estimated Cost on Creation of Linked Transactions</td>
<td>uselatestcostestimate</td>
</tr>
<tr>
<td>Use Preferred Bin on Item Receipts</td>
<td>usepreferredbinonitemrcpt</td>
</tr>
<tr>
<td>Use Popup to Select Serial/Lot Numbers on Sales</td>
<td>useserialnumberselect</td>
</tr>
<tr>
<td>Use Zero Cost for Linked Underwater Sales</td>
<td>zeroconstunderwater</td>
</tr>
<tr>
<td><strong>On the Order Management tab</strong></td>
<td></td>
</tr>
<tr>
<td>Preference UI Label</td>
<td>Preference Internal ID</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>Build Based on Commitment</td>
<td>buildcommitted</td>
</tr>
<tr>
<td>Filter Bulk Fulfillment Page by Location</td>
<td>bulkfulloctfiltering</td>
</tr>
<tr>
<td>Convert Absolute Discounts to Percentage When Billing</td>
<td>convertabsolutediscounts</td>
</tr>
<tr>
<td>Default Location for Purchase Orders</td>
<td>defaultpurchaseorderlocation</td>
</tr>
<tr>
<td>Default Location for Sales Orders</td>
<td>defaultsalesorderlocation</td>
</tr>
<tr>
<td>Default Items to Zero Received/Fulfilled</td>
<td>defaultunfulfilled</td>
</tr>
<tr>
<td>Default Return Auth. Status</td>
<td>defrtnauthstatus</td>
</tr>
<tr>
<td>Default Sales Order Status</td>
<td>defsalesordstatus</td>
</tr>
<tr>
<td>Default Vendor Return Auth. Status</td>
<td>defvendauthstatus</td>
</tr>
<tr>
<td>Include Committed Quantities on Drop Shipments/ Special Orders</td>
<td>dropshipincludecommitted</td>
</tr>
<tr>
<td>Drop Ship P.O. Form</td>
<td>dropshiptemplate</td>
</tr>
<tr>
<td>Send Email Confirmation when Sales Order Canceled</td>
<td>emailcancelorder</td>
</tr>
<tr>
<td>Automatically Email Drop Ship P.O.s</td>
<td>emaildropshippos</td>
</tr>
<tr>
<td>Automatically Fax Drop Ship P.O.s</td>
<td>faxdropshippos</td>
</tr>
<tr>
<td>Fulfill Based on Commitment</td>
<td>fulfillcommitted</td>
</tr>
<tr>
<td>Base Invoice Date on Billing Schedule Date</td>
<td>invoiceusescheduledate</td>
</tr>
<tr>
<td>Limit Vendor List on Items</td>
<td>limititemvendors</td>
</tr>
<tr>
<td>Name for Packed Status</td>
<td>namingpacked</td>
</tr>
<tr>
<td>Name for Picked Status</td>
<td>namingpicked</td>
</tr>
<tr>
<td>Name for Shipped Status</td>
<td>namingshipped</td>
</tr>
<tr>
<td>Send Order Fulfilled Confirmation Emails</td>
<td>ordfulfillconfemail</td>
</tr>
<tr>
<td>Use Web Site Template for Fulfillment Emails</td>
<td>ordfulfillusestoretemplates</td>
</tr>
<tr>
<td>Allow Overage on Assembly Builds</td>
<td>overbuilds</td>
</tr>
<tr>
<td>Allow Overage on Item Fulfillments</td>
<td>overfulfills</td>
</tr>
<tr>
<td>Allow Overage on Item Receipts</td>
<td>overreceipts</td>
</tr>
<tr>
<td>Show Drop Ship Items on Packing Slips</td>
<td>packingslipdropship</td>
</tr>
<tr>
<td>Limit Status on Packing Slip Queu</td>
<td>packingslipstatus</td>
</tr>
<tr>
<td>Always Print Kit Items on Picking Tickets</td>
<td>pickingticketkititems</td>
</tr>
<tr>
<td>Show Non-Inventory Items on Picking Tickets and Packing Slips</td>
<td>pickingticketnoninvt</td>
</tr>
<tr>
<td>Show Uncommitted Items on Picking Tickets</td>
<td>pickingticketuncommitted</td>
</tr>
<tr>
<td>Allow Expenses on Purchase Order</td>
<td>poexpenses</td>
</tr>
<tr>
<td>Queue Drop Ship P.O.s for Printing</td>
<td>printdropshippos</td>
</tr>
<tr>
<td>Require Re-approval on Edit of Sales Order</td>
<td>reapprovesoonedit</td>
</tr>
<tr>
<td>Restock Returned Items</td>
<td>restockreturns</td>
</tr>
</tbody>
</table>
Preference Names and IDs

<table>
<thead>
<tr>
<th>Preference UI Label</th>
<th>Preference Internal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show All Ordered Items on Packing Slips</td>
<td>showopenpackslip</td>
</tr>
<tr>
<td>Show Unfulfilled Items on Invoices</td>
<td>showunshippeditems</td>
</tr>
<tr>
<td>Bill in Advance of Receipt</td>
<td>unreceivedbills</td>
</tr>
<tr>
<td>Refund in Advance of Return</td>
<td>unreceivedrtnauths</td>
</tr>
<tr>
<td>Credit in Advance of Vendor Return</td>
<td>unreturnedvendauths</td>
</tr>
<tr>
<td>Invoice in Advance of Fulfillment</td>
<td>unshippedinvoices</td>
</tr>
<tr>
<td>Write-Off Account for Returns</td>
<td>writeoffaccount</td>
</tr>
</tbody>
</table>

**On the Time & Expenses tab**

<table>
<thead>
<tr>
<th>Preference UI Label</th>
<th>Preference Internal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatically Notify Supervisor</td>
<td>autonotifysupv</td>
</tr>
<tr>
<td>Combine Detail Items on Expense Reports</td>
<td>combineexpenseitems</td>
</tr>
<tr>
<td>Copy Expense Memos to Invoices</td>
<td>copyexpensememos</td>
</tr>
<tr>
<td>Copy Time Memos to Invoices</td>
<td>copytimememos</td>
</tr>
<tr>
<td>Expenses Billable by Default</td>
<td>defaultexpensebillable</td>
</tr>
<tr>
<td>Items Billable by Default</td>
<td>defaultitemsbillable</td>
</tr>
<tr>
<td>Time Billable by Default</td>
<td>defaulttimebillable</td>
</tr>
<tr>
<td>Override Rates on Time Records</td>
<td>overridetimerates</td>
</tr>
<tr>
<td>Show Planned Time in Time Entry</td>
<td>showplannedtime</td>
</tr>
<tr>
<td>Require Approvals on Time Records</td>
<td>timeapprovals</td>
</tr>
<tr>
<td>Show Jobs Only for Time and Expense Entry</td>
<td>timeexpensejobonly</td>
</tr>
</tbody>
</table>

**Accounting Periods**

These are the account preferences that can be found by going to Setup > Accounting > Manage Accounting Periods.

The internal ID for the Accounting Periods page is `accountingperiods`. 

<table>
<thead>
<tr>
<th>Preference UI Label</th>
<th>Preference Internal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Fiscal Month</td>
<td>fiscalmonth</td>
</tr>
<tr>
<td>Fiscal Year Ene</td>
<td>fiscalyear</td>
</tr>
<tr>
<td>Period Format</td>
<td>periodstyle</td>
</tr>
<tr>
<td>Year in Period Name</td>
<td>periodname</td>
</tr>
<tr>
<td>One-Day Year-End Adjustment Period</td>
<td>lastday</td>
</tr>
</tbody>
</table>
Tax Setup

These are the account preferences that can be found by going to Setup > Accounting > Setup Taxes. Note that field internal IDs are suffixed with the edition type. In the table below, fields are suffixed with `us`, for the US edition. Field IDs in your account may be suffixed with a different addition type. Also note that the sublists on this page are not supported in SuiteScript.

The internal ID for the Set Up Taxes page is `taxpreferences`.

<table>
<thead>
<tr>
<th>Preference UI Label</th>
<th>Preference Internal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaulttaxableus</td>
<td></td>
</tr>
<tr>
<td>defaultitemstaxableus</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td></td>
</tr>
<tr>
<td>chargoutofdistrictus</td>
<td></td>
</tr>
<tr>
<td>perlinetaxesus</td>
<td></td>
</tr>
<tr>
<td>storeordertaxationus</td>
<td></td>
</tr>
<tr>
<td>enabletaxlookups</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 76 Feature Names and IDs

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.

<table>
<thead>
<tr>
<th>Feature Internal ID</th>
<th>Feature Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNTING</td>
<td>Accounting</td>
</tr>
<tr>
<td>ACCOUNTINGPERIODS</td>
<td>Accounting Periods</td>
</tr>
<tr>
<td>ACHVEND</td>
<td>ACH Vendor Payments</td>
</tr>
<tr>
<td>ADVANCEDJOBS</td>
<td>Advanced Projects</td>
</tr>
<tr>
<td>ADVANCEDSITECUST</td>
<td>Advanced Site Customization</td>
</tr>
<tr>
<td>ADVBILLING</td>
<td>Advanced Billing</td>
</tr>
<tr>
<td>ADVFORECASTING</td>
<td>Advanced Forecasting</td>
</tr>
<tr>
<td>ADVINVENTORYMGMT</td>
<td>Advanced Inventory Management</td>
</tr>
<tr>
<td>ADVPARTNERACCESS</td>
<td>Advanced Partner Access</td>
</tr>
<tr>
<td>ADVRECEIVING</td>
<td>Advanced Receiving</td>
</tr>
<tr>
<td>ADVSHIPPING</td>
<td>Advanced Shipping</td>
</tr>
<tr>
<td>ADVTAXENGINE</td>
<td>Advanced Taxes</td>
</tr>
<tr>
<td>ADVWEBREPORTS</td>
<td>Advanced Web Reports</td>
</tr>
<tr>
<td>ADVWEBSEARCH</td>
<td>Advanced Web Search</td>
</tr>
<tr>
<td>ALTSALESAMOUNT</td>
<td>Alternate Sales Amount</td>
</tr>
<tr>
<td>AMORTIZATION</td>
<td>Amortization</td>
</tr>
<tr>
<td>APPROVALROUTING</td>
<td>Approval Routing</td>
</tr>
<tr>
<td>ASSEMBLIES</td>
<td>Assembly Items</td>
</tr>
<tr>
<td>BARCODES</td>
<td>Bar Coding and Item Labels</td>
</tr>
<tr>
<td>BILLINGCLASSES</td>
<td>Per-Employee Billing Rates</td>
</tr>
<tr>
<td>BILLPAY</td>
<td>Online Bill Pay</td>
</tr>
<tr>
<td>BILLSCOSTS</td>
<td>Bill Costs To Customers</td>
</tr>
<tr>
<td>BINMANAGEMENT</td>
<td>Bin Management</td>
</tr>
<tr>
<td>CAMPAIGNSUBSCRIPTIONS</td>
<td>Subscription Categories</td>
</tr>
<tr>
<td>CCTRACKING</td>
<td>Credit Card Payments</td>
</tr>
<tr>
<td>CLASSES</td>
<td>Classes</td>
</tr>
<tr>
<td>Feature Internal ID</td>
<td>Feature Name</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>COMMISSIONS</td>
<td>Employee Commissions</td>
</tr>
<tr>
<td>CREATESUITEBUNDLES</td>
<td>Create bundles with SuiteBundler?</td>
</tr>
<tr>
<td>CRM</td>
<td>Customer Relationship Management</td>
</tr>
<tr>
<td>CRMTIME</td>
<td>Time Tracking for CRM</td>
</tr>
<tr>
<td>CUSTOMCODE</td>
<td>Client SuiteScript</td>
</tr>
<tr>
<td>CUSTOMERACCESS</td>
<td>Customer Access</td>
</tr>
<tr>
<td>CUSTOMRECORDS</td>
<td>Custom Records</td>
</tr>
<tr>
<td>DEPARTMENTS</td>
<td>Departments</td>
</tr>
<tr>
<td>DIRECTDEPOSIT</td>
<td>Direct Deposit</td>
</tr>
<tr>
<td>DOCUMENTPUBLISHING</td>
<td>Document Publishing</td>
</tr>
<tr>
<td>DOCUMENTS</td>
<td>File Cabinet</td>
</tr>
<tr>
<td>DOWNLOADITEMS</td>
<td>Sell Downloadable Files</td>
</tr>
<tr>
<td>DROPSHIPMENTS</td>
<td>Drop Shipments &amp; Special Orders</td>
</tr>
<tr>
<td>DUPLICATES</td>
<td>Duplicate Detection &amp; Merge</td>
</tr>
<tr>
<td>EBAY</td>
<td>eBay Integration</td>
</tr>
<tr>
<td>EFT</td>
<td>Electronic Funds Transfer</td>
</tr>
<tr>
<td>EMAILINTEGRATION</td>
<td>Capture Email Replies</td>
</tr>
<tr>
<td>EMPPERMS</td>
<td>Global Permissions</td>
</tr>
<tr>
<td>ENTERPRISE</td>
<td>Enterprise Reporting Views</td>
</tr>
<tr>
<td>ESCALATIONRULES</td>
<td>Automated Case Escalation</td>
</tr>
<tr>
<td>ESTIMATES</td>
<td>Estimates</td>
</tr>
<tr>
<td>EXPENSEALLOCATION</td>
<td>Expense Allocation</td>
</tr>
<tr>
<td>EXPREPORTS</td>
<td>Expense Reports</td>
</tr>
<tr>
<td>EXTCRM</td>
<td>Online Forms</td>
</tr>
<tr>
<td>EXTREMELIST</td>
<td>Direct List Editing</td>
</tr>
<tr>
<td>EXTSTORE</td>
<td>External Catalog Site (WSDK)</td>
</tr>
<tr>
<td>FXRATEUPDATES</td>
<td>Currency Exchange Rate Integration</td>
</tr>
<tr>
<td>GCO</td>
<td>Google Checkout</td>
</tr>
<tr>
<td>GIFTCERTIFICATES</td>
<td>Gift Certificates</td>
</tr>
<tr>
<td>GROSSPROFIT</td>
<td>Gross Profit</td>
</tr>
<tr>
<td>HELPDESK</td>
<td>Help Desk</td>
</tr>
<tr>
<td>HISTORICALMETRICS</td>
<td>Historical Metrics</td>
</tr>
<tr>
<td>INBOUNDCASEEMAIL</td>
<td>Email Case Capture</td>
</tr>
<tr>
<td>INTRANET</td>
<td>Intranet</td>
</tr>
<tr>
<td>INVENTORY</td>
<td>Inventory</td>
</tr>
<tr>
<td>IPADDRESSRULES</td>
<td>IP Address Rules</td>
</tr>
<tr>
<td>Feature Internal ID</td>
<td>Feature Name</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>ISSUEDB</td>
<td>Issue Management</td>
</tr>
<tr>
<td>ITEMOPTIONS</td>
<td>Item Options</td>
</tr>
<tr>
<td>JOBS</td>
<td>Projects</td>
</tr>
<tr>
<td>KNOWLEDGEBASE</td>
<td>Knowledge Base</td>
</tr>
<tr>
<td>KPIREPORTS</td>
<td>KPI Scorecards</td>
</tr>
<tr>
<td>LANDEDCOST</td>
<td>Landed Cost</td>
</tr>
<tr>
<td>LOCATIONS</td>
<td>Locations</td>
</tr>
<tr>
<td>LOTNUMBEREDINVENTORY</td>
<td>Lot Tracking</td>
</tr>
<tr>
<td>MAILMERGE</td>
<td>Mail Merge</td>
</tr>
<tr>
<td>MARKETING</td>
<td>Marketing Automation</td>
</tr>
<tr>
<td>MATRIXITEMS</td>
<td>Matrix Items</td>
</tr>
<tr>
<td>MULTICURRENCY</td>
<td>Multiple Currencies</td>
</tr>
<tr>
<td>MULTILANGUAGE</td>
<td>Multi-Language</td>
</tr>
<tr>
<td>MULTILOCINVT</td>
<td>Multi-Location Inventory</td>
</tr>
<tr>
<td>MULTIPARTNER</td>
<td>Multi-Partner Management</td>
</tr>
<tr>
<td>MULTIPLEBUDGETS</td>
<td>Multiple Budgets</td>
</tr>
<tr>
<td>MULTISHIPTO</td>
<td>Multiple Shipping Routes</td>
</tr>
<tr>
<td>MULTISITE</td>
<td>Multiple Web Sites</td>
</tr>
<tr>
<td>MULTIVENDOR</td>
<td>Multiple Vendors</td>
</tr>
<tr>
<td>MULTPRICE</td>
<td>Multiple Prices</td>
</tr>
<tr>
<td>ODBC</td>
<td>ODBC Connections for Advanced Reporting</td>
</tr>
<tr>
<td>OFFLINECLIENT</td>
<td>Offline Client for Sales</td>
</tr>
<tr>
<td>ONLINEORDERING</td>
<td>Online Ordering</td>
</tr>
<tr>
<td>OPPORTUNITIES</td>
<td>Opportunities</td>
</tr>
<tr>
<td>OUTLOOKINTEGRATION</td>
<td>Outlook Integration</td>
</tr>
<tr>
<td>PARTNERACCESS</td>
<td>Partner Access</td>
</tr>
<tr>
<td>PARTNERCOMMISSIONS</td>
<td>Partner Commissions/Royalties</td>
</tr>
<tr>
<td>PAYABLES</td>
<td>A/P</td>
</tr>
<tr>
<td>PAYPALINTEGRATION</td>
<td>PayPal Integration</td>
</tr>
<tr>
<td>PAYROLL</td>
<td>Payroll</td>
</tr>
<tr>
<td>PAYROLLSERVICE</td>
<td>Payroll Service</td>
</tr>
<tr>
<td>PICKPACKSHIP</td>
<td>Pick, Pack and Ship</td>
</tr>
<tr>
<td>PRM</td>
<td>Partner Relationship Management</td>
</tr>
<tr>
<td>PROMOCODES</td>
<td>Promotion Codes</td>
</tr>
<tr>
<td>PURCHASEORDERS</td>
<td>Purchase Orders</td>
</tr>
<tr>
<td>PURCHASEREQS</td>
<td>Purchase Requests</td>
</tr>
<tr>
<td>Feature Internal ID</td>
<td>Feature Name</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>QUANTITYPRICING</td>
<td>Quantity Pricing</td>
</tr>
<tr>
<td>RECEIVABLES</td>
<td>A/R</td>
</tr>
<tr>
<td>RETURNAUTHS</td>
<td>Return Authorizations</td>
</tr>
<tr>
<td>REVENUECOMMITMENTS</td>
<td>Revenue Commitments</td>
</tr>
<tr>
<td>REVENUERECOGNITION</td>
<td>Revenue Recognition</td>
</tr>
<tr>
<td>REVRECSALESORDERFORECASTING</td>
<td>Sales Order Revenue Forecasting</td>
</tr>
<tr>
<td>REVRECVSOE</td>
<td>VSOE</td>
</tr>
<tr>
<td>SALESCAMPAIGNS</td>
<td>Sales Campaigns</td>
</tr>
<tr>
<td>SALESCAIRNERS</td>
<td>Sales Orders</td>
</tr>
<tr>
<td>SERIALIZEDINVENTORY</td>
<td>Serialized Inventory</td>
</tr>
<tr>
<td>SERVERSIDESCRIPTING</td>
<td>Server SuiteScript</td>
</tr>
<tr>
<td>SERVICEPRINTEDCHECKS</td>
<td>Service Printed Checks and Stubs</td>
</tr>
<tr>
<td>SERVICEPRINTEDW2S?</td>
<td>Service Printed W-2s and 1099s</td>
</tr>
<tr>
<td>SFA</td>
<td>Sales Force Automation</td>
</tr>
<tr>
<td>SHIPPINGLABELS</td>
<td>Shipping Label Integration</td>
</tr>
<tr>
<td>SITELOCATIONALIASSES</td>
<td>Descriptive URLs</td>
</tr>
<tr>
<td>SOFTDESCRIPTORS</td>
<td>Credit Card Soft Descriptors</td>
</tr>
<tr>
<td>SUBSIDIARIES</td>
<td>Subsidiaries</td>
</tr>
<tr>
<td>SUPPORT</td>
<td>Customer Support and Service</td>
</tr>
<tr>
<td>SYNCHRONIZATION</td>
<td>Synchronization</td>
</tr>
<tr>
<td>TEAMSELLING</td>
<td>Team Selling</td>
</tr>
<tr>
<td>TELEPHONY</td>
<td>Telephony Integration</td>
</tr>
<tr>
<td>TIMETRACKING</td>
<td>Time Tracking</td>
</tr>
<tr>
<td>UNITSOFMEASURE</td>
<td>Multiple Units of Measure</td>
</tr>
<tr>
<td>UPSELL</td>
<td>Upsell Manager</td>
</tr>
<tr>
<td>VENDORACCESS</td>
<td>Vendor Access</td>
</tr>
<tr>
<td>VENDORRETURNAUTHS</td>
<td>Vendor Return Authorizations</td>
</tr>
<tr>
<td>WEBSHOSTING</td>
<td>Host HTML Files</td>
</tr>
<tr>
<td>WEBSERVICESEXTERNAL</td>
<td>Web Services</td>
</tr>
<tr>
<td>WEBSITE</td>
<td>Web Site</td>
</tr>
<tr>
<td>WEBSTORE</td>
<td>Web Store</td>
</tr>
<tr>
<td>WORKORDERS</td>
<td>Work Orders</td>
</tr>
</tbody>
</table>
Chapter 77 Supported File Types

This section provides a list of all file 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
Hello World');
```

**Important:** Be aware that the `nlapiCreateFile` function does not support the creation of non-text file types.

<table>
<thead>
<tr>
<th>File Type ID</th>
<th>Name</th>
<th>Extension</th>
<th>Content Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOCAD</td>
<td>AutoCad</td>
<td>.dwg</td>
<td>application/x-autocad</td>
</tr>
<tr>
<td>BMPIMAGE</td>
<td>BMP Image</td>
<td>.bmp</td>
<td>image/x-xbitmap</td>
</tr>
<tr>
<td>CSV</td>
<td>CSV File</td>
<td>.csv</td>
<td>text/csv</td>
</tr>
<tr>
<td>EXCEL</td>
<td>Excel File</td>
<td>.xls</td>
<td>application/vnd.ms-excel</td>
</tr>
<tr>
<td>FLASH</td>
<td>Flash Animation</td>
<td>.swf</td>
<td>application/x-shockwave-flash</td>
</tr>
<tr>
<td>GIFIMAGE</td>
<td>GIF Image</td>
<td>.gif</td>
<td>image/gif</td>
</tr>
<tr>
<td>GZIP</td>
<td>GNU Zip File</td>
<td>.gz</td>
<td>application/x-gzip-compressed</td>
</tr>
<tr>
<td>HTMLDOC</td>
<td>HTML File</td>
<td>.htm</td>
<td>text/html</td>
</tr>
<tr>
<td>ICON</td>
<td>Icon Image</td>
<td>.ico</td>
<td>image/ico</td>
</tr>
<tr>
<td>JAVASCRIPT</td>
<td>JavaScript File</td>
<td>.js</td>
<td>text/javascript</td>
</tr>
<tr>
<td>JPGIMAGE</td>
<td>JPEG Image</td>
<td>.jpg</td>
<td>image/jpeg</td>
</tr>
<tr>
<td>MESSAGERFC</td>
<td>Message RFC</td>
<td>.eml</td>
<td>message/rfc822</td>
</tr>
<tr>
<td>MP3</td>
<td>MP3 Audio</td>
<td>.mp3</td>
<td>audio/mpeg</td>
</tr>
<tr>
<td>MPEGMOVIE</td>
<td>MPEG Video</td>
<td>.mpg</td>
<td>video/mpeg</td>
</tr>
<tr>
<td>MSPROJECT</td>
<td>Project File</td>
<td>.mpp</td>
<td>application/vnd.ms-project</td>
</tr>
<tr>
<td>PDF</td>
<td>PDF File</td>
<td>.pdf</td>
<td>application/pdf</td>
</tr>
<tr>
<td>PJPGIMAGE</td>
<td>PJPEG Image</td>
<td>.pjpeg</td>
<td>image/pjpeg</td>
</tr>
<tr>
<td>File Type ID</td>
<td>Name</td>
<td>Extension</td>
<td>Content Type</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------</td>
<td>-----------</td>
<td>--------------------</td>
</tr>
<tr>
<td>PLAINTEXT</td>
<td>Plain Text File</td>
<td>.txt</td>
<td>text/plain</td>
</tr>
<tr>
<td>PNGIMAGE</td>
<td>PNG Image</td>
<td>.png</td>
<td>image/x-png</td>
</tr>
<tr>
<td>POSTSCRIPT</td>
<td>PostScript File</td>
<td>.ps</td>
<td>application/postscript</td>
</tr>
<tr>
<td>POWERPOINT</td>
<td>PowerPoint File</td>
<td>.ppt</td>
<td>application/vnd.ms-powerpoint</td>
</tr>
<tr>
<td>QUICKTIME</td>
<td>QuickTime Video</td>
<td>.mov</td>
<td>video/quicktime</td>
</tr>
<tr>
<td>RTF</td>
<td>RTF File</td>
<td>.rtf</td>
<td>application/rtf</td>
</tr>
<tr>
<td>SMS</td>
<td>SMS File</td>
<td>.sms</td>
<td>application/sms</td>
</tr>
<tr>
<td>STYLESHEET</td>
<td>CSS File</td>
<td>.css</td>
<td>text/css</td>
</tr>
<tr>
<td>TIFFIMAGE</td>
<td>TIFF Image</td>
<td>.tiff</td>
<td>image/tiff</td>
</tr>
<tr>
<td>VISIO</td>
<td>Visio File</td>
<td>.vsd</td>
<td>application/vnd.visio</td>
</tr>
<tr>
<td>WORD</td>
<td>Word File</td>
<td>.doc</td>
<td>application/msword</td>
</tr>
<tr>
<td>XMLDOC</td>
<td>XML File</td>
<td>.xml</td>
<td>text/xml</td>
</tr>
<tr>
<td>ZIP</td>
<td>Zip File</td>
<td>.zip</td>
<td>application/zip</td>
</tr>
</tbody>
</table>
Chapter 78 Button IDs

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 nlobjButton.setLabel(label) and nlobjButton.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.

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 Configuring Buttons and Actions for details.

<table>
<thead>
<tr>
<th>Button UI Label</th>
<th>Button Internal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Items</td>
<td>addmatrix</td>
</tr>
<tr>
<td>Accept</td>
<td>accept</td>
</tr>
<tr>
<td>Accept Payment</td>
<td>acceptpayment</td>
</tr>
<tr>
<td>Apply</td>
<td>apply</td>
</tr>
<tr>
<td>Approve</td>
<td>approve</td>
</tr>
<tr>
<td>Approve Return</td>
<td>approvereturn</td>
</tr>
<tr>
<td>Authorize Return</td>
<td>return</td>
</tr>
<tr>
<td>Auto Fill</td>
<td>autofill</td>
</tr>
<tr>
<td>Bill</td>
<td>bill</td>
</tr>
<tr>
<td>Bill Remaining</td>
<td>billremaining</td>
</tr>
<tr>
<td>Cancel Order</td>
<td>cancelorder</td>
</tr>
<tr>
<td>Cancel Return</td>
<td>cancelreturn</td>
</tr>
<tr>
<td>Clear Splits</td>
<td>clearsplits</td>
</tr>
<tr>
<td>Close</td>
<td>closeremaining</td>
</tr>
<tr>
<td>Convert</td>
<td>convertlead</td>
</tr>
<tr>
<td>Convert to Inventory</td>
<td>convertinvt</td>
</tr>
<tr>
<td>Convert to Lot Numbered Inventory</td>
<td>convertlot</td>
</tr>
<tr>
<td>Button UI Label</td>
<td>Button Internal ID</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Convert to Serialized Inventory</td>
<td>converterserial</td>
</tr>
<tr>
<td>Create Build</td>
<td>createbuild</td>
</tr>
<tr>
<td>Create Matrix</td>
<td>creatematrix</td>
</tr>
<tr>
<td>Credit</td>
<td>credit</td>
</tr>
<tr>
<td>Decline</td>
<td>decline</td>
</tr>
<tr>
<td>Delete</td>
<td>delete</td>
</tr>
<tr>
<td>Email</td>
<td>email</td>
</tr>
<tr>
<td>Fax</td>
<td>fax</td>
</tr>
<tr>
<td>Fulfill</td>
<td>process</td>
</tr>
<tr>
<td>Generate Price List</td>
<td>generateprice</td>
</tr>
<tr>
<td>Generate Statement</td>
<td>generatestate</td>
</tr>
<tr>
<td>GL Impact</td>
<td>glimpact</td>
</tr>
<tr>
<td>Go To Register</td>
<td>gotoregister</td>
</tr>
<tr>
<td>Grab</td>
<td>grab</td>
</tr>
<tr>
<td>Make Copy</td>
<td>makecopy</td>
</tr>
<tr>
<td>Make Payment</td>
<td>payment</td>
</tr>
<tr>
<td>Make Standalone Copy</td>
<td>makestandalone</td>
</tr>
<tr>
<td>Memorize</td>
<td>memorize</td>
</tr>
<tr>
<td>Merge</td>
<td>merge</td>
</tr>
<tr>
<td>New</td>
<td>new</td>
</tr>
<tr>
<td>New Event Field</td>
<td>neweventfield</td>
</tr>
<tr>
<td>Next Bill</td>
<td>nextbill</td>
</tr>
<tr>
<td>Next Week</td>
<td>next</td>
</tr>
<tr>
<td>Prev Week</td>
<td>prev</td>
</tr>
<tr>
<td>Print</td>
<td>print</td>
</tr>
<tr>
<td>Print Bill of Materials</td>
<td>printbom</td>
</tr>
<tr>
<td>Print Label</td>
<td>printlabel</td>
</tr>
<tr>
<td>Print Labels</td>
<td>printlabels</td>
</tr>
<tr>
<td>Print Picking Ticket</td>
<td>printpicktick</td>
</tr>
<tr>
<td>Print Summary</td>
<td>depositsummary</td>
</tr>
<tr>
<td>Quick Accept</td>
<td>quickaccept</td>
</tr>
<tr>
<td>Recalc</td>
<td>recal</td>
</tr>
<tr>
<td>Receive</td>
<td>receive</td>
</tr>
<tr>
<td>Refund</td>
<td>refund</td>
</tr>
<tr>
<td>Reject</td>
<td>reject</td>
</tr>
<tr>
<td>Renew</td>
<td>renewal</td>
</tr>
<tr>
<td>Button UI Label</td>
<td>Button Internal ID</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Reset</td>
<td>resetter</td>
</tr>
<tr>
<td>Revenue Commitment Reversal</td>
<td>revcomrv</td>
</tr>
<tr>
<td>Save As</td>
<td>submitas</td>
</tr>
<tr>
<td>Save &amp; Bill</td>
<td>submitbill</td>
</tr>
<tr>
<td>Save &amp; Convert</td>
<td>submitconvert</td>
</tr>
<tr>
<td>Save &amp; Copy</td>
<td>submitcopy</td>
</tr>
<tr>
<td>Save &amp; Edit</td>
<td>submitedit</td>
</tr>
<tr>
<td>Save &amp; Email</td>
<td>saveemail</td>
</tr>
<tr>
<td>Save &amp; Fulfill</td>
<td>submitfulfill</td>
</tr>
<tr>
<td>Save &amp; New</td>
<td>submitnew</td>
</tr>
<tr>
<td>Save &amp; Next</td>
<td>submitnext</td>
</tr>
<tr>
<td>Save &amp; Print</td>
<td>saveprint</td>
</tr>
<tr>
<td>Save &amp; Print BOM</td>
<td>saveprintbom</td>
</tr>
<tr>
<td>Save &amp; Print Label</td>
<td>saveandprintlabel</td>
</tr>
<tr>
<td>Save &amp; Refund</td>
<td>submitrefund</td>
</tr>
<tr>
<td>Save &amp; Same</td>
<td>submitsame</td>
</tr>
<tr>
<td>Save Baseline</td>
<td>savebaseline</td>
</tr>
<tr>
<td>Search</td>
<td>search</td>
</tr>
<tr>
<td>Show Activity</td>
<td>showactivity</td>
</tr>
<tr>
<td>Submit Invoice</td>
<td>submitinvoice</td>
</tr>
<tr>
<td>Tentative</td>
<td>tentative</td>
</tr>
<tr>
<td>Unbuild</td>
<td>createunbuild</td>
</tr>
<tr>
<td>Update Matrix</td>
<td>updatematrix</td>
</tr>
<tr>
<td>Update VSOE</td>
<td>updatevsoe</td>
</tr>
<tr>
<td>View All Transactions</td>
<td>viewalltransactions</td>
</tr>
<tr>
<td>Void</td>
<td>void</td>
</tr>
<tr>
<td>W4 Worksheet</td>
<td>w4data</td>
</tr>
</tbody>
</table>

Related Topics

- Configuring Buttons and Actions
- nlobjButton
Chapter 79 Supported Tasklinks

The following table lists NetSuite tasklinks that can be referenced when using `nlapiResolveURL(type, identifier, id, displayMode)` or `nlapiSetRedirectURL(type, identifier, id, editmode, parameters)`.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Page Label in NetSuite</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDIT_ACCOUNT</td>
<td>New Accounts</td>
<td>/app/accounting/account/account.nl</td>
</tr>
<tr>
<td>EDIT_ACCOUNTINGOTHERLIST</td>
<td>New Accounting List Element</td>
<td>/app/common/otherlists/accountingotherlist.nl</td>
</tr>
<tr>
<td>EDIT_ACTIVITY</td>
<td>New Activity</td>
<td>/app/crm/calendar/activity.nl</td>
</tr>
<tr>
<td>EDIT_ALLOCATION</td>
<td>Create Allocation Schedules</td>
<td>/app/accounting/transactions/allocation.nl</td>
</tr>
<tr>
<td>EDIT_AMENDW4</td>
<td>Form W-4</td>
<td>/app/common/entity/amendw4.nl</td>
</tr>
<tr>
<td>EDIT_AMORTIZATIONSCHE</td>
<td>New Amortization Template</td>
<td>/app/accounting/otherlists/revrecschedule.nl?type=Amortization</td>
</tr>
<tr>
<td>EDIT_BILLINGSCHEDULE</td>
<td>New Billing Schedule</td>
<td>/app/accounting/otherlists/billingschedule.nl</td>
</tr>
<tr>
<td>EDIT_BINNUMBERRECORD</td>
<td>New Bin</td>
<td>/app/accounting/transactions/inventory/binnumberrecord.nl</td>
</tr>
<tr>
<td>EDIT_BULKOP</td>
<td>Edit Mass Update</td>
<td>/app/common/bulk/bulkop.nl</td>
</tr>
<tr>
<td>EDIT_BUNDLE</td>
<td>Create Bundle</td>
<td>/app/setup/assistants/bundlebuilder.nl?new=T</td>
</tr>
<tr>
<td>EDIT_CALENDARPREference</td>
<td>Calendar Preference</td>
<td>/app/crm/calendar/calendarpreference.nl</td>
</tr>
<tr>
<td>EDIT_CALL</td>
<td>New Phone Call</td>
<td>/app/crm/calendar/call.nl</td>
</tr>
<tr>
<td>EDIT_CAMPAIGN</td>
<td>New Marketing Campaign</td>
<td>/app/crm/marketing/campaign.nl</td>
</tr>
<tr>
<td>EDIT_CAMPAIGNAUDIENCE</td>
<td>New Campaign Audience</td>
<td>/app/crm/marketing/campaignaudience.nl</td>
</tr>
<tr>
<td>EDIT_CAMPAIGNBULK</td>
<td>Create Keyword Campaigns</td>
<td>/app/crm/marketing/campaign.nl?bulk=T</td>
</tr>
<tr>
<td>EDIT_CAMPAIGNBULKIMPORT</td>
<td>Import Keywords</td>
<td>/app/setup/assistants/nsimport/simpleimport.nl?rectype=CAMPAIGNKEYWORD</td>
</tr>
<tr>
<td>EDIT_CAMPAIGNCATEGORY</td>
<td>New Campaign Category</td>
<td>/app/crm/marketing/campaigncategory.nl</td>
</tr>
<tr>
<td>EDIT_CAMPAIGNCHANNEL</td>
<td>New Campaign Channel</td>
<td>/app/crm/marketing/campaignchannel.nl</td>
</tr>
<tr>
<td>EDIT_CAMPAIGNEMAIL</td>
<td>New Campaign Email Address</td>
<td>/app/crm/marketing/campaignemail.nl</td>
</tr>
<tr>
<td>EDIT_CAMPAIGNFAMILY</td>
<td>New Campaign Family</td>
<td>/app/crm/marketing/campaignfamily.nl</td>
</tr>
<tr>
<td>EDIT_CAMPAIGNOFFER</td>
<td>New Campaign Offer</td>
<td>/app/crm/marketing/campaignoffer.nl</td>
</tr>
<tr>
<td>EDIT_CAMPAIGNSEARCHENGINE</td>
<td>New Campaign Search Engine</td>
<td>/app/crm/marketing/campaignsearchengine.nl</td>
</tr>
<tr>
<td>EDIT_CAMPAIGNSUBSCRIPTION</td>
<td>New Campaign Subscription</td>
<td>/app/crm/marketing/campaignsubscription.nl</td>
</tr>
<tr>
<td>EDIT_CAMPAIGNVERTICAL</td>
<td>New Campaign Vertical</td>
<td>/app/crm/marketing/campaignvertical.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>EDIT_CASEFIELDRULE</td>
<td>New Case Rule</td>
<td>/app/crm/support/casefieldrule.nl</td>
</tr>
<tr>
<td>EDIT_CASEFORM</td>
<td>New Online Case Forms</td>
<td>/app/crm/support/caseform.nl</td>
</tr>
<tr>
<td>EDIT_CASEISSUE</td>
<td>New Case Issue</td>
<td>/app/crm/support/caseissue.nl</td>
</tr>
<tr>
<td>EDIT_CASEORIGIN</td>
<td>New Case Origin Type</td>
<td>/app/crm/support/caseorigin.nl</td>
</tr>
<tr>
<td>EDIT_CASEPRIORITY</td>
<td>New Case Priority</td>
<td>/app/crm/support/casepriority.nl</td>
</tr>
<tr>
<td>EDIT_CASESTATUS</td>
<td>New Case Status</td>
<td>/app/crm/support/casestatus.nl</td>
</tr>
<tr>
<td>EDIT_CASETERRITORY</td>
<td>New Case Territory</td>
<td>/app/crm/support/supportterritory.nl</td>
</tr>
<tr>
<td>EDIT_CASETYPE</td>
<td>New Case Type</td>
<td>/app/crm/support/casetype.nl</td>
</tr>
<tr>
<td>EDIT_CLASS</td>
<td>New Class</td>
<td>/app/common/otherlists/classtype.nl</td>
</tr>
<tr>
<td>EDIT_COLORTHEME</td>
<td>New Color Theme</td>
<td>/app/setup/look/colortheme.nl</td>
</tr>
<tr>
<td>EDIT_COMMISSIONSCHEDULE</td>
<td>New Employee Schedule</td>
<td>/app/crm/sales/commissions/commissionschedule.nl</td>
</tr>
<tr>
<td>EDIT_COMPETITOR</td>
<td>Competitor</td>
<td>/app/crm/sales/competitor.nl</td>
</tr>
<tr>
<td>EDIT_CONTACT</td>
<td>New Contacts</td>
<td>/app/common/entity/contact.nl</td>
</tr>
<tr>
<td>EDIT_CRMGROUP</td>
<td>New Groups</td>
<td>/app/crm/common/crmgroup.nl</td>
</tr>
<tr>
<td>EDIT_CRMMESSAGE</td>
<td>New Email</td>
<td>/app/crm/common/crmmessage.nl</td>
</tr>
<tr>
<td>EDIT_CRMOTHERLIST</td>
<td>New CRM List Element</td>
<td>/app/common/otherlists/crmotherlist.nl</td>
</tr>
<tr>
<td>EDIT_CRMTEMPLATE</td>
<td>New Marketing Templates</td>
<td>/app/crm/common/merge/marketingtemplate.nl</td>
</tr>
<tr>
<td>EDIT_CURRENCY</td>
<td>New Currencies</td>
<td>/app/common/multicurrency/currency.nl</td>
</tr>
<tr>
<td>EDIT_CURRENCYRATE</td>
<td>New Currency Exchange Rate</td>
<td>/app/common/multicurrency/currencerate.nl</td>
</tr>
<tr>
<td>EDIT_CUSTADDRESSFORM</td>
<td>Address Form</td>
<td>/app/common/custom/custaddressform.nl?e=T</td>
</tr>
<tr>
<td>EDIT_CUSTBODYFIELD</td>
<td>New Transaction Body Fields</td>
<td>/app/common/custom/custbodyfield.nl</td>
</tr>
<tr>
<td>EDIT_CUSTCATEGORY</td>
<td>New Center Category</td>
<td>/app/common/custom/custcategory.nl</td>
</tr>
<tr>
<td>EDIT_CUSTCENTER</td>
<td>New Center</td>
<td>/app/common/custom/custcenter.nl</td>
</tr>
<tr>
<td>EDIT_CUSTCOLUMNFIELD</td>
<td>New Transaction Column Fields</td>
<td>/app/common/custom/custcolumnfield.nl</td>
</tr>
<tr>
<td>EDIT_CUSTEMAILLAYOUT</td>
<td>New Transaction Form HTML Layouts</td>
<td>/app/common/custom/custemaillayout.nl</td>
</tr>
<tr>
<td>EDIT_CUSTENTITYFIELD</td>
<td>New Entity Fields</td>
<td>/app/common/custom/entitycustfield.nl</td>
</tr>
<tr>
<td>EDIT_CUSTENTRYFORM</td>
<td>New Entry Forms</td>
<td>/app/common/custom/custentryform.nl</td>
</tr>
<tr>
<td>EDIT_CUSTEVENTFIELD</td>
<td>New CRM Fields</td>
<td>/app/common/custom/eventcustfield.nl</td>
</tr>
<tr>
<td>EDIT_CUSTFORM</td>
<td>New Transaction Forms</td>
<td>/app/common/custom/custform.nl</td>
</tr>
<tr>
<td>EDIT_CUSTITEMFIELD</td>
<td>New Item Fields</td>
<td>/app/common/custom/itemcustfield.nl</td>
</tr>
<tr>
<td>EDIT_CUSTITEMNUMBERFIELD</td>
<td>New Item Number Field</td>
<td>/app/common/custom/itemnumbercustfield.nl</td>
</tr>
<tr>
<td>EDIT_CUSTJOB</td>
<td>New Customers</td>
<td>/app/common/entity/custjob.nl</td>
</tr>
<tr>
<td>EDIT_CUSTLAYOUT</td>
<td>New Transaction Form PDF Layouts</td>
<td>/app/common/custom/custlayout.nl</td>
</tr>
<tr>
<td>EDIT_CUSTLIST</td>
<td>New Lists</td>
<td>/app/common/custom/custlist.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>EDIT_CUSTOMERFIELDRULE</td>
<td>New Set Up Sales Rules</td>
<td>/app/crm/sales/customerfieldrule.nl</td>
</tr>
<tr>
<td>EDIT_CUSTOMERFORM</td>
<td>New Online Customer Forms</td>
<td>/app/crm/sales/leadform.nl</td>
</tr>
<tr>
<td>EDIT_CUSTOMERSTATUS</td>
<td>New Customer Statuses</td>
<td>/app/crm/sales/customerstatus.nl</td>
</tr>
<tr>
<td>EDIT_CUSTOMOTHERFIELD</td>
<td>New Other Field</td>
<td>/app/common/custom/othercustfield.nl</td>
</tr>
<tr>
<td>EDIT_CUSTPROFILE</td>
<td>Customer Profile</td>
<td>/app/common/entity/custprofile.nl?category=billing&amp;sc=4</td>
</tr>
<tr>
<td>EDIT_CUSTRECORD</td>
<td>New Record Types</td>
<td>/app/common/entity/custrecord.nl</td>
</tr>
<tr>
<td>EDIT_CUSTRECORDFIELD</td>
<td>Custom Record Field</td>
<td>/app/common/custom/custreccustfield.nl</td>
</tr>
<tr>
<td>EDIT_CUSTRECORDFORM</td>
<td>New Custom Record Form</td>
<td>/app/common/custom/custrecordform.nl</td>
</tr>
<tr>
<td>EDIT_CUSTSCRIPTFIELD</td>
<td>New Script Fields</td>
<td>/app/common/custom/scriptcustfield.nl</td>
</tr>
<tr>
<td>EDIT_CUSTSECTION</td>
<td>New Center Tab</td>
<td>/app/common/custom/custsection.nl</td>
</tr>
<tr>
<td>EDIT_CUSTTASKS</td>
<td>Center Links</td>
<td>/app/common/custom/custtasks.nl</td>
</tr>
<tr>
<td>EDIT_CUST</td>
<td>Custom Record Entry</td>
<td>/app/common/custom/custcordentry.nl</td>
</tr>
<tr>
<td>EDIT_DEPARTMENT</td>
<td>New Departments</td>
<td>/app/common/otherlists/departmenttype.nl</td>
</tr>
<tr>
<td>EDIT_EDITPROFILE</td>
<td>Employee Profile</td>
<td>/app/common/entity/editprofile.nl</td>
</tr>
<tr>
<td>EDIT_EMAILTEMPLATE</td>
<td>New Email Templates</td>
<td>/app/common/merge/emailtemplate.nl</td>
</tr>
<tr>
<td>EDITEMPLCATEGORY</td>
<td>New Employee Directory</td>
<td>/app/site/setup/emplcategory.nl</td>
</tr>
<tr>
<td>EDIT_EMPLOYEE</td>
<td>New Employees</td>
<td>/app/common/entity/employee.nl</td>
</tr>
<tr>
<td>EDIT_EMPLOYEESFA</td>
<td>New Assign Support Reps</td>
<td>/app/common/entity/employeesfa.nl</td>
</tr>
<tr>
<td>EDIT_EMPOTHERLIST</td>
<td>New Employee Related List Element</td>
<td>/app/common/otherlists/empotherlist.nl</td>
</tr>
<tr>
<td>EDIT_ESCALATIONRULE</td>
<td>New Escalation Rules</td>
<td>/app/crm/support/escalationfieldrule.nl</td>
</tr>
<tr>
<td>EDIT_ESCALATIONTERRITORY</td>
<td>New Manage Escalation Assignment</td>
<td>/app/crm/support/escalationterritory.nl</td>
</tr>
<tr>
<td>EDIT_EVENT</td>
<td>New Event</td>
<td>/app/crm/calendar/event.nl</td>
</tr>
<tr>
<td>EDIT_EXPCATEGORY</td>
<td>New Expense Categories</td>
<td>/app/accounting/otherlists/expcategory.nl</td>
</tr>
<tr>
<td>EDIT_FAXMESSAGE</td>
<td>New Fax Message</td>
<td>/app/crm/common/merge/faxmessage.nl</td>
</tr>
<tr>
<td>EDIT_FAXTEMPLATE</td>
<td>New Fax Templates</td>
<td>/app/crm/common/merge/faxtemplate.nl</td>
</tr>
<tr>
<td>EDIT_FISCALPERIOD</td>
<td>Setup Fiscal Period</td>
<td>/app/setup/period/fiscalperiod.nl</td>
</tr>
<tr>
<td>EDIT_GCOACCOUNT</td>
<td>Google Checkout Accounts</td>
<td>/app/site/setup/google/gcoaccount.nl</td>
</tr>
<tr>
<td>EDIT_IC_ALLOCATION</td>
<td>Create Intercompany Allocation Schedules</td>
<td>/app/accounting/transactions/intercompanyallocation.nl</td>
</tr>
<tr>
<td>EDIT_INFOITEM</td>
<td>New Information Items</td>
<td>/app/site/setup/infoitem.nl?Information_TYPE=TEXT</td>
</tr>
<tr>
<td>EDIT_INFOITEMFORM</td>
<td>New Publish Forms</td>
<td>/app/site/setup/infoitem.nl?Information_TYPE=FORM</td>
</tr>
<tr>
<td>EDIT_INSTALLBUNDLE</td>
<td>Install Bundle</td>
<td>/app/bundler/installbundle.nl</td>
</tr>
<tr>
<td>EDIT_ISSUE</td>
<td>New Issue</td>
<td>/app/crm/support/issuedb/issue.nl</td>
</tr>
<tr>
<td>EDIT_ISSUEPRODUCT</td>
<td>New Product</td>
<td>/app/crm/support/issuedb/issueproduct.nl</td>
</tr>
<tr>
<td>EDIT_ISSUETAG</td>
<td>New Issue Tag</td>
<td>/app/crm/support/issuedb/issuetag.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>EDIT_ISSUEUSERTYPE</td>
<td>New Issue Role</td>
<td>/app/crm/support/issuedb/issueusertype.nl</td>
</tr>
<tr>
<td>EDIT_ITEM</td>
<td>New Items</td>
<td>/app/common/item/item.nl</td>
</tr>
<tr>
<td>EDIT_ITEMOPTION</td>
<td>New Transaction Item Options</td>
<td>/app/common/custom/itemoption.nl</td>
</tr>
<tr>
<td>EDIT_JOB</td>
<td>New Job</td>
<td>/app/common/entity/custjob.nl?job=T</td>
</tr>
<tr>
<td>EDIT_KBCATEGORY</td>
<td>New Knowledge Base</td>
<td>/app/site/setup/kbcategory.nl</td>
</tr>
<tr>
<td>EDIT_KPIREPORT</td>
<td>New KPI Scorecard</td>
<td>/app/center/enhanced/kpireportsetup.nl</td>
</tr>
<tr>
<td>EDIT_LEAD</td>
<td>New Leads</td>
<td>/app/common/entity/custjob.nl?stage=lead</td>
</tr>
<tr>
<td>EDIT_LOCATION</td>
<td>New Locations</td>
<td>/app/common/otherlists/locationtype.nl</td>
</tr>
<tr>
<td>EDIT_MAILMERGE</td>
<td>Bulk Merge</td>
<td>/app/crm/common/merge/mailmerge.nl</td>
</tr>
<tr>
<td>EDIT_MAILMESSAGE</td>
<td>New Word Message</td>
<td>/app/crm/common/merge/mailmessage.nl</td>
</tr>
<tr>
<td>EDIT_MAILTEMPLATE</td>
<td>New Letter Templates</td>
<td>/app/crm/common/merge/mailtemplate.nl</td>
</tr>
<tr>
<td>EDIT_MEDIAITEM</td>
<td>New Media Item</td>
<td>/app/common/media/mediaitem.nl</td>
</tr>
<tr>
<td>EDIT_MEDIAITEMFOLDER</td>
<td>New File Cabinet</td>
<td>/app/common/media/mediaitemfolder.nl</td>
</tr>
<tr>
<td>EDIT_MEMDOC</td>
<td>Enter Memorized Transactions</td>
<td>/app/accounting/transactions/bulkmemdoc.nl</td>
</tr>
<tr>
<td>EDIT_NEXUS</td>
<td>Nexus</td>
<td>/app/setup/nexus.nl</td>
</tr>
<tr>
<td>EDIT_OPENIDSSO</td>
<td>OpenID Single Sign-on</td>
<td>/app/setup/openidsetup.nl</td>
</tr>
<tr>
<td>EDIT_OTHERNAME</td>
<td>New Other Names</td>
<td>/app/common/entity/othername.nl</td>
</tr>
<tr>
<td>EDIT_PARTNER</td>
<td>New Partners</td>
<td>/app/common/entity/partner.nl</td>
</tr>
<tr>
<td>EDIT_PARTNERCOMMISSIONSCHEDULE</td>
<td>New Partner Schedule</td>
<td>/app/crm/sales/commissions/partnercommissionschedule.nl</td>
</tr>
<tr>
<td>EDIT_PARTNERPLANASSIGN</td>
<td>New Partner Plan</td>
<td>/app/crm/sales/commissions/planassignpartner.nl</td>
</tr>
<tr>
<td>EDIT_PAYROLLITEM</td>
<td>New Payroll Items</td>
<td>/app/common/item/payrollitem.nl</td>
</tr>
<tr>
<td>EDIT_PDFMESSAGE</td>
<td>New PDF Message</td>
<td>/app/crm/common/merge/pdftemplate.nl</td>
</tr>
<tr>
<td>EDIT_PDFTEMPLATE</td>
<td>New PDF Templates</td>
<td>/app/crm/common/merge/pdftemplate.nl</td>
</tr>
<tr>
<td>EDIT_PERIOD</td>
<td>New Manage Accounting Periods</td>
<td>/app/setup/period/fiscalperiod.nl</td>
</tr>
<tr>
<td>EDIT_PLANASSIGN</td>
<td>New Employee Plan</td>
<td>/app/crm/sales/commissions/planassign.nl</td>
</tr>
<tr>
<td>EDIT_PRESCATEGORY</td>
<td>New Categories</td>
<td>/app/site/setup/prescategory.nl</td>
</tr>
<tr>
<td>EDIT_PROJECTTASK</td>
<td>New Job Tasks</td>
<td>/app/accounting/project/projecttask.nl</td>
</tr>
<tr>
<td>EDIT_PROSPECT</td>
<td>New Prospects</td>
<td>/app/common/entity/custjob.nl?stage=prospect</td>
</tr>
<tr>
<td>EDIT_QUANTITYPRICINGSCHEDULE</td>
<td>New Quantity Pricing Schedule</td>
<td>/app/accounting/otherlists/quantitypricingschedule.nl</td>
</tr>
<tr>
<td>EDIT_REDIRECT</td>
<td>New Redirect</td>
<td>/app/site/setup/redirect.nl</td>
</tr>
<tr>
<td>EDIT_REFERRALCODE</td>
<td>New Promotion Codes</td>
<td>/app/crm/sales/referralcode.nl</td>
</tr>
<tr>
<td>EDIT_RELATEDITEM</td>
<td>New Related Items Category</td>
<td>/app/site/setup/relateditem.nl</td>
</tr>
<tr>
<td>EDIT_REPORTSECTIONTYPE</td>
<td>New Financial Financial Type</td>
<td>/app/reporting/reportsectiontypes.nl</td>
</tr>
<tr>
<td>EDIT_RESOLVECONFLICTS</td>
<td>Resolve Conflicts</td>
<td>/app/common/entity/conflictresolution.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td>EDIT_RESOURCE</td>
<td>Resource</td>
<td>/app/crm/calendar/resource.nl</td>
</tr>
<tr>
<td>EDIT_REVRECSCHEDULE</td>
<td>New Revenue Recognition Template</td>
<td>/app/accounting/otherlists/revrecschedule.nl</td>
</tr>
<tr>
<td>EDIT_ROLE</td>
<td>New Role</td>
<td>/app/setup/role.nl</td>
</tr>
<tr>
<td>EDIT_RPTGROUP</td>
<td>New Financial Statement Section</td>
<td>/app/reporting/reportgroup.nl</td>
</tr>
<tr>
<td>EDIT_RPTLAYOUT</td>
<td>New Financial Statement Layout</td>
<td>/app/reporting/reportsectionlayout.nl</td>
</tr>
<tr>
<td>EDIT_RSSFEED</td>
<td>New RSS Feed</td>
<td>/app/site/hosting/rssfeed.nl</td>
</tr>
<tr>
<td>EDIT_SALESCAMPAIGN</td>
<td>New Sales Campaign</td>
<td>/app/crm/marketing/salescampaign.nl</td>
</tr>
<tr>
<td>EDIT_SALESTEAM</td>
<td>New Sales Teams</td>
<td>/app/crm/common/crmgroup.nl?grouptype=SalesTeam</td>
</tr>
<tr>
<td>EDIT_SALESTERRITORY</td>
<td>New Manage Sales Territories</td>
<td>/app/crm/sales/salesterritory.nl</td>
</tr>
<tr>
<td>EDIT_SAVEDSEARCH</td>
<td>New Saved Search</td>
<td>/app/common/search/search.nl?cu=T&amp;e=F</td>
</tr>
<tr>
<td>EDIT_SCRIPT</td>
<td>New Script</td>
<td>/app/common/scripting/script.nl</td>
</tr>
<tr>
<td>EDIT_SCRIPTEDRECORD</td>
<td>Scripted Record</td>
<td>/app/common/scripting/scriptedrecord.nl</td>
</tr>
<tr>
<td>EDIT_SCRIPTRECORD</td>
<td>New Script Deployment</td>
<td>/app/common/scripting/scriptrecord.nl</td>
</tr>
<tr>
<td>EDIT_SEARCH</td>
<td>New Search</td>
<td>/app/common/search/search.nl</td>
</tr>
<tr>
<td>EDIT_SHIPITEM</td>
<td>New Shipping Items</td>
<td>/app/common/item/shipitem.nl</td>
</tr>
<tr>
<td>EDIT_SITEITEMTEMPLAT</td>
<td>Item Template</td>
<td>/app/site/setup/siteitemtemplate.nl</td>
</tr>
<tr>
<td>EDIT_SITEMEDIA</td>
<td>Site Media</td>
<td>/app/site/media/sitemedia.nl</td>
</tr>
<tr>
<td>EDIT_SITETAG</td>
<td>Web Site Tag</td>
<td>/app/site/setup/sitetag.nl</td>
</tr>
<tr>
<td>EDIT_SITETHHEME</td>
<td>Web Site Theme</td>
<td>/app/site/setup/sitetheme.nl</td>
</tr>
<tr>
<td>EDIT_SOLUTION</td>
<td>New Solutions</td>
<td>/app/crm/support/kb/solution.nl</td>
</tr>
<tr>
<td>EDIT_SSCATEGORY</td>
<td>New Publish Saved Search</td>
<td>/app/site/setup/sscategory.nl</td>
</tr>
<tr>
<td>EDIT_STATE</td>
<td>New State/Province/County</td>
<td>/app/setup/state.nl</td>
</tr>
<tr>
<td>EDIT_STOREITEMLISTLAYOUT</td>
<td>New Layouts</td>
<td>/app/site/setup/storeitemlistlayout.nl</td>
</tr>
<tr>
<td>EDIT_STORETAB</td>
<td>New Tabs</td>
<td>/app/site/setup/storetab.nl</td>
</tr>
<tr>
<td>EDIT_SUBSIDIARY</td>
<td>New Subsidiaries</td>
<td>/app/common/otherlists/subsidiarytype.nl</td>
</tr>
<tr>
<td>EDIT_SUITESIGNON</td>
<td>SuiteSignOn</td>
<td>/app/setup/ssoapp.nl</td>
</tr>
<tr>
<td>EDIT_SUPPORTCASE</td>
<td>New Cases</td>
<td>/app/crm/support/supportcase.nl</td>
</tr>
<tr>
<td>EDIT_SYSALERT</td>
<td>System Alert</td>
<td>/app/common/otherlists/systemalert.nl</td>
</tr>
<tr>
<td>EDIT_TASK</td>
<td>New Tasks</td>
<td>/app/crm/calendar/task.nl</td>
</tr>
<tr>
<td>EDIT_TAXACCT</td>
<td>Tax Control Account</td>
<td>/app/setup/taxacct.nl</td>
</tr>
<tr>
<td>EDIT_TAXGROUP</td>
<td>New Tax Groups</td>
<td>/app/common/item/taxgroup.nl</td>
</tr>
<tr>
<td>EDIT_TAXITEM</td>
<td>New Tax Codes</td>
<td>/app/common/item/taxitem.nl</td>
</tr>
<tr>
<td>EDIT_TAXPERIOD</td>
<td>New Manage Tax Periods</td>
<td>/app/setup/period/taxperiod.nl</td>
</tr>
<tr>
<td>EDIT_TAXSCHEDULE</td>
<td>New Tax Schedule</td>
<td>/app/common/item/taxschedule.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>EDIT_TAXTYPE</td>
<td>Tax Type</td>
<td>/app/setup/taxtype.nl</td>
</tr>
<tr>
<td>EDIT_THIRDPARTYTRACKINGPIXEL</td>
<td>Third Party Conversion Tracking URLs</td>
<td>/app/accounting/otherlists/trackingpixel.nl</td>
</tr>
<tr>
<td>EDIT_TOPIC</td>
<td>New Topics</td>
<td>/app/crm/support/kb/topic.nl</td>
</tr>
<tr>
<td>EDIT_TRANSACTIONLIST</td>
<td>Paycheck History</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Paycheck</td>
</tr>
<tr>
<td>EDIT_TRAN_ADJJOURNAL</td>
<td>Make Currency Adjustment Journal Entries</td>
<td>/app/accounting/transactions/journal.nl?adjustmentjournal=T</td>
</tr>
<tr>
<td>EDIT_TRAN_BINTRNFR</td>
<td>Bin Transfer</td>
<td>/app/accounting/transactions/bintrnfr.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_BINWKSHT</td>
<td>Bin Putaway Worksheet</td>
<td>/app/accounting/transactions/binwksht.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_BUILD</td>
<td>Build Assemblies</td>
<td>/app/accounting/transactions/build.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_CARDCHRG</td>
<td>Use Credit Card</td>
<td>/app/accounting/transactions/cardchrg.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_CASHRFND</td>
<td>Refund Cash Sales</td>
<td>/app/accounting/transactions/cashrfnd.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_CASHSALE</td>
<td>Enter Cash Sales</td>
<td>/app/accounting/transactions/cashsale.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_CHECK</td>
<td>Write Checks</td>
<td>/app/accounting/transactions/check.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_COMMISSN</td>
<td>Individual Employee Commission</td>
<td>/app/accounting/transactions/commissn.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_CUSTCHRG</td>
<td>Create Statement Charges</td>
<td>/app/accounting/transactions/custchrg.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_CUSTCRED</td>
<td>Issue Credit Memos</td>
<td>/app/accounting/transactions/custcred.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_CUSTDEP</td>
<td>Record Customer Deposits</td>
<td>/app/accounting/transactions/custdep.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_CUSTINV</td>
<td>Create Invoices</td>
<td>/app/accounting/transactions/custinv.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_CUSTPYMT</td>
<td>Accept Customer Payments</td>
<td>/app/accounting/transactions/custpymt.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_CUSTRFND</td>
<td>Issue Customer Refund</td>
<td>/app/accounting/transactions/custrfnd.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_DEPAPPL</td>
<td>Apply Customer Deposits</td>
<td>/app/accounting/transactions/depappl.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_DEPOSIT</td>
<td>Make Deposits</td>
<td>/app/accounting/transactions/deposit.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_ESTIMATE</td>
<td>Prepare Estimates</td>
<td>/app/accounting/transactions/estimate.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_EXPREPT</td>
<td>Enter Expense Reports</td>
<td>/app/accounting/transactions/exprept.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_FXREVAL</td>
<td>Revalue Open Currency Balances</td>
<td>/app/accounting/transactions/fxrevalsetup.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_ICJOURNAL</td>
<td>Make Intercompany Journal Entries</td>
<td>/app/accounting/transactions/journal.nl?icj=T</td>
</tr>
<tr>
<td>EDIT_TRAN_ICJOURNALIMPORT</td>
<td>Import Intercompany Journal Entries</td>
<td>/app/setup/assistants/nsimport/simpleimport.nl?rectype=INTERCOMPANYJOURNALENTRY</td>
</tr>
<tr>
<td>EDIT_TRAN_ICTRNFRORD</td>
<td>Enter Intercompany Transfer Orders</td>
<td>/app/accounting/transactions/trnfrord.nl?icto=T</td>
</tr>
<tr>
<td>EDIT_TRAN_INVADJST</td>
<td>Adjust Inventory</td>
<td>/app/accounting/transactions/invadjst.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_INVDISTR</td>
<td>Distribute Inventory</td>
<td>/app/accounting/transactions/invdistr.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_INVTRNFR</td>
<td>Transfer Inventory</td>
<td>/app/accounting/transactions/invtrnfr.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>EDIT_TRAN_INVWKSHLT</td>
<td>Adjust Inventory Worksheet</td>
<td>/app/accounting/transactions/invwkshlt.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_ITEMRCPT</td>
<td>New Item Receipt</td>
<td>/app/accounting/transactions/itemrcpt.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_ITEMSHIP</td>
<td>New Item Fulfillment</td>
<td>/app/accounting/transactions/itemship.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_JOURNAL</td>
<td>Make Journal Entries</td>
<td>/app/accounting/transactions/journal.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_JOURNALIMPORT</td>
<td>Import Journal Entries</td>
<td>/app/setup/assistants/nsimport/simpleimport.nl?rectype=JOURNALENTRY</td>
</tr>
<tr>
<td>EDIT_TRAN_LIAADJST</td>
<td>Liability Adjustment</td>
<td>/app/accounting/transactions/liaadjst.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_LIABPMT</td>
<td>Pay Payroll Liabilities</td>
<td>/app/accounting/transactions/liabpymt.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_OPPRTNTY</td>
<td>Create Opportunities</td>
<td>/app/accounting/transactions/opprntty.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_PARTNERCOMMISSN</td>
<td>Individual Partner Commission</td>
<td>/app/accounting/transactions/partnercommissn.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_PAYCHECK</td>
<td>Create Individual Paycheck</td>
<td>/app/accounting/transactions/paycheck.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_PURCHORD</td>
<td>Enter Purchase Orders</td>
<td>/app/accounting/transactions/purchord.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_PURCHORD_REQ</td>
<td>Enter Purchase Requests</td>
<td>/app/accounting/transactions/purchord.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_REPLENISHLOC</td>
<td>Replenish Location</td>
<td>/app/accounting/transactions/replenishloc.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_REVCOMMRV</td>
<td>RevComRv</td>
<td>/app/accounting/transactions/revcomrv.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_RTNAUTH</td>
<td>Issue Return Authorizations</td>
<td>/app/accounting/transactions/rtnauth.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_SALESORD</td>
<td>Enter Sales Orders</td>
<td>/app/accounting/transactions/salesord.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_TAXLIAB</td>
<td>Write GST Liability</td>
<td>/app/accounting/transactions/vatliab.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_TAXLIAB2</td>
<td>Write Tax Liability</td>
<td>/app/accounting/transactions/vatliab.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_TAXPYMT</td>
<td>Pay Sales Tax</td>
<td>/app/accounting/transactions/taxpymt.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_TAXPYMT2</td>
<td>Pay Sales Tax</td>
<td>/app/accounting/transactions/taxpymt.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_TAXPYMTCA</td>
<td>Pay PST</td>
<td>/app/accounting/transactions/taxpymt.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_TEGCOLLECT</td>
<td>Collect Tegata</td>
<td>/app/accounting/transactions/tegcollect.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_TEGPAY</td>
<td>Pay Tegata</td>
<td>/app/accounting/transactions/tegpay.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_TEGPYBL</td>
<td>Issue Tegata</td>
<td>/app/accounting/transactions/tegpybl.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_TEGRCVBL</td>
<td>Receive Tegata</td>
<td>/app/accounting/transactions/tegrcvbl.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_TRANSFER</td>
<td>Transfer Funds</td>
<td>/app/accounting/transactions/transfer.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_TRNFRORD</td>
<td>Enter Transfer Orders</td>
<td>/app/accounting/transactions/trnfrord.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_UNBUILD</td>
<td>Unbuild Assemblies</td>
<td>/app/accounting/transactions/unbuild.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_VATLIABAU</td>
<td>Pay Tax Liability</td>
<td>/app/accounting/transactions/vatliab.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_VENDAUTH</td>
<td>Enter Vendor Return Authorizations</td>
<td>/app/accounting/transactions/vendauth.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_VENDBILL</td>
<td>Enter Bills</td>
<td>/app/accounting/transactions/vendbill.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_VENDCRED</td>
<td>Enter Vendor Credits</td>
<td>/app/accounting/transactions/vendcred.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_VENDPYMT</td>
<td>Pay Single Vendor</td>
<td>/app/accounting/transactions/vendpymt.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>EDIT_TRAN_WORKORD</td>
<td>Enter Work Orders</td>
<td>/app/accounting/transactions/workord.nl</td>
</tr>
<tr>
<td>EDIT_TRAN_YTDADJST</td>
<td>Create Payroll Adjustment</td>
<td>/app/accounting/transactions/ytdadjst.nl</td>
</tr>
<tr>
<td>EDIT_UNITSTYPE</td>
<td>New Units Of Measure</td>
<td>/app/common/units/unitstype.nl</td>
</tr>
<tr>
<td>EDIT_UPGRADEBUNDLE</td>
<td>Managed Bundles</td>
<td>/app/bundler/bundlelist.nl?type=S&amp;subtype=m</td>
</tr>
<tr>
<td>EDIT_URLALIAS</td>
<td>New Promotional URL</td>
<td>/app/setup/urlalias.nl</td>
</tr>
<tr>
<td>EDIT_VENDOR</td>
<td>New Vendors</td>
<td>/app/common/entity/vendor.nl</td>
</tr>
<tr>
<td>EDIT_WORKCALENDAR</td>
<td>New Work Calendar</td>
<td>/app/accounting/project/workcalendar.nl</td>
</tr>
<tr>
<td>EDIT_WORKFLOW</td>
<td>New Workflow</td>
<td>/app/common/workflow/setup/workflowmanager.nl</td>
</tr>
<tr>
<td>EDIT_WORKPLACE</td>
<td>New Workplaces</td>
<td>/app/common/otherlists/workplacetype.nl</td>
</tr>
<tr>
<td>INTL_SECTION</td>
<td>New Tabs</td>
<td>/internal/admin/section.nl</td>
</tr>
<tr>
<td>INTL_SECTIONS</td>
<td>Tabs</td>
<td>/internal/admin/sections.nl</td>
</tr>
<tr>
<td>INTL_TASKLINK</td>
<td>New Tasklinks</td>
<td>/internal/admin/tasklink.nl</td>
</tr>
<tr>
<td>INTL_TASKLINKS</td>
<td>Tasklinks</td>
<td>/internal/admin/tasklinks.nl</td>
</tr>
<tr>
<td>LIST_ACCOUNT</td>
<td>Accounts</td>
<td>/app/accounting/account/accounts.nl</td>
</tr>
<tr>
<td>LIST_ACCOUNTINGOTHERLIST</td>
<td>Accounting Lists</td>
<td>/app/common/otherlists/accountingotherlists.nl</td>
</tr>
<tr>
<td>LIST_ACTIVITY</td>
<td>Activity List</td>
<td>/app/crm/calendar/activitylist.nl</td>
</tr>
<tr>
<td>LIST_AD_DP_BATCHES</td>
<td>Payroll Batches</td>
<td>/app/payroll/payrollbatchlist.nl</td>
</tr>
<tr>
<td>LIST_AD_DP_PAYCHECK</td>
<td>Individual Paychecks</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Paycheck</td>
</tr>
<tr>
<td>LIST_ALLOCATION</td>
<td>Allocation Schedules</td>
<td>/app/accounting/transactions/allocschedulelist.nl</td>
</tr>
<tr>
<td>LIST_AMORTIZATION</td>
<td>Amortization Templates</td>
<td>/app/accounting/otherlists/amortizationtemplates.nl</td>
</tr>
<tr>
<td>LIST_AMORTIZATION_SCHEDULE</td>
<td>Amortization Schedules</td>
<td>/app/accounting/otherlists/amortizationschedules.nl</td>
</tr>
<tr>
<td>LIST_APPROVEACH</td>
<td>Approve Direct Deposits</td>
<td>/app/accounting/transactions/approveach.nl</td>
</tr>
<tr>
<td>LIST_APPROVEEFT</td>
<td>Approve Electronic Funds Transfers</td>
<td>/app/accounting/transactions/approveeft.nl</td>
</tr>
<tr>
<td>LIST_APPROVEVP</td>
<td>Approve Vendor Payment Transfers</td>
<td>/app/accounting/transactions/approvevp.nl</td>
</tr>
<tr>
<td>LIST_BILLING_SCHEDULE</td>
<td>Billing Schedules</td>
<td>/app/accounting/otherlists/billingschedules.nl</td>
</tr>
<tr>
<td>LIST_BIN</td>
<td>Bins</td>
<td>/app/accounting/transactions/inventory/binlist.nl</td>
</tr>
<tr>
<td>LIST_BUDGET</td>
<td>Budgets</td>
<td>/app/accounting/transactions/budgetlist.nl</td>
</tr>
<tr>
<td>LIST_BUDGETRATES</td>
<td>Budget Exchange Rates</td>
<td>/app/accounting/otherlists/budgetrates.nl</td>
</tr>
<tr>
<td>LIST_BULKOP</td>
<td>Mass Updates</td>
<td>/app/common/bulk/bulkops.nl</td>
</tr>
<tr>
<td>LIST_BULKRESULTS</td>
<td>Mass Update Results</td>
<td>/app/common/bulk/bulkresults.nl</td>
</tr>
<tr>
<td>LIST_BUNDLE</td>
<td>List Bundles</td>
<td>/app/bundler/bundlelist.nl?type=S</td>
</tr>
<tr>
<td>LISTCALENDAR</td>
<td>Calendar</td>
<td>/app/crm/calendar/calendar.nl</td>
</tr>
<tr>
<td>LIST_CALL</td>
<td>Phone Calls</td>
<td>/app/crm/calendar/calllist.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td>LIST_CAMPAIGN</td>
<td>Marketing Campaigns</td>
<td>/app/crm/marketing/campaignlist.nl?Campaign_ISSALES_CAMPAIGN=F</td>
</tr>
<tr>
<td>LIST_CAMPAIGN_AUDIENCE</td>
<td>Campaign Audiences</td>
<td>/app/crm/marketing/campaignaudiences.nl</td>
</tr>
<tr>
<td>LIST_CAMPAIGNCALENDAR</td>
<td>Campaign Calendar</td>
<td>/app/crm/marketing/campaigncalendar.nl</td>
</tr>
<tr>
<td>LIST_CAMPAIGNCATEGORY</td>
<td>Campaign Categories</td>
<td>/app/crm/marketing/campaigncategories.nl</td>
</tr>
<tr>
<td>LIST_CAMPAIGN_CHANNEL</td>
<td>Campaign Channels</td>
<td>/app/crm/marketing/campaignchannels.nl</td>
</tr>
<tr>
<td>LIST_CAMPAIGNEMAIL</td>
<td>Campaign Email Addresses</td>
<td>/app/crm/marketing/campaignemails.nl</td>
</tr>
<tr>
<td>LIST_CAMPAIGNFAMILY</td>
<td>Campaign Families</td>
<td>/app/crm/marketing/campaignfamilies.nl</td>
</tr>
<tr>
<td>LIST_CAMPAIGNOFFER</td>
<td>Campaign Offers</td>
<td>/app/crm/marketing/campaignoffers.nl</td>
</tr>
<tr>
<td>LIST_CAMPAIGNSEARCHENGINE</td>
<td>Campaign Search Engines</td>
<td>/app/crm/marketing/campaignsearchengines.nl</td>
</tr>
<tr>
<td>LIST_CAMPAIGNSUBSCRIPTION</td>
<td>Campaign Subscriptions</td>
<td>/app/crm/marketing/campaignsubscriptions.nl</td>
</tr>
<tr>
<td>LIST_CAMPAIGNVERTICAL</td>
<td>Campaign Verticals</td>
<td>/app/crm/marketing/campaignverticals.nl</td>
</tr>
<tr>
<td>LIST_CASEFIELDRULE</td>
<td>Set Up Case Rules</td>
<td>/app/crm/support/casefieldrules.nl</td>
</tr>
<tr>
<td>LIST_CASEFORM</td>
<td>Online Case Forms</td>
<td>/app/crm/support/caseforms.nl</td>
</tr>
<tr>
<td>LIST_CASEISSUE</td>
<td>Case Issues</td>
<td>/app/crm/support/caseissuelist.nl</td>
</tr>
<tr>
<td>LIST_CASEORIGIN</td>
<td>Case Origin Types</td>
<td>/app/crm/support/caseoriginlist.nl</td>
</tr>
<tr>
<td>LIST_CASEPRIORITY</td>
<td>Case Priorities</td>
<td>/app/crm/support/caseprioritylist.nl</td>
</tr>
<tr>
<td>LIST_CASESTATUS</td>
<td>Case Statuses</td>
<td>/app/crm/support/casestatuslist.nl</td>
</tr>
<tr>
<td>LIST_CASETERRITORIES</td>
<td>Case Territory List</td>
<td>/app/crm/support/supportterritorylist.nl</td>
</tr>
<tr>
<td>LIST_CASETERRITORY</td>
<td>Manage Case Territories</td>
<td>/app/crm/common/automation/territorymanager.nl?case=T</td>
</tr>
<tr>
<td>LIST_CASETERRITORYASSIGN</td>
<td>Territory Reassignment</td>
<td>/app/common/bulk/bulkop.nl?searchtype=Case&amp;opcode=ReAssign</td>
</tr>
<tr>
<td>LIST_CASETYPE</td>
<td>Case Types</td>
<td>/app/crm/support/cas_ty pelist.nl</td>
</tr>
<tr>
<td>LIST_CCTRAN</td>
<td>View Credit Card Transactions</td>
<td>/app/accounting/transactions/cardtrans.nl</td>
</tr>
<tr>
<td>LIST_CHART_ACCOUNT</td>
<td>Chart of Accounts</td>
<td>/app/accounting/account/accounts.nl?report=T&amp;code=COA</td>
</tr>
<tr>
<td>LIST_CLASS</td>
<td>Classes</td>
<td>/app/common/otherlists/classlist.nl</td>
</tr>
<tr>
<td>LIST_COLORTHEME</td>
<td>Color Themes</td>
<td>/app/setup/look/colorthemes.nl</td>
</tr>
<tr>
<td>LIST_COMMISIONSCHEDULE</td>
<td>Employee Schedules</td>
<td>/app/crm/sales/commissions/commissionscheds.nl</td>
</tr>
<tr>
<td>LIST_COMPETITOR</td>
<td>Competitors</td>
<td>/app/crm/sales/competitorlist.nl</td>
</tr>
<tr>
<td>LISTCONSOLRATES</td>
<td>Consolidated Exchange Rates</td>
<td>/app/accounting/otherlists/consolidatedrates.nl</td>
</tr>
<tr>
<td>LIST_CONTACT</td>
<td>Contacts</td>
<td>/app/common/entity/contactlist.nl</td>
</tr>
<tr>
<td>LIST_COUNTRY</td>
<td>Set Up Countries</td>
<td>/app/setup/countries.nl</td>
</tr>
<tr>
<td>LIST_CRMGROUP</td>
<td>Groups</td>
<td>/app/crm/common/crmgrouplist.nl</td>
</tr>
<tr>
<td>LIST_CRMOTHERLIST</td>
<td>CRM Lists</td>
<td>/app/common/otherlists/crmotherlists.nl</td>
</tr>
<tr>
<td>LIST_CRMTEMPLATE</td>
<td>Marketing Templates</td>
<td>/app/crm/common/merge/marketingtemplates.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>LIST_CURRENCY</td>
<td>Currencies</td>
<td>/app/common/multicurrency/currencylist.nl</td>
</tr>
<tr>
<td>LIST_CURRENCYRATE</td>
<td>Currency Exchange Rates</td>
<td>/app/common/multicurrency/currencyratelist.nl</td>
</tr>
<tr>
<td>LIST_CUSTBODYFIELD</td>
<td>Transaction Body Fields</td>
<td>/app/common/custom/bodycustfields.nl</td>
</tr>
<tr>
<td>LIST_CUSTCATEGORY</td>
<td>Center Categories</td>
<td>/app/common/custom/custcategories.nl</td>
</tr>
<tr>
<td>LIST_CUSTCENTER</td>
<td>Centers</td>
<td>/app/common/custom/custcenters.nl</td>
</tr>
<tr>
<td>LIST_CUSTCOLUMNFIELD</td>
<td>Transaction Column Fields</td>
<td>/app/common/custom/columncustfields.nl</td>
</tr>
<tr>
<td>LIST_CUSTEMAILLAYOUT</td>
<td>Transaction Form HTML Layouts</td>
<td>/app/common/custom/custemaillayouts.nl</td>
</tr>
<tr>
<td>LIST_CUSTENTITYFIELD</td>
<td>Entity Fields</td>
<td>/app/common/custom/entitycustfields.nl</td>
</tr>
<tr>
<td>LIST_CUSTENTRYFORM</td>
<td>Entry Forms</td>
<td>/app/common/custom/custentryforms.nl</td>
</tr>
<tr>
<td>LIST_CUSTEVENTFIELD</td>
<td>CRM Fields</td>
<td>/app/common/custom/eventcustfields.nl</td>
</tr>
<tr>
<td>LIST_CUSTFIELDTAB</td>
<td>Subtabs</td>
<td>/app/common/custom/custfieldtabs.nl</td>
</tr>
<tr>
<td>LIST_CUSTFORM</td>
<td>Transaction Forms</td>
<td>/app/common/custom/custforms.nl</td>
</tr>
<tr>
<td>LIST_CUSTITEMFIELD</td>
<td>Item Fields</td>
<td>/app/common/custom/itemcustfields.nl</td>
</tr>
<tr>
<td>LIST_CUSTITEMNUMBERFIELD</td>
<td>Item Number Fields</td>
<td>/app/common/custom/itemnumbercustfields.nl</td>
</tr>
<tr>
<td>LIST_CUSTJOB</td>
<td>Customers</td>
<td>/app/common/entity/custjoblist.nl?Customer_STAGE=CUSTOMER</td>
</tr>
<tr>
<td>LIST_CUSTLAYOUT</td>
<td>Transaction Form PDF Layouts</td>
<td>/app/common/custom/custlayouts.nl</td>
</tr>
<tr>
<td>LIST_CUSTLIST</td>
<td>Lists</td>
<td>/app/common/custom/custlists.nl</td>
</tr>
<tr>
<td>LIST_CUSTOMCODEFILES</td>
<td>SuiteScripts</td>
<td>/app/common/media/mediaitemfolders.nl?folder=-15</td>
</tr>
<tr>
<td>LISTCUSTOMERFIELDRULE</td>
<td>Set Up Sales Rules</td>
<td>/app/crm/sales/customerfieldrules.nl</td>
</tr>
<tr>
<td>LIST_CUSTOMERFORM</td>
<td>Online Customer Forms</td>
<td>/app/crm/sales/leadforms.nl</td>
</tr>
<tr>
<td>LIST_CUSTOMERSTATUS</td>
<td>Customer Statuses</td>
<td>/app/crm/sales/customerstatuslist.nl</td>
</tr>
<tr>
<td>LIST_CUSTOMSUBLIST</td>
<td>Sublists</td>
<td>/app/common/custom/customsublists.nl</td>
</tr>
<tr>
<td>LIST_CUSTOTHERFIELD</td>
<td>Other Custom Fields</td>
<td>/app/common/custom/othercustfields.nl</td>
</tr>
<tr>
<td>LIST_CUSTRECORD</td>
<td>Record Types</td>
<td>/app/common/custom/custrecords.nl</td>
</tr>
<tr>
<td>LIST_CUSTRECORDFORM</td>
<td>Online Custom Record Forms</td>
<td>/app/common/custom/custrecordforms.nl</td>
</tr>
<tr>
<td>LIST_CUSTSECTION</td>
<td>Center Tabs</td>
<td>/app/common/custom/custsections.nl</td>
</tr>
<tr>
<td>LIST_DEPARTMENT</td>
<td>Departments</td>
<td>/app/common/otherlists/departmentlist.nl</td>
</tr>
<tr>
<td>LIST_DIFFROLE</td>
<td>Show Role Differences</td>
<td>/app/setup/diffroles.nl</td>
</tr>
<tr>
<td>LIST_DUPLICATES</td>
<td>Mass Duplicate Record Merge</td>
<td>/app/common/entity/manageduplicates.nl</td>
</tr>
<tr>
<td>LIST_EMAILTEMPLATE</td>
<td>Email Templates</td>
<td>/app/crm/common/merge/emailtemplates.nl</td>
</tr>
<tr>
<td>LIST_EMPLCATEGORY</td>
<td>Employee Directory</td>
<td>/app/site/setup/emplcategories.nl</td>
</tr>
<tr>
<td>LIST_EMPLOYEE</td>
<td>Employees</td>
<td>/app/common/entity/employeelist.nl</td>
</tr>
<tr>
<td>LIST_EMPOTHERLIST</td>
<td>Employee Related Lists</td>
<td>/app/common/otherlists/empotherlists.nl</td>
</tr>
<tr>
<td>LIST_ENTITY</td>
<td>All Entities</td>
<td>/app/common/entity/entitylist.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>LIST_ESCALATIONRULE</td>
<td>Set Up Escalation Rules</td>
<td>/app/crm/support/escalationfieldrules.nl</td>
</tr>
<tr>
<td>LIST_ESCALATIONTERRITORY</td>
<td>Manage Escalation Assignment</td>
<td>/app/crm/common/automation/territorymanager.nl?escalation=T</td>
</tr>
<tr>
<td>LIST_EVENT</td>
<td>Events</td>
<td>/app/crm/calendar/eventlist.nl</td>
</tr>
<tr>
<td>LIST_EXPSCATEGORY</td>
<td>Expense Categories</td>
<td>/app/accounting/otherlists/expscategories.nl</td>
</tr>
<tr>
<td>LIST_FAXTEMPLATE</td>
<td>Fax Templates</td>
<td>/app/crm/common/merge/faxtemplates.nl</td>
</tr>
<tr>
<td>LIST_FCSITEFOLDER</td>
<td>Web Site Hosting Files</td>
<td>/app/common/media/mediaitemfolders.nl?folder=-100</td>
</tr>
<tr>
<td>LIST_FILECABINET</td>
<td>Media Items</td>
<td>/app/common/media/mediaitems.nl</td>
</tr>
<tr>
<td>LIST_FINCHRG</td>
<td>Assess Finance Charges</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CustInvc&amp;Transaction_FINCHRG=T</td>
</tr>
<tr>
<td>LIST_FORECAST</td>
<td>List Sales Rep Forecast</td>
<td>/app/crm/sales/forecastlist.nl?Forecast_ISTEAM=F</td>
</tr>
<tr>
<td>LIST_GCOACCOUNT</td>
<td>Google Checkout Accounts</td>
<td>/app/site/setup/google/gcoaccountlist.nl</td>
</tr>
<tr>
<td>LIST_GIFTCERTIFICATES</td>
<td>Gift Certificates</td>
<td>/app/accounting/otherlists/giftcertificates.nl</td>
</tr>
<tr>
<td>LIST_IMAGE</td>
<td>Images</td>
<td>/app/common/media/mediaitemfolders.nl?folder=-4</td>
</tr>
<tr>
<td>LIST_INFOITEM</td>
<td>Information Items</td>
<td>/app/site/setup/infoitemlist.nl?searchid=-2540</td>
</tr>
<tr>
<td>LIST_INFOITEMFORM</td>
<td>Publish Forms</td>
<td>/app/site/setup/infoitemlist.nl?searchid=-2541</td>
</tr>
<tr>
<td>LIST_INSTALLEDPACKAGE</td>
<td>Installed Bundles</td>
<td>/app/bundler/bundlelist.nl?type=I</td>
</tr>
<tr>
<td>LIST_ISSUE</td>
<td>Issues</td>
<td>/app/crm/support/issuedb/issuelist.nl</td>
</tr>
<tr>
<td>LIST_ITEM</td>
<td>Items</td>
<td>/app/common/item/itemlist.nl</td>
</tr>
<tr>
<td>LIST_ITEMOPTION</td>
<td>Transaction Item Options</td>
<td>/app/common/custom/itemoptions.nl</td>
</tr>
<tr>
<td>LIST_JOB</td>
<td>Jobs</td>
<td>/app/common/entity/joblist.nl</td>
</tr>
<tr>
<td>LIST_KBCATEGORY</td>
<td>Knowledge Base</td>
<td>/app/site/setup/kbcategories.nl</td>
</tr>
<tr>
<td>LIST_KPIREPORT</td>
<td>KPI Scorecards</td>
<td>/app/center/enhanced/kpireports.nl</td>
</tr>
<tr>
<td>LIST_LEAD</td>
<td>Leads</td>
<td>/app/crm/support/issuedb/custjoblist.nl?Customer_STAGE=LEAD</td>
</tr>
<tr>
<td>LIST_LOCATION</td>
<td>Locations</td>
<td>/app/common/otherlists/locationlist.nl</td>
</tr>
<tr>
<td>LIST_MAILMERGE</td>
<td>Merge History</td>
<td>/app/crm/common/merge/mailmergehistory.nl</td>
</tr>
<tr>
<td>LIST_MAILTEMPLATE</td>
<td>Letter Templates</td>
<td>/app/crm/common/merge/mailtemplates.nl</td>
</tr>
<tr>
<td>LIST_MEDIAITEMFOLDER</td>
<td>File Cabinet</td>
<td>/app/common/media/mediaitemfolders.nl</td>
</tr>
<tr>
<td>LIST_MEDIAITEMFOLDER_LOG</td>
<td>Job Status</td>
<td>/app/external/xml/upload/uploadlog.nl?displayType=FILECABINET</td>
</tr>
<tr>
<td>LIST_MEMDOC</td>
<td>Memorized Transactions</td>
<td>/app/accounting/otherlists/memdoclist.nl</td>
</tr>
<tr>
<td>LIST_MGRFORECAST</td>
<td>List Sales Manager Forecast</td>
<td>/app/crm/sales/forecastlist.nl?Forecast_ISTEAM=T</td>
</tr>
<tr>
<td>LIST_MYROLES</td>
<td>My Roles</td>
<td>/app/center/myroles.nl</td>
</tr>
<tr>
<td>LIST_OTHERNAME</td>
<td>Other Names</td>
<td>/app/common/entity/othernames.nl</td>
</tr>
<tr>
<td>LIST_PARTNER</td>
<td>Partners</td>
<td>/app/common/entity/partnerlist.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>LIST_PARTNERCOMMISIIONSCHEDULED</td>
<td>Partner Schedules</td>
<td>/app/crm/sales/commissions/partnercommissionsscheds.nl</td>
</tr>
<tr>
<td>LIST_PARTNERPLANASSIGN</td>
<td>Partner Plans</td>
<td>/app/crm/sales/commissions/partnercommissionplans.nl</td>
</tr>
<tr>
<td>LIST_PARTNERPLANASSIGNS</td>
<td>Partner Plan Assignments</td>
<td>/app/crm/sales/commissions/partnerplanassigns.nl</td>
</tr>
<tr>
<td>LIST_PAYROLLBATCH</td>
<td>Payroll Batches</td>
<td>/app/payroll/payrollbatchlist.nl</td>
</tr>
<tr>
<td>LIST_PAYROLLISSUES</td>
<td>Payroll Issues</td>
<td>/app/payroll/cdstatus.nl</td>
</tr>
<tr>
<td>LIST_PAYROLLITEM</td>
<td>Payroll Items</td>
<td>/app/common/item/payrollitems.nl</td>
</tr>
<tr>
<td>LIST_PDFTEMPLATE</td>
<td>PDF Templates</td>
<td>/app/crm/common/merge/pdftemplates.nl</td>
</tr>
<tr>
<td>LIST_PERIOD</td>
<td>Manage Accounting Periods</td>
<td>/app/setup/period/fiscalperiods.nl</td>
</tr>
<tr>
<td>LIST_PLANASSIGN</td>
<td>Employee Plans</td>
<td>/app/crm/sales/commissions/commissionplans.nl</td>
</tr>
<tr>
<td>LIST_PLANASSIGNS</td>
<td>Employee Plan Assignments</td>
<td>/app/crm/sales/commissions/planassigns.nl</td>
</tr>
<tr>
<td>LIST_PRESCATEGORY</td>
<td>Categories</td>
<td>/app/site/setup/prescategories.nl?ctype=PRES</td>
</tr>
<tr>
<td>LIST_PROJECTTASK</td>
<td>Project Tasks</td>
<td>/app/accounting/project/projecttasks.nl</td>
</tr>
<tr>
<td>LIST_PROJECTTASKIMPORT</td>
<td>Import Project Tasks</td>
<td>/app/setup/helpers/nsimport/simpleimport.nl?rectype=PROJECTTASK</td>
</tr>
<tr>
<td>LIST_PROSPECT</td>
<td>Prospects</td>
<td>/app/common/entity/custjoblist.nl?Customer_STAGE=PROSPECT</td>
</tr>
<tr>
<td>LIST_QUANTITYPRICINGSCHEDULES</td>
<td>Quantity Pricing Schedules</td>
<td>/app/accounting/otherlists/quantitypricingschedules.nl</td>
</tr>
<tr>
<td>LIST_QUOTA</td>
<td>Quotas</td>
<td>/app/crm/sales/quotalist.nl</td>
</tr>
<tr>
<td>LIST_RECENTRECORDS</td>
<td>Recent Records</td>
<td>/app/common/otherlists/recentrecords.nl</td>
</tr>
<tr>
<td>LIST_RECVDFILES</td>
<td>Attachments Received</td>
<td>/app/common/media/mediaitemfolders.nl?folder=-10</td>
</tr>
<tr>
<td>LIST_REFERRALCODE</td>
<td>Promotion Codes</td>
<td>/app/crm/sales/referralcodeslist.nl</td>
</tr>
<tr>
<td>LIST_REFERREDITEM</td>
<td>Related Items Categories</td>
<td>/app/site/setup/referreditems.nl</td>
</tr>
<tr>
<td>LIST_REPORESULT</td>
<td>Report Results</td>
<td>/app/reporting/workqueue/reportresultlist.nl</td>
</tr>
<tr>
<td>LIST_REPOSCHEDULE</td>
<td>Report Schedules</td>
<td>/app/reporting/workqueue/reportschedulerlist.nl</td>
</tr>
<tr>
<td>LIST_RESOURCE</td>
<td>Resources</td>
<td>/app/crm/calendar/resources.nl</td>
</tr>
<tr>
<td>LIST_REVRECSCHEDULES</td>
<td>Revenue Recognition Schedules</td>
<td>/app/accounting/otherlists/revrecschedules.nl</td>
</tr>
<tr>
<td>LIST_REVRECSCHEDULE</td>
<td>Revenue Recognition Templates</td>
<td>/app/accounting/otherlists/revrectemplates.nl</td>
</tr>
<tr>
<td>LIST_ROLE</td>
<td>Manage Roles</td>
<td>/app/setup/roletlist.nl</td>
</tr>
<tr>
<td>LIST_RSSFEED</td>
<td>RSS Feeds</td>
<td>/app/site/hosting/rssfeeds.nl</td>
</tr>
<tr>
<td>LIST_SALESCAMPAIGN</td>
<td>Sales Campaigns</td>
<td>/app/crm/marketing/salescampaignlist.nl</td>
</tr>
<tr>
<td>LIST SALESTEAM</td>
<td>Sales Teams</td>
<td>/app/crm/common/salesteamlist.nl</td>
</tr>
<tr>
<td>LIST_SALESTERRITORIES</td>
<td>Sales Territory List</td>
<td>/app/crm/sales/salesterritorylist.nl</td>
</tr>
<tr>
<td>LIST_SALES TERRITORY</td>
<td>Manage Sales Territories</td>
<td>/app/crm/common/automation/territorymanager.nl?sales=T</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>LIST_SALESTERRITORYASSIGN</td>
<td>Territory Reassignment</td>
<td>/app/common/bulk/bulkop.nl?searchtype=Customer&amp;opcode=ReAssign</td>
</tr>
<tr>
<td>LIST_SAVEDASHBOARD</td>
<td>Published Dashboards</td>
<td>/app/center/setup/savedashboards.nl</td>
</tr>
<tr>
<td>LIST_SAVEDBULKOP</td>
<td>Saved Mass Updates</td>
<td>/app/common/bulk/savedbulkops.nl</td>
</tr>
<tr>
<td>LIST_SAVEDSEARCH</td>
<td>Saved Searches</td>
<td>/app/common/search/savedsearches.nl</td>
</tr>
<tr>
<td>LIST_SCRIPT</td>
<td>Scripts</td>
<td>/app/common/scripting/scriptlist.nl</td>
</tr>
<tr>
<td>LIST_SCRIPTCALENDAR</td>
<td>Script Calendar</td>
<td>/app/common/scripting/scriptcalendar.nl</td>
</tr>
<tr>
<td>LIST_SCRIPTEDRECORD</td>
<td>Scripted Records</td>
<td>/app/common/scripting/scriptedrecords.nl</td>
</tr>
<tr>
<td>LIST_SCRIPTRECORD</td>
<td>Script Deployments</td>
<td>/app/common/scripting/scriptrecordlist.nl</td>
</tr>
<tr>
<td>LIST_SCRIPTSTATUS</td>
<td>Scheduled Script Status</td>
<td>/app/common/scripting/scriptstatus.nl</td>
</tr>
<tr>
<td>LIST_SEARCHRESULTS</td>
<td>Search Results</td>
<td>/app/common/search/searchresults.nl</td>
</tr>
<tr>
<td>LIST_SENTFILES</td>
<td>Attachments Sent</td>
<td>/app/common/media/mediaitemfolders.nl?folder=-14</td>
</tr>
<tr>
<td>LIST_SHIPITEM</td>
<td>Shipping Items</td>
<td>/app/common/item/shipitems.nl</td>
</tr>
<tr>
<td>LIST_SHIPPINGMANIFEST</td>
<td>Shipping Manifest</td>
<td>/app/common/shipping/fedex/shippingmanifest.nl</td>
</tr>
<tr>
<td>LIST_SITEITEMTEMPLAT</td>
<td>Item/Category Templates</td>
<td>/app/site/setup/siteitemtemplates.nl</td>
</tr>
<tr>
<td>LIST_SITETAGS</td>
<td>Web Site Tags</td>
<td>/app/site/setup/sitetags.nl</td>
</tr>
<tr>
<td>LIST_SITETHemes</td>
<td>Web Site Themes</td>
<td>/app/site/setup/sitethemes.nl</td>
</tr>
<tr>
<td>LIST_SMBIMPORT</td>
<td>SMB Import</td>
<td>/app/external/xml/upload/upload.nl</td>
</tr>
<tr>
<td>LIST_SOLUTION</td>
<td>Solutions</td>
<td>/app/crm/support/kb/solutions.nl</td>
</tr>
<tr>
<td>LIST_SSCATEGORY</td>
<td>Publish Saved Search</td>
<td>/app/site/setup/sscategories.nl</td>
</tr>
<tr>
<td>LIST_STATE</td>
<td>Set Up States/Provinces/Counties</td>
<td>/app/setup/states.nl</td>
</tr>
<tr>
<td>LIST_STATUSACH</td>
<td>View Direct Deposit Status</td>
<td>/app/accounting/transactions/statusach.nl</td>
</tr>
<tr>
<td>LIST_STATUSEFT</td>
<td>View Electronic Funds Transfer Status</td>
<td>/app/accounting/transactions/statuseft.nl</td>
</tr>
<tr>
<td>LIST_STATUSVP</td>
<td>View Vendor Payment Status</td>
<td>/app/accounting/transactions/statusvp.nl</td>
</tr>
<tr>
<td>LIST_STOREITEMLISTLAYOUT</td>
<td>Layouts</td>
<td>/app/site/setup/storeitemlistlayouts.nl</td>
</tr>
<tr>
<td>LIST_STORETAB</td>
<td>Tabs</td>
<td>/app/site/setup/storetabs.nl</td>
</tr>
<tr>
<td>LIST_SUBSIDIARY</td>
<td>Subsidiaries</td>
<td>/app/common/otherlists/subsidiarylist.nl</td>
</tr>
<tr>
<td>LIST_SUPPORTCASE</td>
<td>Cases</td>
<td>/app/crm/support/caselist.nl</td>
</tr>
<tr>
<td>LIST_SYSALERT</td>
<td>System Alerts</td>
<td>/app/common/otherlists/systemalerts.nl</td>
</tr>
<tr>
<td>LIST_SYSTEMEMAILTEMPLATE</td>
<td>Set Up System Email Templates</td>
<td>/app/crm/common/merge/systememailtemplates.nl</td>
</tr>
<tr>
<td>LIST_TASK</td>
<td>Tasks</td>
<td>/app/crm/calendar/tasklist.nl</td>
</tr>
<tr>
<td>LIST_TAXGROUP</td>
<td>Tax Groups</td>
<td>/app/common/item/taxgroups.nl</td>
</tr>
<tr>
<td>LIST_TAXITEM</td>
<td>Tax Codes</td>
<td>/app/common/item/taxitems.nl</td>
</tr>
<tr>
<td>LIST_TAXPERIOD</td>
<td>Manage Tax Periods</td>
<td>/app/setup/period/taxperiods.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>LIST_TAXSCHEDULE</td>
<td>Tax Schedules</td>
<td>/app/common/item/taxschedules.nl</td>
</tr>
<tr>
<td>LIST_TEMPLATEFILES</td>
<td>Template Files</td>
<td>/app/common/media/mediatemfolders.nl?folder=-9</td>
</tr>
<tr>
<td>LIST_THIRDPARTYTRACKINGPIXEL</td>
<td>Third Party Conversion Tracking URLs</td>
<td>/app/site/setup/google/trackingpixels.nl</td>
</tr>
<tr>
<td>LIST_TOPIC</td>
<td>Topics</td>
<td>/app/crm/support/kb/topics.nl</td>
</tr>
<tr>
<td>LIST_TRANSACTION</td>
<td>Transactions</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=@ALL@</td>
</tr>
<tr>
<td>LIST_TRN_BINTRNFR</td>
<td>Bin Transfers</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=BinTrnfr</td>
</tr>
<tr>
<td>LIST_TRN_BUILD</td>
<td>Assembly Builds</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Build</td>
</tr>
<tr>
<td>LIST_TRN_CARDCHRG</td>
<td>Credit Card Charges</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CardChrg</td>
</tr>
<tr>
<td>LIST_TRN_CASHRFND</td>
<td>Cash Refunds</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CashRfnd</td>
</tr>
<tr>
<td>LIST_TRN_CASHSALE</td>
<td>Cash Sales</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CashSale</td>
</tr>
<tr>
<td>LIST_TRN_CHECK</td>
<td>Checks</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Check</td>
</tr>
<tr>
<td>LIST_TRN_COMMISSN</td>
<td>Individual Employee Commissions</td>
<td>/app/accounting/transactions/commissns.nl</td>
</tr>
<tr>
<td>LIST_TRN_CUSTCHRG</td>
<td>Statement Charges</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CustChrg</td>
</tr>
<tr>
<td>LIST_TRN_CUSTCRED</td>
<td>Credit Memos</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CustCred</td>
</tr>
<tr>
<td>LIST_TRN_CUSTDEP</td>
<td>Customer Deposits</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CustDep</td>
</tr>
<tr>
<td>LIST_TRN_CUSTINVC</td>
<td>Invoices</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CustInvvc</td>
</tr>
<tr>
<td>LIST_TRN_CUSTPYMT</td>
<td>Customer Payments</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CustPymt</td>
</tr>
<tr>
<td>LIST_TRN_CUSTRFND</td>
<td>Customer Refunds</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CustRfnd</td>
</tr>
<tr>
<td>LIST_TRN_DEPAPPL</td>
<td>Deposit Applications</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=DepAppl</td>
</tr>
<tr>
<td>LIST_TRN_DEPOSIT</td>
<td>Deposits</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Deposit</td>
</tr>
<tr>
<td>LIST_TRN_DOWNLOAD</td>
<td>My Downloads</td>
<td>/app/common/entity/downloads.nl</td>
</tr>
<tr>
<td>LIST_TRN_ESTIMATE</td>
<td>Estimates</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Estimate</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>LIST_TRAN_EXPREPT</td>
<td>Expense Reports</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=ExpRept</td>
</tr>
<tr>
<td>LIST_TRAN_FXREVAL</td>
<td>Currency Revaluations</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=FxReval</td>
</tr>
<tr>
<td>LIST_TRAN_ICTRNFRORD</td>
<td>Intercompany Transfer Orders</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TrnfrOrd&amp;icto=T</td>
</tr>
<tr>
<td>LIST_TRAN_INVADJST</td>
<td>Inventory Adjustments</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=InvAdjst</td>
</tr>
<tr>
<td>LIST_TRAN_INVDISTR</td>
<td>Inventory Distributions</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=InvDistr</td>
</tr>
<tr>
<td>LIST_TRAN_INVTRNFR</td>
<td>Inventory Transfers</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=InvTmfr</td>
</tr>
<tr>
<td>LIST_TRAN_INVWKSHT</td>
<td>Inventory Worksheets</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=InvWksht</td>
</tr>
<tr>
<td>LIST_TRAN_ITEMRCPT</td>
<td>Item Receipts</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=ItemRcpt</td>
</tr>
<tr>
<td>LIST_TRAN_ITEMSHIP</td>
<td>Item Fulfillments</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=ItemShip</td>
</tr>
<tr>
<td>LIST_TRAN_LIABPYMT</td>
<td>Liability Payments</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=LiabPymt</td>
</tr>
<tr>
<td>LIST_TRAN_OPPRTNTY</td>
<td>Opportunities</td>
<td>/app/accounting/transactions/opprtntylist.nl</td>
</tr>
<tr>
<td>LIST_TRAN_PARTNERCOMMISSN</td>
<td>Individual Partner Commissions</td>
<td>/app/accounting/transactions/partnercommissns.nl</td>
</tr>
<tr>
<td>LIST_TRAN_PAYCHECK</td>
<td>Paychecks</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Paycheck</td>
</tr>
<tr>
<td>LIST_TRAN_PURCHORD</td>
<td>Purchase Orders</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=PurchOrd</td>
</tr>
<tr>
<td>LIST_TRAN_PURCHORD_REQ</td>
<td>View Purchase Requests/Orders</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=PurchOrd</td>
</tr>
<tr>
<td>LIST_TRAN_REORDER</td>
<td>Items Ordered</td>
<td>/app/common/item/itemsordered.nl</td>
</tr>
<tr>
<td>LIST_TRAN_REVCOMRV</td>
<td>View Revenue Commitment Reversals</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=RevComRv</td>
</tr>
<tr>
<td>LIST_TRAN_RTNAUTH</td>
<td>Return Authorizations</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=RtnAuth</td>
</tr>
<tr>
<td>LIST_TRAN_SALESORD</td>
<td>Sales Orders</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=SalesOrd</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>LIST_TRAN_TAXLIAB</td>
<td>Tax Liabilities</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TaxLiab</td>
</tr>
<tr>
<td>LIST_TRAN_TAXPYMT</td>
<td>Tax Payments</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TaxPymt</td>
</tr>
<tr>
<td>LIST_TRAN_TEGPYBL</td>
<td>Issued Tegatas</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TegPybl</td>
</tr>
<tr>
<td>LIST_TRAN_TEGRCVBL</td>
<td>Received Tegatas</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TegRcvbl</td>
</tr>
<tr>
<td>LIST_TRAN_TRANSFER</td>
<td>Bank Transfers</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Transfer</td>
</tr>
<tr>
<td>LIST_TRAN_TRNFRORD</td>
<td>Transfer Orders</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TrnfrOrd</td>
</tr>
<tr>
<td>LIST_TRAN_UNBUILD</td>
<td>Assembly Unbuilds</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Unbuild</td>
</tr>
<tr>
<td>LIST_TRAN_VATLIAB</td>
<td>Tax Liabilities</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TaxLiab</td>
</tr>
<tr>
<td>LIST_TRAN_VATLIABAU</td>
<td>GST Liabilities</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TaxLiab</td>
</tr>
<tr>
<td>LIST_TRAN_VATLIABUK</td>
<td>VAT Liabilities</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TaxLiab</td>
</tr>
<tr>
<td>LIST_TRAN_VENDAUTH</td>
<td>Vendor Return Authorizations</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=VendAuth</td>
</tr>
<tr>
<td>LIST_TRAN_VENDBILL</td>
<td>Bills</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=VendBill</td>
</tr>
<tr>
<td>LIST_TRAN_VENDCRED</td>
<td>Bill Credits</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=VendCred</td>
</tr>
<tr>
<td>LIST_TRAN_VENDPYMT</td>
<td>Bill Payments</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=VendPymt</td>
</tr>
<tr>
<td>LIST_TRAN_WORKORD</td>
<td>Work Orders</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=WorkOrd</td>
</tr>
<tr>
<td>LIST_TRAN_YTDADJST</td>
<td>Payroll Adjustment</td>
<td>/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=YtdAdjst</td>
</tr>
<tr>
<td>LIST_UNCATSITEITEM</td>
<td>View Uncategorized Items</td>
<td>/app/setup/uncatsiteitems.nl</td>
</tr>
<tr>
<td>LIST_UNITSTYPE</td>
<td>Units Of Measure</td>
<td>/app/common/units/unitstypelist.nl</td>
</tr>
<tr>
<td>LIST_UPSELLPOPUP</td>
<td>Upsell Popup</td>
<td>/app/crm/sales/upsell/upsellpopup.nl</td>
</tr>
<tr>
<td>LIST_UPSELLWIZARD</td>
<td>Upsell Manager</td>
<td>/app/crm/sales/upsell/upsellmanager.nl</td>
</tr>
<tr>
<td>LIST_URLALIASES</td>
<td>Promotional URLs</td>
<td>/app/setup/urlaliases.nl</td>
</tr>
<tr>
<td>LIST_USER</td>
<td>Manage Users</td>
<td>/app/setup/listusers.nl</td>
</tr>
<tr>
<td>LIST_VENDOR</td>
<td>Vendors</td>
<td>/app/common/entity/vendorlist.nl</td>
</tr>
<tr>
<td>LIST_WORKCALENDAR</td>
<td>Work Calendars</td>
<td>/app/accounting/project/workcalendars.nl</td>
</tr>
<tr>
<td>LIST_WORKFLOW</td>
<td>Workflows</td>
<td>/app/common/workflow/setup/workflowlist.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>LIST_WORKPLACE</td>
<td>Workplaces</td>
<td>/app/common/otherlists/workplacelist.nl</td>
</tr>
<tr>
<td>SRCH_ACCOUNT</td>
<td>Search Account</td>
<td>/app/common/search/search.nl?searchtype=Account</td>
</tr>
<tr>
<td>SRCH_ACTIVITY</td>
<td>Search Activities</td>
<td>/app/common/search/search.nl?searchtype=Activity</td>
</tr>
<tr>
<td>SRCH_AMORTIZATIONSCSCHDULE</td>
<td>Search Amortization Schedules</td>
<td>/app/common/search/search.nl?searchtype=AmortizationSchedule</td>
</tr>
<tr>
<td>SRCH_BIN</td>
<td>Search Bin</td>
<td>/app/common/search/search.nl?searchtype=BinNumber</td>
</tr>
<tr>
<td>SRCH_BUDGET</td>
<td>Search Set Up Budget</td>
<td>/app/common/search/search.nl?searchtype=Budget</td>
</tr>
<tr>
<td>SRCH_BUDGETRATES</td>
<td>Search Budget Rates</td>
<td>/app/common/search/search.nl?searchtype=Budget</td>
</tr>
<tr>
<td>SRCHCALENDAR</td>
<td>Search Calendar/Event</td>
<td>/app/common/search/search.nl?searchtype=Calendar</td>
</tr>
<tr>
<td>SRCHCALL</td>
<td>Search Calls</td>
<td>/app/common/search/search.nl?searchtype=Call</td>
</tr>
<tr>
<td>SRCH_CAMPAIGN</td>
<td>Search Campaigns</td>
<td>/app/common/search/search.nl?searchtype=Campaign</td>
</tr>
<tr>
<td>SRCH_CASE</td>
<td>Search Case</td>
<td>/app/common/search/search.nl?searchtype=Case</td>
</tr>
<tr>
<td>SRCH_CLASS</td>
<td>Search Class</td>
<td>/app/common/search/search.nl?searchtype=Class</td>
</tr>
<tr>
<td>SRCH_COMPANY</td>
<td>Search All Companies</td>
<td>/app/common/search/search.nl?searchtype=Company</td>
</tr>
<tr>
<td>SRCHCOMPETITOR</td>
<td>Search Competitors</td>
<td>/app/common/search/search.nl?searchtype=Competitor</td>
</tr>
<tr>
<td>SRCH_CONSOLEXCHANGERATE</td>
<td>Consolidated Exchange Rates</td>
<td>/app/common/search/search.nl?searchtype=ConsolExchangeRate</td>
</tr>
<tr>
<td>SRCHCONSOLRATES</td>
<td>Search Consolidated Rates</td>
<td>/app/common/search/search.nl?searchtype=ConsolExchangeRate</td>
</tr>
<tr>
<td>SRCHCONTACT</td>
<td>Search Contact</td>
<td>/app/common/search/search.nl?searchtype=Contact</td>
</tr>
<tr>
<td>SRCHCRMGROUP</td>
<td>Search Group</td>
<td>/app/common/search/search.nl?searchtype=CRMGroup</td>
</tr>
<tr>
<td>SRCHCUSTOMER</td>
<td>Search Customer</td>
<td>/app/common/search/search.nl?searchtype=Customer</td>
</tr>
<tr>
<td>SRCHDEPARTMENT</td>
<td>Search Department</td>
<td>/app/common/search/search.nl?searchtype=Department</td>
</tr>
<tr>
<td>SRCHDOCUMENT</td>
<td>Search File Cabinet</td>
<td>/app/common/search/search.nl?searchtype=Document</td>
</tr>
<tr>
<td>SRCHEMPLOYEE</td>
<td>Search Employee</td>
<td>/app/common/search/search.nl?searchtype=Employee</td>
</tr>
<tr>
<td>SRCHENTITY</td>
<td>Search All Entities</td>
<td>/app/common/search/search.nl?searchtype=Entity</td>
</tr>
<tr>
<td>SRCHFIRSTVISIT</td>
<td>Search First Site Visit</td>
<td>/app/common/search/search.nl?searchtype=FirstVisit</td>
</tr>
<tr>
<td>SRCHGIFTCERTIFICATE</td>
<td>Search Gift Certificates</td>
<td>/app/common/search/search.nl?searchtype=GiftCertificate</td>
</tr>
<tr>
<td>SRCHINFOITEM</td>
<td>Search Information Items</td>
<td>/app/common/search/search.nl?id=-2540&amp;e=T</td>
</tr>
<tr>
<td>SRCHINFOITEMFORM</td>
<td>Search Published Forms</td>
<td>/app/common/search/search.nl?id=-2541&amp;e=T</td>
</tr>
<tr>
<td>SRCHINVENTORYNUMBER</td>
<td>Search Inventory Number</td>
<td>/app/common/search/search.nl?searchtype=InventoryNumber</td>
</tr>
<tr>
<td>SRCHISSUE</td>
<td>Search Issue</td>
<td>/app/common/search/search.nl?searchtype=Issue</td>
</tr>
<tr>
<td>SRCHITEM</td>
<td>Search Item</td>
<td>/app/common/search/search.nl?searchtype=Item</td>
</tr>
<tr>
<td>SRCHJOB</td>
<td>Search Job</td>
<td>/app/common/search/search.nl?searchtype=Job</td>
</tr>
<tr>
<td>SRCHLEAD</td>
<td>Search Leads</td>
<td>/app/common/search/search.nl?searchtype=Customer&amp;Customer_STAGE=LEAD</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>SRCH_LOCATION</td>
<td>Search Location</td>
<td>/app/common/search/search.nl?searchtype=Location</td>
</tr>
<tr>
<td>SRCH_LOGINAUDIT</td>
<td>Search Login Audit Trail</td>
<td>/app/common/search/search.nl?searchtype=LoginAudit</td>
</tr>
<tr>
<td>SRCH_MEDIAITEM</td>
<td>Search Media Items</td>
<td>/app/common/search/search.nl?searchtype=Document</td>
</tr>
<tr>
<td>SRCH_MEMDOCM</td>
<td>Search Memorized Transactions</td>
<td>/app/common/search/search.nl?searchtype=MemDoc</td>
</tr>
<tr>
<td>SRCH_PARTNER</td>
<td>Search Partner</td>
<td>/app/common/search/search.nl?searchtype=Partner</td>
</tr>
<tr>
<td>SRCH_PRESCATEGORY</td>
<td>Search Categories</td>
<td>/app/common/search/search.nl?searchtype=SiteCategory</td>
</tr>
<tr>
<td>SRCH_PRICING</td>
<td>Search Pricing</td>
<td>/app/common/search/search.nl?searchtype=Pricing</td>
</tr>
<tr>
<td>SRCH_PROJECTRESOURCE</td>
<td>Search Project Resources</td>
<td>/app/common/search/search.nl?searchtype=ProjectResource</td>
</tr>
<tr>
<td>SRCH_PROJECTTASK</td>
<td>Search Project Tasks</td>
<td>/app/common/search/search.nl?searchtype=ProjectTask</td>
</tr>
<tr>
<td>SRCH_PROJECTTASKANDCRMTASK</td>
<td>Search Project Tasks and CRM Tasks</td>
<td>/app/common/search/search.nl?searchtype=ProjectTaskAndCrmTask</td>
</tr>
<tr>
<td>SRCH_PROMOTION</td>
<td>Search Promotion Code</td>
<td>/app/common/search/search.nl?searchtype=Promotion</td>
</tr>
<tr>
<td>SRCH_PROSPECT</td>
<td>Search Prospects</td>
<td>/app/common/search/search.nl?searchtype=Prospect</td>
</tr>
<tr>
<td>SRCH_QUOTA</td>
<td>Search Establish Quota</td>
<td>/app/common/search/search.nl?searchtype=Quota</td>
</tr>
<tr>
<td>SRCH_REVREC</td>
<td>Search Revenue Recognition Schedules</td>
<td>/app/common/search/search.nl?searchtype=RevRec</td>
</tr>
<tr>
<td>SRCH_REVRECOGNITIONSCHED</td>
<td>Search Revenue Recognition Schedules</td>
<td>/app/common/search/search.nl?searchtype=RevRecognitionSched</td>
</tr>
<tr>
<td>SRCH_ROLE</td>
<td>Search Role</td>
<td>/app/common/search/search.nl?searchtype=Role</td>
</tr>
<tr>
<td>SRCH_SALESCAMPAIGN</td>
<td>Search Sales Campaigns</td>
<td>/app/common/search/search.nl?searchtype=SalesCampaign</td>
</tr>
<tr>
<td>SRCH_SALETERRITORIES</td>
<td>Sales Territory Search</td>
<td>/app/common/search/search.nl?searchtype=SalesTerritory</td>
</tr>
<tr>
<td>SRCH_SAVEDSEARCH</td>
<td>All Saved Searches</td>
<td>/app/common/search/savedsearches.nl</td>
</tr>
<tr>
<td>SRCH_SCRIPTNOTE</td>
<td>Search Script Execution Log</td>
<td>/app/common/search/search.nl?searchtype=ScriptNote</td>
</tr>
<tr>
<td>SRCH_SHOPPINGCART</td>
<td>Search Shopping Cart</td>
<td>/app/common/search/search.nl?searchtype=ShoppingCart</td>
</tr>
<tr>
<td>SRCH_SOLUTION</td>
<td>Search Solution</td>
<td>/app/common/search/search.nl?searchtype=Solution</td>
</tr>
<tr>
<td>SRCH_SUBSIDIARY</td>
<td>Search Subsidiary</td>
<td>/app/common/search/search.nl?searchtype=Subsidiary</td>
</tr>
<tr>
<td>SRCH_SYSTEMNOTE</td>
<td>Search System Notes</td>
<td>/app/common/search/search.nl?searchtype=SystemNote</td>
</tr>
<tr>
<td>SRCH_TASK</td>
<td>Search Tasks</td>
<td>/app/common/search/search.nl?searchtype=Task</td>
</tr>
<tr>
<td>SRCH_TAXBROUP</td>
<td>Search Tax Group</td>
<td>/app/common/search/search.nl?searchtype=TaxGroup</td>
</tr>
<tr>
<td>SRCH_TAXITEM</td>
<td>Search Tax Code</td>
<td>/app/common/search/search.nl?searchtype=TaxItem</td>
</tr>
<tr>
<td>SRCH_TIME</td>
<td>Search Track Time</td>
<td>/app/common/search/search.nl?searchtype=Time</td>
</tr>
<tr>
<td>SRCH_TOPIC</td>
<td>Search Topic</td>
<td>/app/common/search/search.nl?searchtype=Topic</td>
</tr>
<tr>
<td>SRCH_TRANSACTION</td>
<td>Search Transactions</td>
<td>/app/common/search/search.nl?searchtype=Transaction</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>SRCH_TRAN_BINTRNFR</td>
<td>Search Bin Transfers</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=BinTrnfr</td>
</tr>
<tr>
<td>SRCH_TRAN_BINWKSHT</td>
<td>Search Bin Putaway Worksheets</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=BinWksht</td>
</tr>
<tr>
<td>SRCH_TRAN_BUILD</td>
<td>Search Assembly Builds</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=Build</td>
</tr>
<tr>
<td>SRCH_TRAN_CARDCHRG</td>
<td>Search Credit Card Charges</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=CardChrg</td>
</tr>
<tr>
<td>SRCH_TRAN_CASHRFND</td>
<td>Search Cash Refunds</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=CashRfnd</td>
</tr>
<tr>
<td>SRCH_TRAN_CASHSALE</td>
<td>Search Cash Sales</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=CashSale</td>
</tr>
<tr>
<td>SRCH_TRAN_CHECK</td>
<td>Search Checks</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=Check</td>
</tr>
<tr>
<td>SRCH_TRAN_COMMISSN</td>
<td>Search Individual Employee Commissions</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=Commissn</td>
</tr>
<tr>
<td>SRCH_TRAN_CUSTCHRG</td>
<td>Search Statement Charges</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=CustChrg</td>
</tr>
<tr>
<td>SRCH_TRAN_CUSTCRED</td>
<td>Search Credit Memos</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=CustCred</td>
</tr>
<tr>
<td>SRCH_TRAN_CUSTDEP</td>
<td>Search Customer Deposits</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=CustDep</td>
</tr>
<tr>
<td>SRCH_TRAN_CUSTINVC</td>
<td>Search Invoices</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=CustInvc</td>
</tr>
<tr>
<td>SRCH_TRAN_CUSTPYMT</td>
<td>Search Customer Payments</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=CustPymt</td>
</tr>
<tr>
<td>SRCH_TRAN_CUSTRFND</td>
<td>Search Customer Refunds</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=CustRfnd</td>
</tr>
<tr>
<td>SRCH_TRAN_DEPAPPL</td>
<td>Search Deposit Applications</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=DepAppl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>SRCH_TRAN_DEPOSIT</td>
<td>Search Deposits</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=Deposit</td>
</tr>
<tr>
<td>SRCH_TRAN_ESTIMATE</td>
<td>Search Estimates</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=Estimate</td>
</tr>
<tr>
<td>SRCH_TRAN_EXPREPT</td>
<td>Search Expense Reports</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=ExpenseReport</td>
</tr>
<tr>
<td>SRCH_TRAN_FXREVAL</td>
<td>Search Currency Revaluations</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=CurrencyReval</td>
</tr>
<tr>
<td>SRCH_TRAN_INVADJST</td>
<td>Search Inventory Adjustments</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=InventoryAdjustment</td>
</tr>
<tr>
<td>SRCH_TRAN_INVDISTR</td>
<td>Search Inventory Distributions</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=InventoryDistribution</td>
</tr>
<tr>
<td>SRCH_TRAN_INVTRNFR</td>
<td>Search Inventory Transfers</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=InventoryTransfer</td>
</tr>
<tr>
<td>SRCH_TRAN_INVWKSHT</td>
<td>Search Inventory Worksheets</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=InventoryWorksheet</td>
</tr>
<tr>
<td>SRCH_TRAN_ITEMRCPT</td>
<td>Search Item Receipts</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=ItemReceipt</td>
</tr>
<tr>
<td>SRCH_TRAN_ITEMSHIP</td>
<td>Search Item Fulfillments</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=ItemShip</td>
</tr>
<tr>
<td>SRCH_TRAN_JOURNAL</td>
<td>Search Journal Entries</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=Journal</td>
</tr>
<tr>
<td>SRCH_TRAN_LIABPYMT</td>
<td>Search Liability Payments</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=LiabilityPayment</td>
</tr>
<tr>
<td>SRCH_TRAN_OPPRTNTY</td>
<td>Search Opportunities</td>
<td>/app/common/search/search.nl?searchtype=Opportunity</td>
</tr>
<tr>
<td>SRCH_TRAN_PAYCHECK</td>
<td>Search Paychecks</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=Paycheck</td>
</tr>
<tr>
<td>SRCH_TRAN_PURCHORD</td>
<td>Search Purchase Orders</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=PurchaseOrder</td>
</tr>
<tr>
<td>SRCH_TRAN_REVCOMM</td>
<td>Search Revenue Commitments</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=RevenueCommitment</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>SRCH_TRAN_REVCOMRV</td>
<td>Search View Revenue Commitment Reversals</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=RevComRv</td>
</tr>
<tr>
<td>SRCH_TRAN_RTNAUTH</td>
<td>Search Return Authorizations</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=RtnAuth</td>
</tr>
<tr>
<td>SRCH_TRAN_SALESORD</td>
<td>Search Sales Orders</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=SalesOrd</td>
</tr>
<tr>
<td>SRCH_TRAN_TAXLIAB</td>
<td>Search Tax Liabilities</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=TaxLIab</td>
</tr>
<tr>
<td>SRCH_TRAN_TAXPYMT</td>
<td>Search Tax Payments</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=TaxPYmt</td>
</tr>
<tr>
<td>SRCH_TRAN_TEGPYBL</td>
<td>Search Issued Tegatas</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=TegPYbl</td>
</tr>
<tr>
<td>SRCH_TRAN_TEGRCVBL</td>
<td>Search Received Tegatas</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=TegRCVbl</td>
</tr>
<tr>
<td>SRCH_TRAN_TRANSFER</td>
<td>Search Bank Transfers</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=Transfer</td>
</tr>
<tr>
<td>SRCH_TRAN_TRNFRORD</td>
<td>Search Transfer Orders</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=TrnfrOrd</td>
</tr>
<tr>
<td>SRCH_TRAN_UNBUILD</td>
<td>Search Assembly Unbuilds</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=Unbuild</td>
</tr>
<tr>
<td>SRCH_TRAN_VENDAUTH</td>
<td>Search Vendor Return Authorizations</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=VendorAuth</td>
</tr>
<tr>
<td>SRCH_TRAN_VENDBILL</td>
<td>Search Bills</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=VendorBill</td>
</tr>
<tr>
<td>SRCH_TRAN_VENCRED</td>
<td>Search Bill Credits</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=VendorCred</td>
</tr>
<tr>
<td>SRCH_TRAN_VENDPYMT</td>
<td>Search Bill Payments</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=VendorPYmt</td>
</tr>
<tr>
<td>SRCH_TRAN_WORKORD</td>
<td>Search Work Orders</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=WorkOrd</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>SRCH_TRAN_YTDADJST</td>
<td>Search Payroll Adjustment</td>
<td>/app/common/search/search.nl?searchtype=Transaction&amp;Transaction_TYPE=YtdAdjst</td>
</tr>
<tr>
<td>SRCH_UNITTYPE</td>
<td>Search Unit of Measure</td>
<td>/app/common/search/search.nl?searchtype=UnitsType</td>
</tr>
<tr>
<td>SRCH_USERNOTE</td>
<td>Search User Notes</td>
<td>/app/common/search/search.nl?searchtype=UserNote</td>
</tr>
<tr>
<td>SRCH_VENDOR</td>
<td>Search Vendor</td>
<td>/app/common/search/search.nl?searchtype=Vendor</td>
</tr>
<tr>
<td>SRCH_WORKCALENDAR</td>
<td>Search Work Calendar</td>
<td>/app/common/search/search.nl?searchtype=WorkCalendar</td>
</tr>
<tr>
<td>SRCH_WORKFLOW</td>
<td>Search Workflow</td>
<td>/app/common/search/search.nl?searchtype=Workflow</td>
</tr>
<tr>
<td>SRCH_WORKFLOWINSTANCE</td>
<td>Search Workflow Instances</td>
<td>/app/common/search/search.nl?searchtype=WorkflowInstance</td>
</tr>
<tr>
<td>SRCH_WORKPLACE</td>
<td>Search Workplace</td>
<td>/app/common/search/search.nl?searchtype=Workplace</td>
</tr>
<tr>
<td>SUPT_ALLOW_LOGIN</td>
<td>NetSuite Support Login</td>
<td>/app/crm/support/allowsupportlogin.nl</td>
</tr>
<tr>
<td>SUPT_BUG_BILLING</td>
<td>Billing Questions</td>
<td>&quot;javascript:window.open(&quot;&quot;/app/crm/support/nlcorpsupport.nl?type=billing'&quot;&quot;,&quot;&quot;messagepopup&quot;&quot;,&quot;&quot;resizable=yes,scrollbars=yes,width=800,height=700&quot;&quot; );void(0)&quot;&quot;)</td>
</tr>
<tr>
<td>SUPT_BUG_SUPPORT</td>
<td>Contact Support Online</td>
<td>&quot;javascript:window.open(&quot;&quot;/app/crm/support/nlcorpsupport.nl?type=support'&quot;&quot;,&quot;&quot;messagepopup&quot;&quot;,&quot;&quot;resizable=yes,scrollbars=yes,width=800,height=700&quot;&quot; );void(0)&quot;&quot;)</td>
</tr>
<tr>
<td>SUPT_BUG_SUPPORT_POPUP</td>
<td>Contact Support By Phone</td>
<td>&quot;javascript:window.open(&quot;&quot;/app/crm/support/contactus.nl?l=T'&quot;&quot;,&quot;&quot;messagepopup&quot;&quot;,&quot;&quot;scrollbars,width=450,height=400&quot;&quot; );void(0)&quot;&quot;)</td>
</tr>
<tr>
<td>SUPT_CENTER_ROLE</td>
<td>NetSuite Support Center</td>
<td>/app/login/dashboard.nl?id=</td>
</tr>
<tr>
<td>SUPT GENERICHELP</td>
<td>Generic Help for related topics</td>
<td>&quot;javascript:nlPopupHelp(&quot;&quot;UG_Glossary&quot;&quot;,&quot;&quot;userGuides&quot;)&quot;</td>
</tr>
<tr>
<td>SUPT_GLOSSARY</td>
<td>Glossary</td>
<td>&quot;javascript:nlPopupHelp(&quot;&quot;UG_Glossary&quot;&quot;,&quot;&quot;userGuides&quot;)&quot;</td>
</tr>
<tr>
<td>SUPT_GUIDES</td>
<td>User Guides</td>
<td>&quot;javascript:nlPopupHelp(&quot;&quot;userGuides&quot;&quot;,&quot;&quot;userGuides&quot;)&quot;</td>
</tr>
<tr>
<td>SUPT_HELP</td>
<td>Help</td>
<td>&quot;javascript:nlPopupHelp(&quot;&quot;help&quot;&quot;,&quot;&quot;help&quot;)&quot;</td>
</tr>
<tr>
<td>SUPT_MENUOPTIONS</td>
<td>Customer Support Menu Options</td>
<td>&quot;javascript:window.open(&quot;&quot;/core/help/helppdf.nl?id=309&quot;&quot; );void(0)&quot;&quot;&quot;)</td>
</tr>
<tr>
<td>SUPT_NSCENTRAL</td>
<td>NetSuite Central</td>
<td>&quot;javascript:window.open(&quot;&quot;/app/site/backend/login/shoppingsecureloginredirect.nl?company=NLCORP&amp;nscentral=T&amp;redirect=<a href="https://central.netsuite.com/index.html">https://central.netsuite.com/index.html</a>&quot;&quot; );void(0)&quot;&quot;&quot;)</td>
</tr>
<tr>
<td>SUPT_POLICYFORM</td>
<td>Support Policies</td>
<td>&quot;javascript:nlPopupHelp(&quot;&quot;SUPPORT_POLICY&quot;&quot;,&quot;&quot;help&quot;)&quot;</td>
</tr>
<tr>
<td>SUPTRELEASENOTES</td>
<td>Release Notes</td>
<td>&quot;javascript:nlPopupHelp(&quot;&quot;relNotes&quot;&quot;,&quot;&quot;help&quot;)&quot;</td>
</tr>
<tr>
<td>SUPT_SNEAKPEAK9_5</td>
<td>Version 9.5 Updates</td>
<td>&quot;javascript:showSneakPeek(&quot;&quot;9_5&quot;&quot; );void(0)&quot;</td>
</tr>
<tr>
<td>SUPT_VERSION_10_5</td>
<td>Version 10.6</td>
<td>&quot;javascript:nlPopupHelp(&quot;&quot;sneakPeeks&quot;&quot;,&quot;&quot;help&quot;)&quot;</td>
</tr>
<tr>
<td>TRAN_ACTIVITY</td>
<td>Activity</td>
<td>/app/crm/calendar/activity.nl</td>
</tr>
<tr>
<td>TRAN_ADDCONTENT</td>
<td>Add Content</td>
<td>/app/pages/addcontent.nl</td>
</tr>
<tr>
<td>TRAN_ADDSHORTCUT</td>
<td>Short Cuts</td>
<td>/core/pages/addShortcut.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td>TRAN_ALLOCATEPAYCHECKSTOJOBS</td>
<td>Allocate Paycheck Expenses to Jobs</td>
<td>/app/accounting/transactions/allocatepaycheckstojobs.nl</td>
</tr>
<tr>
<td>TRAN_APPROVAL_EXPREPT</td>
<td>Approve Expense Reports</td>
<td>/app/accounting/transactions/approval.nl?type=exprept&amp;label=Expense%20Report</td>
</tr>
<tr>
<td>TRAN_APPROVAL_PURCHORD</td>
<td>Approve Purchase Requests</td>
<td>/app/accounting/transactions/approval.nl?type=purchord&amp;label=Purchase%20Request</td>
</tr>
<tr>
<td>TRAN_APPROVECOMMISSN</td>
<td>Approve Employee Commissions</td>
<td>/app/accounting/transactions/approvecommissn.nl</td>
</tr>
<tr>
<td>TRAN_APPROVEPARTNERCOMMISSN</td>
<td>Approve Partner Commissions</td>
<td>/app/accounting/transactions/approvepartnercommissn.nl</td>
</tr>
<tr>
<td>TRAN_AUDIT</td>
<td>View Audit Trail</td>
<td>/app/accounting/transactions/audit.nl</td>
</tr>
<tr>
<td>TRAN_BANKRECON</td>
<td>Find Matching Transactions</td>
<td>/app/accounting/transactions/bankrecon.nl</td>
</tr>
<tr>
<td>TRAN_BANKVIEW</td>
<td>Online Banking Statement</td>
<td>/app/accounting/transactions/bankview.nl</td>
</tr>
<tr>
<td>TRAN_BAS</td>
<td>Business Activity Statement</td>
<td>/app/accounting/reports/intl/bas.nl</td>
</tr>
<tr>
<td>TRAN_BILLPAY_LOG</td>
<td>Job Status</td>
<td>/app/external/xml/upload/uploadlog.nl?displayType=BILLPAY</td>
</tr>
<tr>
<td>TRAN_BUDGET</td>
<td>Set Up Budgets</td>
<td>/app/accounting/transactions/budgets.nl</td>
</tr>
<tr>
<td>TRAN_BULKBILL_LOG</td>
<td>Job Status</td>
<td>/app/external/xml/upload/uploadlog.nl?displayType=BULKBILL</td>
</tr>
<tr>
<td>TRAN_BULKCOMMITREVENUE_LOG</td>
<td>Job Status</td>
<td>/app/external/xml/upload/uploadlog.nl?displayType=BULKCOMMITREVENUE</td>
</tr>
<tr>
<td>TRAN_BULKFULFILL_LOG</td>
<td>Job Status</td>
<td>/app/external/xml/upload/uploadlog.nl?displayType=BULKFULFILL</td>
</tr>
<tr>
<td>TRANCALENDARPREFERENCE</td>
<td>Calendar Preferences</td>
<td>/app/crm/calendar/calendarpreference.nl</td>
</tr>
<tr>
<td>TRAN_CAMPAIGNSETUP</td>
<td>Marketing Preferences</td>
<td>/app/setup/campaignsetup.nl</td>
</tr>
<tr>
<td>TRAN_CLEARHOLD</td>
<td>Manage Payment Holds</td>
<td>/app/accounting/transactions/salesordermanager.nl?type=clearholds</td>
</tr>
<tr>
<td>TRAN_COPY_BUDGET</td>
<td>Copy Budgets</td>
<td>/app/accounting/transactions/copybudget.nl</td>
</tr>
<tr>
<td>TRAN_CREATEAMORTIZATIONJE</td>
<td>Create Amortization Journal Entries</td>
<td>/app/accounting/transactions/createamortizationje.nl</td>
</tr>
<tr>
<td>TRAN_CREATEAMORTIZATIONJE_LOG</td>
<td>Create Journal Entries Status</td>
<td>/app/accounting/transactions/createamortizationjestatus.nl?type=AMORTIZATION</td>
</tr>
<tr>
<td>TRAN_CREATEINTERCOADJE</td>
<td>Create Intercompany Adjustment Journal Entries</td>
<td>/app/accounting/transactions/intercoadj/createintercoadjje.nl</td>
</tr>
<tr>
<td>TRAN_CREATEINTERCOADJE_LOG</td>
<td>Create Journal Entries Status</td>
<td>/app/accounting/transactions/intercoadj/createintercoadjjestatus.nl</td>
</tr>
<tr>
<td>TRAN_CREATEREVRECJE</td>
<td>Create Revenue Recognition Journal Entries</td>
<td>/app/accounting/transactions/createrevrecje.nl</td>
</tr>
<tr>
<td>TRAN_CREATEREVRECJE_LOG</td>
<td>Create Journal Entries Status</td>
<td>/app/accounting/transactions/createrevrecjestatus.nl?type=REVREC</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>TRAN_CREATE_JOBS_FROM_ORDERS</td>
<td>Create Jobs From Sales Orders</td>
<td>/app/accounting/transactions/jobcreationmanager.nl</td>
</tr>
<tr>
<td>TRAN_CREATE_WORK_ORDERS_FOR_STOCK</td>
<td>Mass Create Work Orders</td>
<td>/app/accounting/transactions/createworkordersforstock.nl</td>
</tr>
<tr>
<td>TRAN_CUSTOMIZEITEMLIST</td>
<td>Customize Item List</td>
<td>/app/accounting/transactions/customizeitemlist.nl</td>
</tr>
<tr>
<td>TRAN_CUSTOMIZETRANLIST</td>
<td>Customize Transaction List</td>
<td>/app/accounting/transactions/customizetransanlist.nl</td>
</tr>
<tr>
<td>TRAN_DEPOSITSUMMARY</td>
<td>Deposit Summary</td>
<td>/app/accounting/transactions/depositsummary.nl</td>
</tr>
<tr>
<td>TRAN_DOMAINS</td>
<td>Set Up Domains</td>
<td>/app/setup/domains.nl</td>
</tr>
<tr>
<td>TRAN_EMAILPWD</td>
<td>Change Email Password</td>
<td>/app/center/emailpwd.nl</td>
</tr>
<tr>
<td>TRAN_EMPLOYEESFA</td>
<td>Assign Reps</td>
<td>/app/common/entity/employeesfa.nl</td>
</tr>
<tr>
<td>TRAN_FINCHRG</td>
<td>Assess Finance Charges</td>
<td>/app/accounting/transactions/finchrg.nl</td>
</tr>
<tr>
<td>TRAN_FORECAST</td>
<td>Edit Sales Rep Forecast</td>
<td>/app/crm/sales/forecast.nl</td>
</tr>
<tr>
<td>TRAN_GENERATEFISCALPERIODS</td>
<td>Generate Fiscal Periods</td>
<td>/app/setup/period/generatefiscalperiods.nl</td>
</tr>
<tr>
<td>TRAN_GENERATETAXPERIODS</td>
<td>Generate Tax Periods</td>
<td>/app/setup/period/generatetaxperiods.nl</td>
</tr>
<tr>
<td>TRAN_GIFTCERTCREATEJE</td>
<td>Recognize Gift Certificate Income</td>
<td>/app/accounting/transactions/giftcertcreateje.nl</td>
</tr>
<tr>
<td>TRAN_GSTREFUND</td>
<td>Process GST/HST Refund</td>
<td>/app/accounting/transactions/gstrefund.nl</td>
</tr>
<tr>
<td>TRAN_HISTORY</td>
<td>History</td>
<td>/app/accounting/transactions/history.nl</td>
</tr>
<tr>
<td>TRAN_IMPACT</td>
<td>GL Impact</td>
<td>/app/accounting/transactions/impact.nl</td>
</tr>
<tr>
<td>TRAN_INVOICECUSTOMERS</td>
<td>Invoice Billable Customers</td>
<td>/app/accounting/transactions/invoicecustomers.nl</td>
</tr>
<tr>
<td>TRAN_ITEMSHIPPACK</td>
<td>Mark Orders Packed</td>
<td>/app/accounting/transactions/itemshipmanager.nl?type=pack</td>
</tr>
<tr>
<td>TRAN_ITEMSHIPSHIP</td>
<td>Mark Orders Shipped</td>
<td>/app/accounting/transactions/itemshipmanager.nl?type=ship</td>
</tr>
<tr>
<td>TRAN_JOURNALAPPROVAL</td>
<td>Approve Journal Entries</td>
<td>/app/accounting/transactions/journalapproval.nl</td>
</tr>
<tr>
<td>TRANLOGINAUDIT</td>
<td>Login Audit</td>
<td>/app/setup/loginAudit.nl</td>
</tr>
<tr>
<td>TRAN_MARKVSOEDELIVERED</td>
<td>Mark VSOE Delivered Status</td>
<td>/app/accounting/transactions/revrec/markvsoedelivered.nl</td>
</tr>
<tr>
<td>TRAN_MGRFORECAST</td>
<td>Edit Sales Manager Forecast</td>
<td>/app/crm/sales/mgrforecast.nl</td>
</tr>
<tr>
<td>TRAN_NLVENDOR</td>
<td>Vendor Center</td>
<td>/app/center/nlvendor.nl</td>
</tr>
<tr>
<td>TRAN_OPENBAL</td>
<td>Enter Opening Balances</td>
<td>/app/accounting/transactions/openbal.nl</td>
</tr>
<tr>
<td>TRAN_ORDERITEMS</td>
<td>Order Items</td>
<td>/app/accounting/transactions/orderitems.nl</td>
</tr>
<tr>
<td>TRAN_PAYMENTS</td>
<td>Payments</td>
<td>/app/accounting/transactions/payments.nl</td>
</tr>
<tr>
<td>TRAN_PAYROLLBATCH</td>
<td>Payroll Batch</td>
<td>/app/payroll/payrollbatch.nl</td>
</tr>
<tr>
<td>TRAN_PAYROLLRUN</td>
<td>Create Payroll</td>
<td>/app/accounting/transactions/payroll/payrollrun.nl</td>
</tr>
<tr>
<td>TRAN_PAYROLLSTATUS</td>
<td>View Payroll Status</td>
<td>/app/payroll/cdstatus.nl</td>
</tr>
<tr>
<td>TRAN_PDF_F940</td>
<td>Annual Federal Unemployment (940)</td>
<td>/app/accounting/reports/taxformfederal.nl?formtype=f940</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>TRAN_PDF_F941</td>
<td>Quarterly Federal Tax Return (941)</td>
<td>/app/accounting/reports/taxformfederal.nl?formtype=f941</td>
</tr>
<tr>
<td>TRAN_POSTVENDORBILLVARIANCES</td>
<td>Post Vendor Bill Variances</td>
<td>/app/accounting/transactions/vendorbillvariance/postvendorbillvariances.nl</td>
</tr>
<tr>
<td>TRAN_POSTVENDORBILLVARIANCES_STATUS</td>
<td>Post Vendor Bill Variances Status</td>
<td>/app/accounting/transactions/vendorbillvariance/postvendorbillvariancesstatus.nl</td>
</tr>
<tr>
<td>TRAN_PREVIEWW2</td>
<td>Form W-2 Preview</td>
<td>/app/accounting/reports/w2preview.nl</td>
</tr>
<tr>
<td>TRAN_PRINT</td>
<td>Print Checks and Forms</td>
<td>/app/accounting/print/print.nl</td>
</tr>
<tr>
<td>TRAN_PRINT1096</td>
<td>Form 1096</td>
<td>/app/accounting/reports/NLPrint1096s.nl?mode=frame</td>
</tr>
<tr>
<td>TRAN_PRINT1099</td>
<td>Form 1099-MISC</td>
<td>/app/accounting/reports/NLPrint1099s.nl?mode=frame</td>
</tr>
<tr>
<td>TRAN_PRINTBARCODES</td>
<td>Generate Barcodes</td>
<td>/app/accounting/print/printbarcodes.nl?printtype=null&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINTMAILINGLABELS</td>
<td>Print Mailing Labels</td>
<td>/app/accounting/print/printmailinglabels.nl?printtype=null&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINTPRICELIST</td>
<td>Individual Price List</td>
<td>/app/accounting/print/printpricelist.nl</td>
</tr>
<tr>
<td>TRAN_PRINTSTATEMENT</td>
<td>Individual Statement</td>
<td>/app/accounting/print/printstatement.nl</td>
</tr>
<tr>
<td>TRAN_PRINTW2</td>
<td>Form W-2</td>
<td>/app/accounting/reports/NLPrintW2s.nl?mode=frame</td>
</tr>
<tr>
<td>TRAN_PRINTW2AUDIT</td>
<td>W-2 and 1099 Audit Information</td>
<td>/app/accounting/reports/printw2audit.nl?mode=frame</td>
</tr>
<tr>
<td>TRAN_PRINTW3</td>
<td>Form W-3</td>
<td>/app/accounting/reports/NLPrintW3s.nl?mode=frame</td>
</tr>
<tr>
<td>TRAN_PRINT_BOM</td>
<td>Print Bills of Materials</td>
<td>/app/accounting/print/printframe.nl?trantype=workord&amp;printtype=bom&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_CASHSALE</td>
<td>Print Receipts</td>
<td>/app/accounting/print/printframe.nl?trantype=cashsale&amp;printtype=transaction&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_CHECK</td>
<td>Print Checks</td>
<td>/app/accounting/print/printframe.nl?trantype=check&amp;printtype=transaction&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_COMMERCIALINVOICE</td>
<td>Print Commerical Invoice</td>
<td>/app/accounting/print/commericalinvoice.nl</td>
</tr>
<tr>
<td>TRAN_PRINT_CUSTCRED</td>
<td>Print Credits Memos</td>
<td>/app/accounting/print/printframe.nl?trantype=custcred&amp;printtype=transaction&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_CUSTINVC</td>
<td>Print Invoices</td>
<td>/app/accounting/print/printframe.nl?trantype=custinvc&amp;printtype=transaction&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_ESTIMATE</td>
<td>Print Estimates</td>
<td>/app/accounting/print/printframe.nl?trantype=estimate&amp;printtype=transaction&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_INTEGRATEDSHIPPINGLABEL</td>
<td>Print Intetgrated Shipping Labels</td>
<td>/app/accounting/print/printlabels.nl?printtype=integratedshippinglabel&amp;method=print&amp;title=Integrated Shipping Labels</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>TRAN_PRINT_ITEM_DETAIL_STATEMENT</td>
<td>Generate Item Detail Statements</td>
<td>/app/accounting/print/printframe.nl?trantype=&amp;printtype=itemdetailstatement&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_ONE_ITEM_DETAIL_STATEMENT</td>
<td>Individual Item Detail Statement</td>
<td>/app/accounting/print/printitemdetaillstatement.nl</td>
</tr>
<tr>
<td>TRAN_PRINT_PACKINGSLIP</td>
<td>Print Packing Slips</td>
<td>/app/accounting/print/printframe.nl?trantype=&amp;printtype=packingslip&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_PAYCHECK</td>
<td>Print Paychecks</td>
<td>/app/accounting/print/printframe.nl?trantype=paycheck&amp;printtype=transaction&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_PAYMENTVOUCHER</td>
<td>Print Payment Vouchers</td>
<td>/app/accounting/print/printframe.nl?trantype=vendpymt&amp;printtype=paymentvoucher&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_PICKING_TICKET</td>
<td>Print Picking Tickets</td>
<td>/app/accounting/print/printframe.nl?trantype=salesord&amp;printtype=pickingticket&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_PRICELIST</td>
<td>Generate Price Lists</td>
<td>/app/accounting/print/printframe.nl?trantype=&amp;printtype=pricelist&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_PURCHORD</td>
<td>Print Purchase Orders</td>
<td>/app/accounting/print/printframe.nl?trantype=purchord&amp;printtype=transaction&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_RTNAUTH</td>
<td>Return Authorizations</td>
<td>/app/accounting/print/printform.nl?printtype=transaction&amp;trantype=rtnauth&amp;method=print&amp;title=Return%20Authorizations</td>
</tr>
<tr>
<td>TRAN_PRINT_SALESORD</td>
<td>Print Sales Orders</td>
<td>/app/accounting/print/printframe.nl?trantype=salesord&amp;printtype=transaction&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_SHIPPINGLABEL</td>
<td>Print Shipping Labels</td>
<td>/app/accounting/print/printframe.nl?trantype=&amp;printtype=shippinglabel&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PRINT_STATEMENT</td>
<td>Generate Statements</td>
<td>/app/accounting/print/printframe.nl?trantype=&amp;printtype=statement&amp;method=print</td>
</tr>
<tr>
<td>TRAN_PROCESSCOMMISSN</td>
<td>Authorize Employee Commissions</td>
<td>/app/accounting/transactions/processcommissn.nl</td>
</tr>
<tr>
<td>TRAN_PROCESSORDER</td>
<td>Process Individual Order</td>
<td>/app/accounting/transactions/processorder.nl</td>
</tr>
<tr>
<td>TRAN_PROCESSPARTNERCOMMISSN</td>
<td>Authorize Partner Commissions</td>
<td>/app/accounting/transactions/processpartnercommissn.nl</td>
</tr>
<tr>
<td>TRAN_PURCHORDPROC</td>
<td>Bill Purchase Orders</td>
<td>/app/accounting/transactions/purchordermanager.nl?type=proc</td>
</tr>
<tr>
<td>TRAN_PURCHORDRECEIVE</td>
<td>Receive Order</td>
<td>/app/accounting/transactions/purchordermanager.nl?type=receive</td>
</tr>
<tr>
<td>TRAN_QUOTA</td>
<td>Establish Quotas</td>
<td>/app.crm/sales/quota.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>TRAN_REALLOCITEMS</td>
<td>Reallocate Items</td>
<td>/app/accounting/transactions/reallocitems.nl</td>
</tr>
<tr>
<td>TRAN_RECONCILE</td>
<td>Reconcile Bank Statement</td>
<td>/app/accounting/transactions/reconcile.nl</td>
</tr>
<tr>
<td>TRAN_RECONCILE_CC</td>
<td>Reconcile Credit Card Statement</td>
<td>/app/accounting/transactions/reconcile.nl?page_type=cc</td>
</tr>
<tr>
<td>TRAN_REIMBURSEMENTS</td>
<td>Reimbursements</td>
<td>/app/accounting/transactions/reimbursements.nl</td>
</tr>
<tr>
<td>TRAN_REMINDERS</td>
<td>Setup Reminders</td>
<td>/app.center/setup/reminders.nl</td>
</tr>
<tr>
<td>TRAN_REVIEWNEGATIVEINVENTORY</td>
<td>Review Negative Inventory</td>
<td>/app/accounting/transactions/inventory/reviewnegativeinventory.nl</td>
</tr>
<tr>
<td>TRAN_REVRECCREATEJE</td>
<td>Revenue Recognition Schedules</td>
<td>/app/accounting/transactions/revreccreateje.nl</td>
</tr>
<tr>
<td>TRAN_RTNAUTHAPPRV</td>
<td>Approve Return Authorizations</td>
<td>/app/accounting/transactions/returnauthmanager.nl?type=apprv</td>
</tr>
<tr>
<td>TRAN_RTNAUTHCREDIT</td>
<td>Refund Returns</td>
<td>/app/accounting/transactions/returnauthmanager.nl?type=credit</td>
</tr>
<tr>
<td>TRAN_RTNAUTHRECEIVE</td>
<td>Receive Returned Order</td>
<td>/app/accounting/transactions/returnauthmanager.nl?type=receive</td>
</tr>
<tr>
<td>TRAN_RTNAUTHREVERSEREVCOMMITMENT</td>
<td>Generate Revenue Commitment Reversals</td>
<td>/app/accounting/transactions/returnauthmanager.nl?type=reverserevcommitment</td>
</tr>
<tr>
<td>TRAN_SALESORDAPPRV</td>
<td>Approve Sales Orders</td>
<td>/app/accounting/transactions/salesordermanager.nl?type=apprv</td>
</tr>
<tr>
<td>TRAN_SALESORDCOMMITREVENUE</td>
<td>Generate Revenue Commitments</td>
<td>/app/accounting/transactions/salesordermanager.nl?type=commitrevenue</td>
</tr>
<tr>
<td>TRAN_SALESORDFULFILL</td>
<td>Fulfill Orders</td>
<td>/app/accounting/transactions/salesordermanager.nl?type=fulfill</td>
</tr>
<tr>
<td>TRAN_SALESORDPROC</td>
<td>Bill Sales Orders</td>
<td>/app/accounting/transactions/salesordermanager.nl?type=proc</td>
</tr>
<tr>
<td>TRAN_SEARCH</td>
<td>Search</td>
<td>/app/common/search/search.nl</td>
</tr>
<tr>
<td>TRAN_SHORTCUTS</td>
<td>Add Shortcuts</td>
<td>/app.center/shortcuts.nl</td>
</tr>
<tr>
<td>TRAN_SNAPSHOTCOMPOSER</td>
<td>Custom Snapshot Report</td>
<td>/app/reporting/snapshotcomposer.nl</td>
</tr>
<tr>
<td>TRAN_SNAPSHOTS</td>
<td>Setup Snapshots</td>
<td>/app.center/setup/snapshots.nl</td>
</tr>
<tr>
<td>TRAN_TAXPERIODS</td>
<td>Generate Tax Reporting Periods</td>
<td>/app/setup/period/generatetaxperiods.nl</td>
</tr>
<tr>
<td>TRAN_TIMEAPPROVAL</td>
<td>Approve Time</td>
<td>/app/accounting/transactions/timeapproval.nl</td>
</tr>
<tr>
<td>TRAN_TIMEBILL</td>
<td>Track Time</td>
<td>/app/accounting/transactions/timebill.nl</td>
</tr>
<tr>
<td>TRAN_TIMEBILL_WEEKLY</td>
<td>Weekly Time Sheet</td>
<td>/app/accounting/transactions/timebill.nl?weekly=T</td>
</tr>
<tr>
<td>TRAN_TIMECALC</td>
<td>Calculate Time</td>
<td>/core/pages/timecalc.nl</td>
</tr>
<tr>
<td>TRAN_TIMER</td>
<td>Timer</td>
<td>/core/pages/timer.nl</td>
</tr>
<tr>
<td>TRAN_TRNFRORDAPPRV</td>
<td>Approve Transfer Orders</td>
<td>/app/accounting/transactions/transferordermanager.nl?type=apprv</td>
</tr>
<tr>
<td>TRAN_USERPREFS</td>
<td>Set Preferences</td>
<td>/app.center/userprefs.nl</td>
</tr>
<tr>
<td>Task ID</td>
<td>Page Label in NetSuite</td>
<td>URL</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>TRAN_VAT100</td>
<td>VAT 100</td>
<td>/app/accounting/reports/intl/vat100.nl</td>
</tr>
<tr>
<td>TRAN_VENDAUTHAPPRV</td>
<td>Approve Vendor Returns</td>
<td>/app/accounting/transactions/vendauthmanager.nl?type=apprv</td>
</tr>
<tr>
<td>TRAN_VENDAUTHCREDIT</td>
<td>Credit Vendor Returns</td>
<td>/app/accounting/transactions/vendauthmanager.nl?type=credit</td>
</tr>
<tr>
<td>TRAN_VENDAUTHRETURN</td>
<td>Ship Vendor Returns</td>
<td>/app/accounting/transactions/vendauthmanager.nl?type=return</td>
</tr>
<tr>
<td>TRAN_VENDBILLPURCHORD</td>
<td>New Purchase Order</td>
<td>/app/accounting/transactions/vendbillpurchord.nl</td>
</tr>
<tr>
<td>TRAN_VENDPYMTS</td>
<td>Pay Bills</td>
<td>/app/accounting/transactions/vendpymts.nl</td>
</tr>
<tr>
<td>TRAN_WORKORDBUILD</td>
<td>Build Work Orders</td>
<td>/app/accounting/transactions/salesordermanager.nl?type=build</td>
</tr>
</tbody>
</table>
## Chapter 80 SuiteScript Errors

In addition to the errors listed in this table, you may also receive runtime errors generated by the JavaScript engine. These errors are standard JavaScript errors that are not NetSuite-specific and should be handled the same as in any other JavaScript scripting environment.

### SuiteScript Errors

<table>
<thead>
<tr>
<th>Error Code Returned</th>
<th>Long Description or Message</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SSS_AUTHOR_MUST_BE_EMPLOYEE</strong></td>
<td>The author internal id or email must match an employee.</td>
</tr>
<tr>
<td><strong>SSS_FILE_SIZE_EXCEEDED</strong></td>
<td>The file you are trying to load exceeds the maximum allowed file size of {1} megabyte.</td>
</tr>
<tr>
<td><strong>SSS_INSTRUCTION_COUNT_EXCEEDED</strong></td>
<td>Script Execution Instruction Count Exceeded.</td>
</tr>
<tr>
<td><strong>SSS_INVALID_ATTACH_RECORD_TYPE</strong></td>
<td>Attaching of record type {1} to {2} is not supported.</td>
</tr>
<tr>
<td><strong>SSS_INVALID_BCC_EMAIL</strong></td>
<td>One or more bcc emails are not valid.</td>
</tr>
<tr>
<td><strong>SSS_INVALID_CC_EMAIL</strong></td>
<td>One or more cc emails are not valid.</td>
</tr>
<tr>
<td><strong>SSS_INVALID_EMAIL_TEMPLATE</strong></td>
<td>That email template is invalid, disabled, or no longer exists. Please select an active email template.</td>
</tr>
<tr>
<td><strong>SSS_INVALID_FORM_ELEMENT_NAME</strong></td>
<td>You have entered an invalid form element name. It must be prefixed with &quot;custpage&quot;, unique, and cannot contain any non-alphanumeric characters in order to be added to the form or sublist.</td>
</tr>
<tr>
<td><strong>SSS_INVALID_HEADER</strong></td>
<td>One or more headers are not valid.</td>
</tr>
<tr>
<td><strong>SSS_INVALID_HOST_CERT</strong></td>
<td>An untrusted, unsupported, or invalid certificate was found for this host.</td>
</tr>
<tr>
<td><strong>SSS_INVALID_LIST_COLUMN_NAME</strong></td>
<td>You have entered an invalid list column name. It must be unique and cannot contain any non-alphanumeric characters.</td>
</tr>
<tr>
<td><strong>SSS_INVALID_LOG_TYPE</strong></td>
<td>Execution log type must be one of AUDIT, DEBUG, ERROR, or EMERGENCY.</td>
</tr>
<tr>
<td><strong>SSS_INVALID_SCRIPTLET_ID</strong></td>
<td>That Suitelet is invalid, disabled, or no longer exists.</td>
</tr>
<tr>
<td><strong>SSS_INVALID_SRCH_COL</strong></td>
<td>An nlobjSearchColumn contains an invalid column, or is not in proper syntax: {1}.</td>
</tr>
<tr>
<td><strong>SSS_INVALID_SRCH_COLUMN_JOIN</strong></td>
<td>An nlobjSearchColumn contains an invalid column join ID, or is not in proper syntax: {1}.</td>
</tr>
<tr>
<td><strong>SSS_INVALID_SRCH_COLUMN_SUM</strong></td>
<td>An nlobjSearchColumn contains an invalid column summary type, or is not in proper syntax: {1}.</td>
</tr>
<tr>
<td><strong>SSS_INVALID_SRCH_FILTER</strong></td>
<td>An nlobjSearchFilter contains invalid search criteria: {1}.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SSS_INVALID_SRCH_FILTER_EXPR</td>
<td>Malformed search filter expression.</td>
</tr>
<tr>
<td></td>
<td>This is a general error raised when a filter expression cannot be parsed. For example:</td>
</tr>
<tr>
<td></td>
<td>[ f1, 'and', 'and', f2 ]</td>
</tr>
<tr>
<td>SSS_INVALID_SRCH_FILTER_EXPR_DANGLING_OP</td>
<td>Malformed search filter expression: Dangling operator.</td>
</tr>
<tr>
<td></td>
<td>This an error raised when a filter expression has an operator at the end. For example:</td>
</tr>
<tr>
<td></td>
<td>[ f1, 'and', f2, 'or' ]</td>
</tr>
<tr>
<td>SSS_INVALID_SRCH_FILTER_EXPR_OBJ_TYPE</td>
<td>Malformed search filter expression: Unrecognized object type.</td>
</tr>
<tr>
<td></td>
<td>This is an error raised when a filter expression contains something that is not a string or</td>
</tr>
<tr>
<td></td>
<td>parenthesized term. For example:</td>
</tr>
<tr>
<td></td>
<td>[ f1, 17, f2 ]</td>
</tr>
<tr>
<td>SSS_INVALID_SRCH_FILTER_EXPR_PAREN_DEPTH</td>
<td>Malformed search filter expression: Maximum parentheses depth exceeded.</td>
</tr>
<tr>
<td></td>
<td>This is an error raised when adjacent parentheses exceeded a depth of more than three,</td>
</tr>
<tr>
<td></td>
<td>excluding the outermost left and right parentheses. For example:</td>
</tr>
<tr>
<td>SSS_INVALID_SRCH_FILTER_JOIN</td>
<td>An nlobjSearchFilter contains an invalid join ID, or is not in proper syntax: {1}.</td>
</tr>
<tr>
<td>SSS_INVALID_SRCH_FILTER_LIST_PARENS</td>
<td>Malformed search filter list: Unbalanced parentheses.</td>
</tr>
<tr>
<td></td>
<td>This is an error raised when the number of parentheses in an nlobjSearchFilter list does</td>
</tr>
<tr>
<td></td>
<td>not add up right.</td>
</tr>
<tr>
<td>SSS_INVALID_SRCH_FILTER_LIST_TERM</td>
<td>Malformed search filter list: Unexpected object seen where term is expected.</td>
</tr>
<tr>
<td></td>
<td>This is an error raised when the nlobjSearchFilter list produces nonsense during processing.</td>
</tr>
<tr>
<td></td>
<td>The most likely scenario is having extra right parentheses.</td>
</tr>
<tr>
<td>SSS_INVALID_SRCH_OPERATOR</td>
<td>An nlobjSearchFilter contains an invalid operator, or is not in proper syntax: {1}.</td>
</tr>
<tr>
<td>SSS_INVALID_SUBLIST_OPERATION</td>
<td>You have attempted an invalid sublist or line item operation. You are either trying to</td>
</tr>
<tr>
<td></td>
<td>cannot access a field on a non-existent line or you are trying to add or remove lines from</td>
</tr>
<tr>
<td></td>
<td>a static sublist.</td>
</tr>
<tr>
<td>SSS_INVALID_TYPE_ARG</td>
<td>You have entered an invalid type argument: {1}</td>
</tr>
<tr>
<td>SSS_INVALID_UI_OBJECT_TYPE</td>
<td>That operation is not supported for this type of UI object: {1}. It is only supported for:</td>
</tr>
<tr>
<td></td>
<td>{2}.</td>
</tr>
<tr>
<td>SSS_INVALID_URL</td>
<td>The URL must be a fully qualified HTTP or HTTPS URL if it is referencing a non-NetSuite</td>
</tr>
<tr>
<td></td>
<td>resource.</td>
</tr>
<tr>
<td>SSS_INVALID_URL_CATEGORY</td>
<td>The URL category must be one of RECORD, TASKLINK or SUITELET.</td>
</tr>
</tbody>
</table>
### System Errors

<table>
<thead>
<tr>
<th>Error Code Returned</th>
<th>Long Description or Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT_SEARCH_EXCEEDED_MAX_TIME</td>
<td>This search has timed out. You can choose to schedule it to run in the background and have the results emailed to you when complete. On the saved search form, click the Email tab, check Send According to Schedule, choose an email address on the Specific Recipients subtab and a recurrence pattern on the Schedule subtab.</td>
</tr>
<tr>
<td>ABORT_UPLOAD_VIRUS_DETECTED</td>
<td>The file {1} contains a virus {2}. Upload abort.</td>
</tr>
<tr>
<td>ACCTNG_PRD_REQD</td>
<td>Missing next accounting period</td>
</tr>
<tr>
<td>ACCT_DISABLED</td>
<td>account disabled</td>
</tr>
<tr>
<td>ACCT_DISABLED</td>
<td>This account has been disabled.</td>
</tr>
<tr>
<td>ACCT_DISABLED</td>
<td>Please contact &lt;a href=&quot;mailto:{1}&quot;&gt; Accounts Receivable&lt;/a&gt; at 650.627.1316 to re-enable this company.</td>
</tr>
<tr>
<td>ACCT_DISABLED</td>
<td>Your account has been inactivated by an administrator.</td>
</tr>
<tr>
<td>ACCT_NAME_REQD</td>
<td>Accounts require a name.</td>
</tr>
<tr>
<td>ACCT_NEEDS_CAMPAIGN_PROVISION</td>
<td>Please contact your account representative to provision campaign emailing for your account.</td>
</tr>
<tr>
<td>ACCT_NUMS_REQD_OR_DONT_MATCH</td>
<td>Missing ACCT # or ACCT numbers don’t match</td>
</tr>
<tr>
<td>ACCT_NUM_REQD</td>
<td>Missing Account Number. Account number is a required field and it cannot be null or empty.</td>
</tr>
<tr>
<td>ACCT_PRDS_BEING_ADDED</td>
<td>Periods are currently being added to this account. Please try again later.</td>
</tr>
<tr>
<td>ACCT_REQD</td>
<td>Attempting to adjust provisioning for a customer without an existing account</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ACCT_TEMP_DISABLED</td>
<td>You have entered an invalid password on (1) consecutive attempts. Access to your account has been suspended for (2) minutes. If you have forgotten your password, please contact Customer Support.</td>
</tr>
<tr>
<td>ACCT_TEMP_UNAVAILABLE</td>
<td>Can't update information - this company's database is currently offline for maintenance. Please try again later.</td>
</tr>
<tr>
<td>ACCT_TEMP_UNAVAILABLE</td>
<td>(Temporarily unavailable)</td>
</tr>
<tr>
<td>ACCT_TEMP_UNAVAILABLE</td>
<td>The account you are trying to access is currently unavailable while we undergo our regularly scheduled maintenance.</td>
</tr>
<tr>
<td>ACCT_TEMP_UNAVAILABLE</td>
<td>We are currently performing maintenance on our system. Please try again soon.</td>
</tr>
<tr>
<td>ACCT_TEMP_UNAVAILABLE</td>
<td>The account you are trying to access is currently unavailable while we undergo our regularly scheduled maintenance.</td>
</tr>
<tr>
<td>ACCT_TEMP_UNAVAILABLE</td>
<td>Your account is disabled for (1) more minutes due to (2) consecutive failed login attempts.</td>
</tr>
<tr>
<td>ACCT_TEMP_UNAVAILABLE</td>
<td>Your account is not yet ready for you to log in. Please wait and try again.</td>
</tr>
<tr>
<td>ACCT_TEMP_UNAVAILABLE</td>
<td>Your company database is offline.</td>
</tr>
<tr>
<td>ACCT_TEMP_UNAVAILABLE</td>
<td>Your data is still being loaded. Please try again later. Contact &lt;a href='/app/crm/support/nlcorsupport.nl?type=bug&amp;spf=31'&gt;Professional Services&lt;/a&gt; if you have questions.</td>
</tr>
<tr>
<td>ACH_SETUP_REQD</td>
<td>Account (1) is not setup for ACH transactions.</td>
</tr>
<tr>
<td>ACTIVE_ROLE_REQD</td>
<td>You can only set an active login role as the Web Services default role.</td>
</tr>
<tr>
<td>ACTIVE_TRANS_EXIST</td>
<td>There are active direct deposit transactions for this paycheck.</td>
</tr>
<tr>
<td>ADDRESS_LINE_1_REQD</td>
<td>Address Line 1 is a required field and it cannot be null or empty.</td>
</tr>
<tr>
<td>ADMIN_ACCESS_REQ</td>
<td>At least one active administrator for each account must have access.</td>
</tr>
<tr>
<td>ADMIN_ACCESS_REQ</td>
<td>At least one active administrator for this account must have access.</td>
</tr>
<tr>
<td>ADMIN_ACCESS_REQD</td>
<td>Only administrators may enter a memorized transaction in a closed period.</td>
</tr>
<tr>
<td>ADMIN_ONLY_ACCESS</td>
<td>(1) only the administrator may access this page.</td>
</tr>
<tr>
<td>ADMIN_ONLY_ACCESS</td>
<td>(1) only the administrator may currently access this page.</td>
</tr>
<tr>
<td>ADMISSIBILITY_PACKG_TYP_REQD</td>
<td>An Admissibility Package Type is required for this international shipment.</td>
</tr>
<tr>
<td>ALL_DATA_DELETE_REQD</td>
<td>You must first delete all the data in your account before performing this action. Click &lt;a href='/pages/setup/clearaccount.jsp?import=T'&gt;here&lt;/a&gt; to delete your data.</td>
</tr>
<tr>
<td>ALL_MTRX_SUBITMES_OPTNS_REQD</td>
<td>The following matrix subitems exist but aren't included in the options you just specified. On the Matrix tab, please make sure the options you select include all existing subitems:&lt;p&gt;(1)</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ALREADY_IN_INVT</td>
<td>The following {1} numbers are already in inventory: {2}</td>
</tr>
<tr>
<td>ALREADY_IN_INVT</td>
<td>The following {1} number is already in inventory: {2}</td>
</tr>
<tr>
<td>AMORTZN_INVALID_DATE_RANGE</td>
<td>Amortization end date can not be before amortization start date.</td>
</tr>
<tr>
<td>AMORTZN_TMPLT_DATA_MISSING</td>
<td>One or more line items on this transaction have Variable Amortization Templates, but do not have the required {1} also populated. Please either change the Template for these items or indicate which {1} will be used to schedule the amortization.</td>
</tr>
<tr>
<td>AMT_DISALLOWED</td>
<td>Description items may not have an amount.</td>
</tr>
<tr>
<td>AMT_EXCEEDS_APPROVAL_LIMIT</td>
<td>No one in your chain of command has a sufficient spending limit to approve this transaction.</td>
</tr>
<tr>
<td>APPROVAL_PERMS_REQD</td>
<td>{1} The restrictions on your role do not allow you to approve or reject this record.</td>
</tr>
<tr>
<td>AREA_CODE_REQD</td>
<td>Please include an area code with the phone number.</td>
</tr>
<tr>
<td>ASSIGNEE_REQD</td>
<td>{1} must be assigned to {2}</td>
</tr>
<tr>
<td>ATTACHMENT_CONTAINS_VIRUS</td>
<td>The attachment with file name {1} contains a virus {2}. It is removed from the message.</td>
</tr>
<tr>
<td>ATTACHMENT_CONTAINS_VIRUS</td>
<td>The attachment file {0} contains virus {1}. Save message abort.</td>
</tr>
<tr>
<td>ATTACH_SIZE_EXCEEDED</td>
<td>The data you are uploading exceeds the maximum allowable size of {1}. Please change your selection and try again.</td>
</tr>
<tr>
<td>ATTACH_SIZE_EXCEEDED</td>
<td>You have exceeded the maximum attachments size of 5.0 MB. Please remove one or more attachments and try again.</td>
</tr>
<tr>
<td>AT_LEAST_ONE_PACKAGE_REQD</td>
<td>1 or more packages are required.</td>
</tr>
<tr>
<td>AT_LEAST_ONE_SUB_REQD</td>
<td>You must choose at least one subsidiary.</td>
</tr>
<tr>
<td>AUTO_NUM_UPDATE_DISALLOWED</td>
<td>We currently do not support an automatic numbering update of more than {1} {2} records. Please contact &lt;A href='http://app.crm/support/nlcorpsupport.nl?type=support'&gt;NetSuite support&lt;/A&gt; to request a full numbering update of your {2}s.</td>
</tr>
<tr>
<td>BALANCE_EXCEEDS_CREDIT_LIMIT</td>
<td>Customer balance exceeds credit limit</td>
</tr>
<tr>
<td>BANK_ACCT_REQD</td>
<td>You must have a bank account to perform this operation. Click &lt;a href='app/accounting/account/account.nl'&gt;here&lt;/a&gt; to add one.</td>
</tr>
<tr>
<td>BASE_CRNCY_REQD</td>
<td>You may not delete you base currency.</td>
</tr>
<tr>
<td>BILLABLES_DISALLOWED</td>
<td>{1} does not allow billables.</td>
</tr>
<tr>
<td>BILLING_ISSUES</td>
<td>Your account has been locked due to billing issues. You must call your NetSuite Sales Representative for further assistance.</td>
</tr>
<tr>
<td>BILLING_ISSUES</td>
<td>Your account has not been fully paid for. Please log in to your account and follow the billing process or contact your Account Manager.</td>
</tr>
<tr>
<td>BILLING_SCHEDUL_INVALID_RECURR</td>
<td>Billing schedules may not have a recurrence count greater than 500.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>---------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BILLPAY_APPROVAL_UNAVAILBL</td>
<td>Approve Online Bill Payments is currently not available. Please try again in a few minutes.</td>
</tr>
<tr>
<td>BILLPAY_REGSTRTN_REQD</td>
<td>Online bill pay approve payments is not available until your billpay registration is complete.</td>
</tr>
<tr>
<td>BILLPAY_SRVC_UNAVAILBL</td>
<td>Online Bill Pay service is temporarily suspended. If you prefer to wait until Online Bill Pay service is restored, just leave payments to be approved in this list. We will notify you by email as soon as the service is available again. If you need to make an urgent payment, we suggest that you print a check. To do this: 1. Clear the Online Bill Pay check box for that payment. 2. Click the underlined date of the payment, and you will return to the Bill Payment page. 3. On the Bill Payment page, click To Be Printed instead of Bill Pay. 4. Click Submit. 5. Go to Transactions &gt; Print Checks and Forms &gt; Checks, and then mark the check to be printed. 6. Click Submit.</td>
</tr>
<tr>
<td>BILLPAY_SRVC_UNAVAILBL</td>
<td>The Online Bill Pay service is currently not available. Please try again in a few minutes.</td>
</tr>
<tr>
<td>BILL_PAY_STATUS_UNAVAILABLE</td>
<td>View Online Bill Pay Status information is currently not available. Please try again in a few minutes.</td>
</tr>
<tr>
<td>BILL_PAY_STATUS_UNAVAILABLE</td>
<td>View Online Bill Pay Status is not available until your bill pay registration is complete.</td>
</tr>
<tr>
<td>BILL_PMTS_MADE_FROM_ACCT_ONLY</td>
<td>Your payment has been recorded, but online bill payments can only be made from the account &lt;b&gt;{1}&lt;/b&gt;, so no online bill pay payment will be made. You should return to the payment screen if you wish to print the check.</td>
</tr>
<tr>
<td>BIN_DSNT_CONTAIN_ENOUGH_ITEM</td>
<td>The following bins do not contain enough of the requested item ({1}): {2}</td>
</tr>
<tr>
<td>BIN_DSNT_CONTAIN_ENOUGH_ITEM</td>
<td>The following bin does not contain enough of the requested item ({1}): {2}</td>
</tr>
<tr>
<td>BIN_ITEM_UNAVAILBL</td>
<td>The following bins are not available for the specified item: {1}</td>
</tr>
<tr>
<td>BIN_ITEM_UNAVAILBL</td>
<td>The following bins are not available for the specified item ({1}): {2}</td>
</tr>
<tr>
<td>BIN_ITEM_UNAVAILBL</td>
<td>The following bin in not available for the specified item ({1}): {2}</td>
</tr>
<tr>
<td>BIN_ITEM_UNAVAILBL</td>
<td>The following bin is not available for the specified item: {1}</td>
</tr>
<tr>
<td>BIN_SETUP_REQD</td>
<td>The following bins are not associated with the item ‘{1}’: {2}. You can associate bins with an item on the inventory tab of the item record.</td>
</tr>
<tr>
<td>BIN_UNDEFND</td>
<td>The following bins specified for the item (1) are not defined in the transaction location (2): {3}</td>
</tr>
<tr>
<td>CALENDAR_PREFS_REQD</td>
<td>Set up (1) Calendar Preferences first.</td>
</tr>
<tr>
<td>CALENDAR_PREFS_REQD</td>
<td>Set up Calendar Preferences first</td>
</tr>
<tr>
<td>CAMPAIGN_ALREADY_EXECUTED</td>
<td>You cannot delete email campaigns that have already been executed</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CAMPAIGN_IN_USE</td>
<td>You cannot delete a campaign event that already has activity.</td>
</tr>
<tr>
<td>CAMPAIGN_SET_UP_REQD</td>
<td>The following steps need to be performed before a campaign can be created:&lt;p&gt;(1)</td>
</tr>
<tr>
<td>CANT_CALC_FEDEX_RATES</td>
<td>FedEx rates cannot be calculated:</td>
</tr>
<tr>
<td>CANT_CANCEL_APPRVD_RETNR_AUTH</td>
<td>You cannot cancel this return authorization because it has already been approved.</td>
</tr>
<tr>
<td>CANT_CANCEL_BILL_PMT</td>
<td>The Online Bill Payment cannot be stopped because the payment may already have been made.</td>
</tr>
<tr>
<td>CANT_CHANGE_CONTACT_RESTRICTN</td>
<td>You cannot change the restriction on this contact.</td>
</tr>
<tr>
<td>CANT_CHANGE_CRMRECORDTYPELINKS</td>
<td>Cannot alter standard CrmRecordTypeLinks</td>
</tr>
<tr>
<td>CANT_CHANGE_EVENT_PRIMARY_TYP</td>
<td>You cannot change the primary type for this event</td>
</tr>
<tr>
<td>CANT_CHANGE_LEAD_SOURCE_CAT</td>
<td>You cannot change the category for a leadsource that is defined as the default leadsource for another category</td>
</tr>
<tr>
<td>CANT_CHANGE_PSWD</td>
<td>Cannot change password as the company user does not exist.</td>
</tr>
<tr>
<td>CANT_CHANGE_PSWD</td>
<td>You changed your password less than 24 hours ago. NetSuite only allows one password change per 24-hour period.</td>
</tr>
<tr>
<td>CANT_CHANGE_REV_REC_TMPLT</td>
<td>The rev rec template on a billable expense can not be changed or removed once it is saved.</td>
</tr>
<tr>
<td>CANT_CHANGE_REV_REC_TMPLT</td>
<td>The rev rec template on billable time and items can not be changed or removed once it is saved.</td>
</tr>
<tr>
<td>CANT_CHANGE_SUB</td>
<td>You cannot change the subsidiary on this record because doing so will change the subsidiary selected on the associated employee record.</td>
</tr>
<tr>
<td>CANT_CHANGE_TASK_LINK</td>
<td>Cannot alter standard task links</td>
</tr>
<tr>
<td>CANT_CHANGE_UNITS_TYP</td>
<td>You may not change the units type of an item after it has been set.</td>
</tr>
<tr>
<td>CANT_CHANGE_VSOE_ALLOCTN</td>
<td>You are attempting to change the VSOE Allocation for a transaction in a closed period. You must either change the posting period for the related transaction or open the period.</td>
</tr>
<tr>
<td>CANT_COMPLETE_FULFILL</td>
<td>The fulfillment cannot be completed.</td>
</tr>
<tr>
<td>CANT_CONNECT_TO_STORE</td>
<td>Error - Unable to connect to store (1)</td>
</tr>
<tr>
<td>CANT_CREATE_FILES</td>
<td>Could not create files for uploading your data</td>
</tr>
<tr>
<td>CANT_CREATE_NON_UNIQUE_RCRD</td>
<td>A record with the same unique signatures already exists. You must enter unique signatures for each record you create.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because a currency must be defined for the &quot;Ship From&quot; country &quot;(1)&quot; when using the Insured Value option. Go to Lists -&gt; Accounting -&gt; Currencies to create a currency for (1).</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because a currency must be defined for the &quot;Ship To&quot; country &quot;(1)&quot; when using the COD option. Go to Lists -&gt; Accounting -&gt; Currencies to create a currency for (1).</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the Addressee field of the &quot;Ship To&quot; address is not set. Please enter a &quot;Ship To&quot; Addressee on the Item Fulfillment page.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the Address 1 field of the &quot;Ship From&quot; address is not set. Please go to $(regex) to enter the &quot;Ship From&quot; Address 1.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the Address 1 field of the &quot;Ship To&quot; address is not set. Please enter a &quot;Ship To&quot; Address 1 on the Item Fulfillment page.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the Attention field of the &quot;Ship From&quot; address is not set. Please go to $(regex) to enter the &quot;Ship From&quot; Attention.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the City field of the &quot;Ship To&quot; address is not set. Please enter a &quot;Ship To&quot; City on the Item Fulfillment page.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the City of the &quot;Ship From&quot; address is not set. Please go to $(regex) to enter the &quot;Ship From&quot; City.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the Company or Location Name of the &quot;Ship From&quot; address is not set. Please go to $(regex) to enter the &quot;Ship From&quot; Company/Location Name.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the Country field of the &quot;Ship To&quot; address is not set. Please enter a &quot;Ship To&quot; Country on the Item Fulfillment page.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the Country of the &quot;Ship From&quot; address is not set. Please go to $(regex) to set the &quot;Ship From&quot; Country.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the Package Weight was not entered. Please enter a value in the Package Weight field on the Item Fulfillment page.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the Phone Number of the &quot;Ship From&quot; address is not set. Please go to $(regex) to set the &quot;Ship From&quot; Phone Number.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the Phone Number of the &quot;Ship To&quot; address is not set. Please enter a &quot;Ship To&quot; Phone Number on the Item Fulfillment page.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the Pickup Type was not set. Please go to Setup &gt; Set Up Shipping to select a shipping Pickup Type.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the shipping method was not set. Please go to Lists &gt; Shipping Items to select a Shipping Label Integration shipping method for this shipping item.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the State field of the &quot;Ship To&quot; address is not set. Please enter a &quot;Ship To&quot; State on the Item Fulfillment page.</td>
</tr>
<tr>
<td></td>
<td>A shipping label could not be generated because the State of the &quot;Ship From&quot; address is not set. Please go to $(regex) to enter the &quot;Ship From&quot; State.</td>
</tr>
<tr>
<td></td>
<td>A shipping label could not be generated because the Zip Code of the &quot;Ship From&quot; address is not set. Please go to $(regex) to enter the &quot;Ship From&quot; Zip Code.</td>
</tr>
<tr>
<td></td>
<td>A shipping label could not be generated because the Zip field of the &quot;Ship To&quot; address is not set. Please enter a &quot;Ship To&quot; Zip code on the Item Fulfillment page.</td>
</tr>
<tr>
<td></td>
<td>A shipping label could not be generated because your {1} Account Number is not set. Go to Setup &gt; Set Up Shipping &gt; {2} Registration to enter your {3} Account Number.</td>
</tr>
<tr>
<td></td>
<td>A shipping label could not be generated because your {1} Registration Address Line 1 is not set. Go to Setup &gt; {2} Registration to complete the Address Line 1 field.</td>
</tr>
<tr>
<td></td>
<td>A shipping label could not be generated because your {1} Registration City is not set. Go to Setup &gt; {2} Registration to enter your City.</td>
</tr>
<tr>
<td></td>
<td>A shipping label could not be generated because your {1} Registration Company field is not set. Go to Setup &gt; {2} Registration to enter a name in the Company field.</td>
</tr>
<tr>
<td></td>
<td>A shipping label could not be generated because your {1} Registration Country is not set. Go to Setup &gt; {2} Registration to select your Country.</td>
</tr>
<tr>
<td></td>
<td>A shipping label could not be generated because your {1} Registration Ship to Attention field is not set. Go to Setup &gt; {2} Registration to enter a name in the Ship to Attention field.</td>
</tr>
<tr>
<td></td>
<td>A shipping label could not be generated because your {1} Registration State is not set. Go to Setup &gt; {2} Registration to select or enter your State.</td>
</tr>
<tr>
<td></td>
<td>A shipping label could not be generated because your {1} Registration Zip Code is not set. Go to Setup &gt; {2} Registration to enter your Zip Code.</td>
</tr>
<tr>
<td>CANT_CREATE_USER</td>
<td>Could not create the user. Please confirm that you have entered a legal password.</td>
</tr>
<tr>
<td>CANT_CREATE_SHIP_LABEL</td>
<td>A shipping label could not be generated because the In Bond Code field is not set. Please enter a value in the In Bond Code field on the Item Fulfillment page.</td>
</tr>
<tr>
<td>CANT_DELETE_ACCT</td>
<td>This account cannot be deleted because it has associated transactions.</td>
</tr>
<tr>
<td>CANT_DELETE_ACCT</td>
<td>This account cannot be deleted because it is a special type of account needed by {1}</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CANT_DELETE_ACCT</td>
<td>This account cannot be deleted because it is a special type of account needed by NetSuite</td>
</tr>
<tr>
<td>CANT_DELETE_ACCT</td>
<td>This account cannot be deleted because it is a special type of account needed by the system</td>
</tr>
<tr>
<td>CANT_DELETE_ACCT</td>
<td>This account cannot be deleted because it is used by one or more transactions or it has child accounts or it is used by one or more items.</td>
</tr>
<tr>
<td>CANT_DELETE_ACCT_PRD</td>
<td>You may not delete an accounting period with transactions posted to it. You must first edit the transactions, change the posting period and then delete the period.</td>
</tr>
<tr>
<td>CANT_DELETE_ALLOCTN</td>
<td>This allocation detail can not be deleted because it has a journal entry.</td>
</tr>
<tr>
<td>CANT_DELETE_BIN</td>
<td>You may not delete this bin record because it is already in use. You must either remove all references to it in item records and transactions or make it inactive.</td>
</tr>
<tr>
<td>CANT_DELETE_CATEGORY</td>
<td>This category cannot be deleted because it has child items</td>
</tr>
<tr>
<td>CANT_DELETE_CATEGORY</td>
<td>This category cannot be deleted because it has subcategories</td>
</tr>
<tr>
<td>CANT_DELETE_CC_PROCESSOR</td>
<td>This credit card processor is used in transaction and cannot be deleted.</td>
</tr>
<tr>
<td>CANT_DELETE_CELL</td>
<td>This cell cannot be deleted because it has child items</td>
</tr>
<tr>
<td>CANT_DELETE_CHILD_RCRDS_EXIST</td>
<td>This record can not be deleted because it has child records.</td>
</tr>
<tr>
<td>CANT_DELETE_CHILD_RCRD_FOUND</td>
<td>This {1} record cannot be deleted because it is referenced by other records.</td>
</tr>
<tr>
<td>CANT_DELETE_CLASS</td>
<td>This class cannot be deleted because it has child items</td>
</tr>
<tr>
<td>CANT_DELETE_COLOR_THEME</td>
<td>This color theme cannot be deleted because it is being used</td>
</tr>
<tr>
<td>CANT_DELETE_COMMISSIONS_SCHDUL</td>
<td>This schedule has already been used to generate commission calculations and can't be deleted. If no authorizations have been made, schedule can be deleted after being removed from all active plans.</td>
</tr>
<tr>
<td>CANT_DELETE_COMPANY</td>
<td>This company cannot be deleted because it has child entities</td>
</tr>
<tr>
<td>CANT_DELETE_COMPANY_TYP</td>
<td>This company type cannot be deleted because the company has associated transactions.</td>
</tr>
<tr>
<td>CANT_DELETE_CONTACT_HAS_CHILD</td>
<td>The contact record cannot be deleted because it has child records.</td>
</tr>
<tr>
<td>CANT_DELETE_CONTACT_HAS_CHILD</td>
<td>This contact cannot be deleted because it has child entities</td>
</tr>
<tr>
<td>CANT_DELETE_CSTM_FIELD</td>
<td>This custom field cannot be deleted because it is referred to by other custom fields</td>
</tr>
<tr>
<td>CANT_DELETE_CSTM_FORM</td>
<td>This custom form cannot be deleted because it is referred to by other custom forms</td>
</tr>
<tr>
<td>CANT_DELETE_CSTM_ITEM_FIELD</td>
<td>This custom item field has dependent matrix items. It can not be deleted.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CANT_DELETE_CSTM_LAYOUT</td>
<td>This custom layout cannot be deleted because it is used by custom forms</td>
</tr>
<tr>
<td>CANT_DELETE_CSTM_LIST</td>
<td>This custom list cannot be deleted because it is referred to by custom fields</td>
</tr>
<tr>
<td>CANT_DELETE_CSTM_RCRD</td>
<td>This custom record cannot be deleted because it is referred to by custom fields</td>
</tr>
<tr>
<td>CANT_DELETE_CSTM_RCRD_ENTRY</td>
<td>This custom record entry cannot be deleted because it is referred to by other records</td>
</tr>
<tr>
<td>CANT_DELETE_CUST</td>
<td>You can't delete this customer because it's set up as default Anonymous Customer</td>
</tr>
<tr>
<td>CANT_DELETE_CUSTOMER</td>
<td>This customer or job cannot be deleted because it has child entities.</td>
</tr>
<tr>
<td>CANT_DELETE_DEFAULT_FLDR</td>
<td>You cannot delete the default folders.</td>
</tr>
<tr>
<td>CANT_DELETE_DEFAULT_PRIORITY</td>
<td>You cannot delete the default case priority. Please select a new default first.</td>
</tr>
<tr>
<td>CANT_DELETE_DEFAULT_SALES_REP</td>
<td>Default Sales Rep Role cannot be deleted.</td>
</tr>
<tr>
<td>CANT_DELETE_DEFAULT_STATUS</td>
<td>You cannot delete a default case status. Please select a new default first.</td>
</tr>
<tr>
<td>CANT_DELETE_DEFAULT_STATUS</td>
<td>You can't delete or inactivate that status because it is a set up as a default status. Please navigate to &lt;a href='/app/setup/sfasetup.nl' target='_blank'&gt;Sales Preferences&lt;/a&gt; and change that status</td>
</tr>
<tr>
<td>CANT_DELETE_DEFAULT_VALUE</td>
<td>You may not delete or inactivate that value because it is a default. Please select a new default first.</td>
</tr>
<tr>
<td>CANT_DELETE_EMPL</td>
<td>This employee cannot be deleted because it has child entities</td>
</tr>
<tr>
<td>CANT_DELETE_ENTITY</td>
<td>This entity cannot be deleted because it has child items</td>
</tr>
<tr>
<td>CANT_DELETE_FIN_STMTMNT_LAYOUT</td>
<td>This financial statement layout cannot be deleted because it is referred to by other layouts.</td>
</tr>
<tr>
<td>CANT_DELETE_FLDR</td>
<td>These predefined folders cannot be deleted</td>
</tr>
<tr>
<td>CANT_DELETE_HAS_CHILD_ITEM</td>
<td>This (1) cannot be deleted because it has child items</td>
</tr>
<tr>
<td>CANT_DELETE_INFO_ITEM</td>
<td>This information item cannot be deleted because it has child items</td>
</tr>
<tr>
<td>CANT_DELETE_ITEM</td>
<td>This item cannot be deleted because it has child items</td>
</tr>
<tr>
<td>CANT_DELETE_ITEM_LAYOUT</td>
<td>This item/category layout cannot be deleted because it is used by store tabs</td>
</tr>
<tr>
<td>CANT_DELETE_ITEM_TMPLT</td>
<td>This item/category template cannot be deleted because it is referred to by a theme or an item</td>
</tr>
<tr>
<td>CANT_DELETE_JOB_RESOURCE_ROLE</td>
<td>Default Job Resource Role cannot be deleted.</td>
</tr>
<tr>
<td>CANT_DELETE_LEGACY_CATEGORY</td>
<td>Legacy category cannot be removed</td>
</tr>
<tr>
<td>CANT_DELETE_LINE</td>
<td>This line cannot be deleted, because it is referred to by other records. Before removing this line, remove any discount or markup lines applied to it.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CANT_DELETE_MEDIA_ITEM</td>
<td>This media item cannot be deleted because it is being referenced by another item.</td>
</tr>
<tr>
<td>CANT_DELETE_MEMRZD_TRANS</td>
<td>This memorized transaction cannot be deleted because it is referenced in transactions.</td>
</tr>
<tr>
<td>CANT_DELETE_OR_CHANGE_ACCT</td>
<td>Special accounts cannot be deleted and their type cannot be changed.</td>
</tr>
<tr>
<td>CANT_DELETE_PLAN_ASSIGNMENT</td>
<td>Trying to delete plan assignment referenced by precalcs.</td>
</tr>
<tr>
<td>CANT_DELETE_PRESNTN_CAT</td>
<td>This presentation category cannot be deleted because it has subcategories.</td>
</tr>
<tr>
<td>CANT_DELETE_RCRD</td>
<td>This (1) record cannot be deleted because it referenced by other records.</td>
</tr>
<tr>
<td>CANT_DELETE_RCRD</td>
<td>This record cannot be deleted because it has (1) child records (2).</td>
</tr>
<tr>
<td>CANT_DELETE_RCRD</td>
<td>This record cannot be deleted because it is referenced by other records or it is used by one or more transactions.</td>
</tr>
<tr>
<td>CANT_DELETE_RCRDS</td>
<td>This record cannot be deleted, because it is referred to by other records.</td>
</tr>
<tr>
<td>CANT_DELETE_RCRDS</td>
<td>Selected records could not be deleted because one or more of them are of a special type of account needed by (1)</td>
</tr>
<tr>
<td>CANT_DELETE_SITE_THEME</td>
<td>This site theme cannot be deleted because it is being used.</td>
</tr>
<tr>
<td>CANT_DELETE_SOLUTN</td>
<td>This solution cannot be deleted because it has been applied to support cases.</td>
</tr>
<tr>
<td>CANT_DELETE_STATUS_TYPE</td>
<td>You cannot delete the only status of type (1).</td>
</tr>
<tr>
<td>CANT_DELETE_SUBTAB</td>
<td>This subtab cannot be deleted because it is referred to by custom fields.</td>
</tr>
<tr>
<td>CANT_DELETE.SYSTEM NOTE</td>
<td>You cannot alter or delete a system logged note.</td>
</tr>
<tr>
<td>CANT_DELETE.TAX_VENDOR</td>
<td>This is a special tax vendor and cannot be deleted.</td>
</tr>
<tr>
<td>CANT_DELETE_TMPLT_RCRD</td>
<td>This template record cannot be deleted.</td>
</tr>
<tr>
<td>CANT_DELETE_TRANS</td>
<td>This transaction cannot be deleted because it is linked to one or more commission transactions. The commission authorizations due to this transaction need to be removed to be able to delete this transaction.</td>
</tr>
<tr>
<td>CANT_DELETE_TRANS</td>
<td>This transaction cannot be deleted because it is referenced by other transactions. It may be a bill or an invoice that has been paid or an expense that has been reimbursed.</td>
</tr>
<tr>
<td>CANT_DELETE_TRAN_LINE</td>
<td>Failed to delete line (1). This line is linked to another transaction.</td>
</tr>
<tr>
<td>CANT_DELETE_TRAN_LINES</td>
<td>Lines with partially recognized rev rec or amortization schedules cannot be deleted.</td>
</tr>
<tr>
<td>CANT_DELETE_UPDATE_ACCT</td>
<td>This account cannot be deleted or changed because it is a special type of account needed by (1)</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CANT_DELETE_VENDOR</td>
<td>This vendor cannot be deleted because there are dependent items, such as a pending payment. If you wish to remove the payee, you must first delete all such dependent items.</td>
</tr>
<tr>
<td>CANT_DEL_DEFAULT_SHIP_METHOD</td>
<td>This Shipping Item cannot be deleted because it is the Default Shipping Method. Please go to Setup &gt; Accounting &gt; Set Up Shipping and choose a new Default Shipping Method before deleting this Shipping Item.</td>
</tr>
<tr>
<td>CANT_DOWNLOAD_EXPIRED_FILE</td>
<td>This file has expired and can no longer be downloaded</td>
</tr>
<tr>
<td>CANT_EDIT_DPLYMNT_IN_PROGRESS</td>
<td>You cannot change or delete a deployment that is in progress or in the queue.</td>
</tr>
<tr>
<td>CANT_EDIT_DPLYMNT_IN_PROGRESS</td>
<td>You cannot edit a script deployment when it is being executed.</td>
</tr>
<tr>
<td>CANT_EDIT_OLD_CASE</td>
<td>This case cannot be edited because it was closed (1) or more days ago.</td>
</tr>
<tr>
<td>CANT_EDIT_TAGATA</td>
<td>The Receivable Tegata is linked to Invoices and is no longer editable</td>
</tr>
<tr>
<td>CANT_ESTABLISH_LINK</td>
<td>Unable to establish link with (1)</td>
</tr>
<tr>
<td>CANT_FIND_BUG</td>
<td>Cannot locate the bug that was just entered (1)!</td>
</tr>
<tr>
<td>CANT_FIND_MAIL_MERGE_ID</td>
<td>Mail Merge Id not found</td>
</tr>
<tr>
<td>CANT_FIND_RCRD</td>
<td>Could not find record with (1) = (2)</td>
</tr>
<tr>
<td>CANT_FIND_SOURCE_AMORTZN_ACCT</td>
<td>The source account for the amortization schedule could not be determined.</td>
</tr>
<tr>
<td>CANT_FIND_UPS_REG_FOR_LOC</td>
<td>No UPS registration was found for the location selected. Please select a different shipping item, or go to Setup &gt; Set Up Shipping to register a UPS account for this location.</td>
</tr>
<tr>
<td>CANT_INACTIVATE_COMMSSN_PLAN</td>
<td>You cannot inactivate a plan that has commission payments that are pending authorization. Please clear the commission payments at Transactions &gt; Authorize Commissions before inactivating this plan.</td>
</tr>
<tr>
<td>CANT_LOAD_SAVED_SEARCH_PARAM</td>
<td>Error loading saved search params</td>
</tr>
<tr>
<td>CANT_MAKE_CONTACT_PRIVATE</td>
<td>Employee contacts cannot be made private</td>
</tr>
<tr>
<td>CANT_MAKE_CONTACT_PRIVATE</td>
<td>Individual relationship contacts cannot be made private</td>
</tr>
<tr>
<td>CANT_MODIFY_APPRVD_TIME</td>
<td>Time records can not be modified once they have been approved.</td>
</tr>
<tr>
<td>CANT_MODIFY_SUB</td>
<td>You cannot change the subsidiary of this entity because one or more transactions exist for this entity.</td>
</tr>
<tr>
<td>CANT_MODIFY_TAGATA</td>
<td>The Payable Tegata is no longer in Issued state and cannot be modified.</td>
</tr>
<tr>
<td>CANT_MODIFY_TAGATA</td>
<td>The Receivable Tegata is no longer in Holding state and cannot be modified.</td>
</tr>
<tr>
<td>CANT_PAY_TAGATA</td>
<td>The Payable Tegata is linked to bills and cannot be modified.</td>
</tr>
<tr>
<td>CANT_PAY_TAGATA</td>
<td>Endorsed Tegata can only be paid on or after its maturity date.</td>
</tr>
<tr>
<td>CANT_PAY_TAGATA</td>
<td>Payable Tegata can only be paid on or after its maturity date.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CANT_RECEIVE_TAGATA</td>
<td>Receivable Tegata can only be collected on or after its maturity date.</td>
</tr>
<tr>
<td>CANT_REJECT_ORDER</td>
<td>You cannot reject this order because it has already been approved.</td>
</tr>
<tr>
<td>CANT_REMOVE_ACH_PAY_METHOD</td>
<td>ACH payment methods cannot be removed</td>
</tr>
<tr>
<td>CANT_REMOVE_NEXUS</td>
<td>A nexus cannot be removed from a subsidiary if the nexus is associated with a transaction.</td>
</tr>
<tr>
<td>CANT_REMOVE_SCHEDUL</td>
<td>You have attempted to remove an active schedule from a plan. Removing this participant is not permitted once commissions against the plan have been generated.</td>
</tr>
<tr>
<td>CANT_REMOVE_SUB</td>
<td>You cannot remove subsidiary: {1} because this record is used on a transaction for subsidiary: {1}.</td>
</tr>
<tr>
<td>CANT_REMOVE_SUB</td>
<td>You attempted to remove one or more subsidiaries from this item, but the item appears in at least one transaction in those subsidiaries. In order to remove a subsidiary from the item, make sure the item does not appear in any transactions for that subsidiary.</td>
</tr>
<tr>
<td>CANT_REMOV_ALL_FULFILMNT_LINKS</td>
<td>You may not modify this sales order in such a way that it removes all links to any fulfillment. The modifications you made would leave the fulfillment unlinked.</td>
</tr>
<tr>
<td>CANT_REMOV_ITEM_SUB</td>
<td>You may not remove a subsidiary from an item that is a member of an assembly, group, or kit item if the parent item is available in that subsidiary.</td>
</tr>
<tr>
<td>CANT_RESUBMIT_FAILED_DPLYMNT</td>
<td>You cannot submit a deployment for execution whose status is set to Failed or Scheduled.</td>
</tr>
<tr>
<td>CANT_RETURN_USED_GIFT_CERT</td>
<td>Used gift certificates can not be returned.</td>
</tr>
<tr>
<td>CANT_REV_REC_BODY_AND_LINE</td>
<td>The Revenue Recognition fields must be specified at EITHER the transaction body or the item line level, and may NOT be specified at both levels.</td>
</tr>
<tr>
<td>CANT_SCHEDUL_RECUR_EVENT</td>
<td>Because the number of days in each month differs, recurring monthly events cannot be scheduled after the 28th.</td>
</tr>
<tr>
<td>CANT_SEND_EMAIL</td>
<td>Unable to send notification email</td>
</tr>
<tr>
<td>CANT_SEND_EMAIL</td>
<td>Unable to send notification email to support rep</td>
</tr>
<tr>
<td>CANT_SET_CLOSE_DATE</td>
<td>Unable to set expected close date of prospect/lead based on current estimates/opportunities.</td>
</tr>
<tr>
<td>CANT_SET_STATUS</td>
<td>Unable to set status of prospect/lead based on current estimates.</td>
</tr>
<tr>
<td>CANT_SWITCH_ROLES_FROM_LOGIN</td>
<td>Role switching is not allowed from this login.</td>
</tr>
<tr>
<td>CANT_UPDATE_AMT</td>
<td>The amount on lines containing partially/fully recognized schedules can not be changed.</td>
</tr>
<tr>
<td>CANT_UPDATE_DYNAMIC_GROUP</td>
<td>You cannot update dynamic groups. Instead you must modify the saved search associated with the group.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CANT_UPDATE_FLD nutritional</td>
<td>These predefined folders cannot be updated</td>
</tr>
<tr>
<td>CANT_UPDATE_PRODUCT_FEED</td>
<td>This item has multiple product feeds. Web Services schema version 2_6 or greater is required to modify product feeds for this item</td>
</tr>
<tr>
<td>CANT_UPDATE_RECRD HAS CHANGED</td>
<td>Cannot update bug. Record has changed since you last retrieved it.</td>
</tr>
<tr>
<td>CANT_UPDATE_RECUR_EVENT</td>
<td>Event &lt;id (1)&gt; contains recurrence patterns that are not supported in your client application. You are not allowed to update recurrence pattern on this event. Contact your software vendor for the latest Web Services upgrade.</td>
</tr>
<tr>
<td>CANT_UPDATE_STATUS_TYPE</td>
<td>You cannot update the only status of type (1)</td>
</tr>
<tr>
<td>CANT_VOID_TRANS</td>
<td>You cannot void this transaction because it is linked to by one or more transactions such as payments. You must delete or void those transactions first</td>
</tr>
<tr>
<td>CASE_ALREADY_ASSIGNED</td>
<td>This case cannot be grabbed because it is already assigned to another rep. To view the case, go back and click on the case number.</td>
</tr>
<tr>
<td>CASE_DSNT_EXIST</td>
<td>Case doesn’t exist or no customer is associated with case.</td>
</tr>
<tr>
<td>CASE_NOT_GROUP_MEMBER</td>
<td>(1) this case record does not belong to your group.</td>
</tr>
<tr>
<td>CASH_SALE_EDIT_DISALLWD</td>
<td>This cash sale cannot be edited while it has an Automated Clearing House transmission in process.</td>
</tr>
<tr>
<td>CC_ACCT_REQD</td>
<td>You must have a credit card account to perform this operation.</td>
</tr>
<tr>
<td>CC_ACCT_REQD</td>
<td>You must have a credit card account to perform this operation. Click &lt;a href=&quot;/app/accounting/account/account.nl&quot;&gt;here&lt;/a&gt; to add one.</td>
</tr>
<tr>
<td>CC_ALREADY_SAVED</td>
<td>That credit card is already saved. Please use the saved credit card.</td>
</tr>
<tr>
<td>CC_EMAIL_ADDRESS_REQD</td>
<td>Please go back and provide an email address to CC store orders to.</td>
</tr>
<tr>
<td>CC_NUM_REQD</td>
<td>Please provide a credit card number.</td>
</tr>
<tr>
<td>CC_PROCESSOR_ERROR</td>
<td>An error occurred while processing the credit card. Please contact the merchant for assistance.</td>
</tr>
<tr>
<td>CC_PROCESSOR_NOT_FOUND</td>
<td>A suitable credit card processor was not found for this transaction.</td>
</tr>
<tr>
<td>CERT_UNAVAILABLE</td>
<td>Certificate unavailable (most likely has not been presented by client)</td>
</tr>
<tr>
<td>CHANGE_PMT_DATE_AND_REAPPROVE</td>
<td>The payment is more than 30 days past due and has NOT been sent. Edit the payment to change the date and reapprove.</td>
</tr>
<tr>
<td>CHAR_ERROR</td>
<td>Character error on Line# (1) Column# (2) (Byte # (3)). (4)</td>
</tr>
<tr>
<td>CITY_REQD</td>
<td>City is a required field and it cannot be null or empty.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CLASS_ALREADY_EXISTS</td>
<td>A class already exists with that name. Go &lt;a href=&quot;javascript:history.go(-1);&quot;&gt;back&lt;/a&gt;, change the name and resubmit.</td>
</tr>
<tr>
<td>CLASS_NOT_FOUND</td>
<td>Class {1} cannot be found.</td>
</tr>
<tr>
<td>CLASS_NOT_FOUND</td>
<td>could not find class {1}</td>
</tr>
<tr>
<td>CLASS_OR_DEPT_OR_CUST_REQD</td>
<td>only one of class, cust, and dept can be non-null</td>
</tr>
<tr>
<td>CLOSED_TRAN_PRD</td>
<td>You cannot move a transaction to or from a closed period.</td>
</tr>
<tr>
<td>COGS_ERROR</td>
<td>LIFO/FIFO COGS count does not equal the number of items requested COGS ERROR 9765 itemsLinked={1}, itemsTotal={2}, kdoc={3}, nid={4}</td>
</tr>
<tr>
<td>COMMSSN_ALREADY_CALCULTD</td>
<td>You have attempted to remove an active sales participant from a plan. Removing this participant is not permitted once commissions against the plan have been generated.</td>
</tr>
<tr>
<td>COMMSSN_FEATURE_DISABLED</td>
<td>You have not enabled the Commissions feature.</td>
</tr>
<tr>
<td>COMMSSN_FEATURE_DISABLED</td>
<td>You have not enabled the Partner Commissions/Royalties feature.</td>
</tr>
<tr>
<td>COMMSSN_PAYROLL_ITEM_REQD</td>
<td>A commission payroll item must be added for each employee to be processed through payroll.</td>
</tr>
<tr>
<td>COMPANION_PROP_REQD</td>
<td>Error - Items do not have companion property (column) {1}</td>
</tr>
<tr>
<td>COMPANY_DISABLED</td>
<td>Please contact Accounts Receivable at {1} or 650.627.1316 to re-enable this company.</td>
</tr>
<tr>
<td>COMP_DELETED_OR_MERGED</td>
<td>The company you try to attach the context to has been deleted or merged.</td>
</tr>
<tr>
<td>CONCUR_BILLPAY_JOB_DISALLWD</td>
<td>Your account currently has a bill pay approval job in progress. Only one bill pay approval job per account is allowed at a time. Please wait until this process completes before submitting another group of payments for approval. &lt;BR&gt;&lt;BR&gt;Visit the &lt;a href=&quot;/app/external/xml/upload/uploadlog.nl?displayType=BILLPAY&quot;&gt;status page &lt;/a&gt; to track the progress of the current job.</td>
</tr>
<tr>
<td>CONCUR_BULK_JOB_DISALLWD</td>
<td>This Account is already running a bulk processing job. Please visit the &lt;a href=&quot;/app/external/xml/upload/uploadlog.nl?displayType=BULKFULFILL&quot;&gt;status page &lt;/a&gt; to track the progress of the current job.</td>
</tr>
<tr>
<td>CONCUR_MASS_UPDATE_DISALLWD</td>
<td>A mass update is currently running in this account. Please try again in a few minutes.</td>
</tr>
<tr>
<td>CONCUR_SEARCH_DISALLWD</td>
<td>Search aborted by concurrent {1} search. Only one search may run at a time.</td>
</tr>
<tr>
<td>CONSRLD_PRNT_AND_CHILD_DISALLWD</td>
<td>A company can be a consolidated child or a consolidated parent but not both</td>
</tr>
<tr>
<td>CONTACT_ALREADY_EXISTS</td>
<td>A contact record with this name already exists. Every contact record must have a unique name.</td>
</tr>
<tr>
<td>CONTACT_ALREADY_EXISTS</td>
<td>A contact with the name [{1}] already exists.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CONTACT_NOTGROUPMEMBR</td>
<td>{1} this contact does not belong to your group.</td>
</tr>
<tr>
<td>COOKIES_DISABLED</td>
<td>You have disabled cookies from being stored on your computer or turned off per-session cookies. Please enable this feature and try again.</td>
</tr>
<tr>
<td>COUNTRY_STATE_MISMATCH</td>
<td>The country and state/province are mismatched, the country is {1} and the state/province is {2}. Please enter a state/province short name that matches the country (see the &quot;state&quot; record for legal short names).</td>
</tr>
<tr>
<td>CREATEDFROM_REQD</td>
<td>Please enter a value for createdFrom.</td>
</tr>
<tr>
<td>CRNCY_MISMATCH_BASE_CRNCY</td>
<td>The currency you are registered to use is different from the base currency of this company.</td>
</tr>
<tr>
<td>CRNCY_NOT_UPDATED</td>
<td>The following currencies were not updated: {1}</td>
</tr>
<tr>
<td>CRNCY_RCRD_DELETED</td>
<td>This currency record has been deleted. You can create a new currency record at Lists &gt; Currencies.</td>
</tr>
<tr>
<td>CRNCY_REQD</td>
<td>currency expected for pricing element</td>
</tr>
<tr>
<td>CSTM_FIELD_KEY_REQD</td>
<td>The specified custom field key is missing.</td>
</tr>
<tr>
<td>CSTM_FIELD_VALUE_REQD</td>
<td>The specified custom field value is missing.</td>
</tr>
<tr>
<td>CUST_ARLEADY_HAS_ACCT</td>
<td>Attempting to provision a new account to a customer with an existing account</td>
</tr>
<tr>
<td>CUST_CNTR_USER_ACCESS_ONLY</td>
<td>This form is only accessible to customer center users.</td>
</tr>
<tr>
<td>CUST_LEAD_NOTGROUPMEMBR</td>
<td>{1} this customer or lead does not belong to your group.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The credit card transaction was denied by the issuing bank. Please try another card or contact the card issuer for more information.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The credit card has expired or the expiration date does not match the date on file with the card issuer. Please correct the expiration date or try another card.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The credit card transaction was denied due to insufficient funds. Please try another card or contact the card issuer for more information.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The credit card transaction could not be completed because the issuing bank was not available. Please try another card or wait a few minutes and try again.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Inactive card or card not authorized for card-not-present transactions. Please try another card or contact the card issuer for more information.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The card has reached the credit limit. Please try another card or contact the card issuer for more information.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Invalid card verification number. Please check to make sure you have provided the correct card verification number.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Invalid credit card account number. Please check to make sure you have provided the correct credit card account number.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The type of credit card provided is not accepted by this merchant. Please try another card or contact the merchant for more information.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The type of credit card provided is not accepted by this merchant. Please try another card or contact the merchant for more information.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Successful transaction.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The request is missing one or more required fields. Possible action: See the reply fields missingField_0...N for which fields are missing. Resend the request with the complete information.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>One or more fields in the request contains invalid data. Possible action: See the reply fields invalidField_0...N for which fields are invalid. Resend the request with the correct information.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The merchantReferenceCode sent with this authorization request matches the merchantReferenceCode of another authorization request that you sent in the last 15 minutes. Possible action: Resend the request with a unique merchantReferenceCode value.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Error: General system failure. See the documentation for your CyberSource client (SDK) for information about how to handle retries in the case of system errors.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Error: The request was received but there was a server timeout. This error does not include timeouts between the client and the server. Possible action: To avoid duplicating the order, do not resend the request until you have reviewed the order status in the Business Center. See the documentation for your CyberSource client (SDK) for information about how to handle retries in the case of system errors.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Error: The request was received, but a service did not finish running in time. Possible action: To avoid duplicating the order, do not resend the request until you have reviewed the order status in the Business Center. See the documentation for your CyberSource client (SDK) for information about how to handle retries in the case of system errors.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The issuing bank has questions about the request. You do not receive an authorization code programmatically, but you might receive one verbally by calling the processor. Possible action: Call your processor or the issuing bank to possibly receive a verbal authorization. For contact phone numbers, refer to your merchant bank information.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Expired card. You might also receive this if the expiration date you provided does not match the date the issuing bank has on file. Possible action: Request a different card or other form of payment.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>General decline of the card. No other information provided by the issuing bank. Possible action: Request a different card or other form of payment.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Insufficient funds in the account. Possible action: Request a different card or other form of payment.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Stolen or lost card. Possible action: Review the customers information and determine if you want to request a different card from the customer.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Issuing bank unavailable. Possible action: Wait a few minutes and resend the request.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Inactive card or card not authorized for card-not-present transactions. Possible action: Request a different card or other form of payment.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The card has reached the credit limit. Possible action: Request a different card or other form of payment.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Invalid card verification number. Possible action: Request a different card or other form of payment.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The customer matched an entry on the processors negative file. Possible action: Review the order and contact the payment processor.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Invalid account number. Possible action: Request a different card or other form of payment.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The card type is not accepted by the payment processor. Possible action: Request a different card or other form of payment. Also, check with CyberSource Customer Support to make sure your account is configured correctly.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>General decline by the processor. Possible action: Request a different card or other form of payment.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>There is a problem with your CyberSource merchant configuration. Possible action: Do not resend the request. Contact Customer Support to correct the configuration problem.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The requested amount exceeds the originally authorized amount. Occurs, for example, if you try to capture an amount larger than the original authorization amount. This reason code only applies if you are processing a capture through the API. See Using the API for Captures and Credits. Possible action: Issue a new authorization and capture request for the new amount.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Processor failure. Possible action: Tell the customer the payment processing system is unavailable temporarily, and to try their order again in a few minutes.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The authorization has already been captured. This reason code only applies if you are processing a capture through the API. See Using the API for Captures and Credits. Possible action: No action required.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The requested transaction amount must match the previous transaction amount. This reason code only applies if you are processing a capture or credit through the API. See Using the API for Captures and Credits. Possible action: Correct the amount and resend the request.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The card type sent is invalid or does not correlate with the credit card number. Possible action: Ask your customer to verify that the card is really the type that they indicated in your Web store, then resend the request.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The request ID is invalid. This reason code only applies when you are processing a capture or credit through the API. See Using the API for Captures and Credits. Possible action: Request a new authorization, and if successful, proceed with the capture.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>You requested a capture through the API, but there is no corresponding, unused authorization record. Occurs if there was not a previously successful authorization request or if the previously successful authorization has already been used by another capture request. This reason code only applies when you are processing a capture through the API. See Using the API for Captures and Credits. Possible action: Request a new authorization, and if successful, proceed with the capture.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The capture or credit is not voidable because the capture or credit information has already been submitted to your processor. Or, you requested a void for a type of transaction that cannot be voided. This reason code applies only if you are processing a void through the API. See Using the API for Voids for information about voids. Possible action: No action required.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>You requested a credit for a capture that was previously voided. This reason code applies only if you are processing a void through the API. See Using the API for Voids for information about voids. Possible action: No action required.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Error: The request was received, but there was a timeout at the payment processor. Possible action: To avoid duplicating the transaction, do not resend the request until you have reviewed the transaction status in the Business Center.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>The authorization request was approved by the issuing bank but declined by CyberSource based on your Smart Authorization settings. Possible action: Do not capture the authorization without further review. Review the ccAuthReply_avsCode, ccAuthReply_cvCode, and ccAuthReply_authFactorCode fields to determine why CyberSource rejected the request.</td>
</tr>
<tr>
<td>CYBERSOURCE_ERROR</td>
<td>Unable to process credit card transaction. The code returned from CyberSource is not a recognized reason code. Please contact NetSuite support.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CYCLE_IN_PROJECT_PLAN</td>
<td>The changes made to this entity have cause a cycle in the project plan. Select a different parent and/or predecessors to avoid the cycle.</td>
</tr>
<tr>
<td>DASHBOARD_LOCKED</td>
<td>Your dashboard has been set up and locked by an administrator. Please contact them for details.</td>
</tr>
<tr>
<td>DATA_MUST_BE_UNIQUE</td>
<td>The update failed because every entry in this column must be unique.</td>
</tr>
<tr>
<td>DATA_REQD</td>
<td>You need to provide a proper value for the required field: {1}.</td>
</tr>
<tr>
<td>DATA_REQD</td>
<td>You are missing the following required field(s):{1}</td>
</tr>
<tr>
<td>DATE_EXPECTED</td>
<td>You entered '{1}' into a field where a calendar date was expected. Please go back and change this value to the correct date.</td>
</tr>
<tr>
<td>DATE_PARAM_REQD</td>
<td>missing date parameter</td>
</tr>
<tr>
<td>DEFAULT_CUR_REQD</td>
<td>Default currency cannot be null</td>
</tr>
<tr>
<td>DEFAULT_EXPENSE_ACCT_REQD</td>
<td>A default expense account must be specified in order to activate items on the list. Go to Setup &gt; Set Up Payroll and click the Default Accounts subtab. In the Payroll Expenses Account field, choose a default general ledger account for your payroll expenses. Then, click Save.</td>
</tr>
<tr>
<td>DEFAULT_ISSUE_OWNER_REQD</td>
<td>There is no default owner for the issue role {1}. This operation cannot be completed until this is corrected.</td>
</tr>
<tr>
<td>DEFAULT_LIAB_ACCT_REQD</td>
<td>A default liability account must be specified in order to activate items on the list. Go to Setup &gt; Set Up Payroll and click the Default Accounts subtab. In the Payroll Liabilities Account field, choose a default general ledger account for your payroll liabilities. Then, click Save.</td>
</tr>
<tr>
<td>DEFAULT_ROLE_REQD</td>
<td>Login Failed because you do not have a default role for the company and email entered. Please Try Again.</td>
</tr>
<tr>
<td>DEFAULT_TYPE_DELETE_DISALLWD</td>
<td>You cannot delete default types</td>
</tr>
<tr>
<td>DEFERRAL_ACCT_REQD</td>
<td>Lines with amortization templates must have a deferral account.</td>
</tr>
<tr>
<td>DEFERRAL_ACCT_REQD</td>
<td>Lines with revenue recognition templates must have a deferral account.</td>
</tr>
<tr>
<td>DEFERRED_REV_REC_ACCT_REQD</td>
<td>The {1} item does not have a Deferred Revenue Account specified. Please assign the item a Deferred Revenue Account using the standard User Interface, and then re-import the transaction.</td>
</tr>
<tr>
<td>DEPT_IN_USE</td>
<td>Your classes cannot be converted to departments because your existing department records are referred to by transactions or other records. These department records cannot be overwritten.</td>
</tr>
<tr>
<td>DFRNT_SWAP_PRICE_LEVELS_REQD</td>
<td>Please select different price levels to swap prices.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DISALLOWD_IP_ADDRESS</td>
<td>The specified IP address rules must allow the login of your current IP Address. Your current IP address is {1}. For information on entering IP address rules, click Help at the top of the page.</td>
</tr>
<tr>
<td>DISCOUNT_ACCT_SETUP_REQD</td>
<td>Please &lt;a href='app/setup/acctsetup.nl'&gt;Set Up Discount Accounts&lt;/a&gt; first.</td>
</tr>
<tr>
<td>DISCOUNT_DISALLOWD</td>
<td>You have attempted to save this transaction with one or more discounts and where all items have Permit Discount = Never. You must change one of the items to permit a discount, add a new item without the restriction or remove the discount from the transaction.</td>
</tr>
<tr>
<td>DISCOUNT_EXCEED_TOTAL</td>
<td>Discount can not exceed item total.</td>
</tr>
<tr>
<td>DISTRIBUTION_REQD_ONE_DAY_BFORE</td>
<td>All items must be distributed at least one day before they may be transferred.</td>
</tr>
<tr>
<td>DROP_SHIP_OR_SPECIAL_ORD_ALLWD</td>
<td>Items can be Drop Ship or Special Order but not both</td>
</tr>
<tr>
<td>DUE_DATE_BFORE_START_DATE</td>
<td>Due date occurs before start date</td>
</tr>
<tr>
<td>DUE_DATE_REQD</td>
<td>Please enter a value for {1} Due Date</td>
</tr>
<tr>
<td>DUPLICATE_INVENTORY_NUM</td>
<td>Duplicate inventory number found in entry: {1}</td>
</tr>
<tr>
<td>DUPLICATE_INVENTORY_NUM</td>
<td>Duplicate inventory number found on different lines of transaction</td>
</tr>
<tr>
<td>DUPLICATE_KEYS</td>
<td>This record contains duplicated key or keys. Please correct it before next update.</td>
</tr>
<tr>
<td>DUPLICATE_NAME_FOR_PRD</td>
<td>Please choose a different period name. &quot;(1)&quot; is already taken.</td>
</tr>
<tr>
<td>DUPLICATE_NAME_FOR_ROLE</td>
<td>Please choose a different role name. &quot;(1)&quot; is already taken.</td>
</tr>
<tr>
<td>DUPLICATE_USER_NAME</td>
<td>A user with this name already exists.</td>
</tr>
<tr>
<td>DUP_ACCT_NAME</td>
<td>The account name you have chosen is already used.&lt;br&gt;Go &lt;a href='javascript:history.go(-1);';&gt;back&lt;/a&gt;, change the name and resubmit.</td>
</tr>
<tr>
<td>DUP_ACCT_NUM</td>
<td>The account number you have chosen is already used.&lt;br&gt;Go &lt;a href='javascript:history.go(-1);';&gt;back&lt;/a&gt;, change the number and resubmit.</td>
</tr>
<tr>
<td>DUP_ACCT_ON_TRANS</td>
<td>This transaction has duplicate accounts. The main line of the transaction and the line labeled '{1}' both use the account named '{2}'.</td>
</tr>
<tr>
<td>DUP_BIN</td>
<td>There is already another bin with that number. Please choose a bin number that is not used by another bin.</td>
</tr>
<tr>
<td>DUP_CATEGORY</td>
<td>This category already exists</td>
</tr>
<tr>
<td>DUP_CATEGORY_NAME</td>
<td>A category already exists with that name. Go &lt;a href='javascript:history.go(-1);';&gt;back&lt;/a&gt;, change the name and resubmit.</td>
</tr>
<tr>
<td>DUP_COLOR_THEME</td>
<td>This color theme already exists</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DUP_CSTM_FIELD</td>
<td>This custom field already exists</td>
</tr>
<tr>
<td>DUP_CSTM_LAYOUT</td>
<td>This custom layout already exists</td>
</tr>
<tr>
<td>DUP_CSTM_LIST</td>
<td>There is already a Custom List or Custom List element with that name</td>
</tr>
<tr>
<td>DUP_CSTM_RCRD</td>
<td>There is already a Custom Record with that name</td>
</tr>
<tr>
<td>DUP_CSTM_RCRD_ENTRY</td>
<td>There is already a Custom Record Entry with that name</td>
</tr>
<tr>
<td>DUP_CSTM_TAB</td>
<td>This custom tab already exists</td>
</tr>
<tr>
<td>DUP_EMPL_EMAIL</td>
<td>There is already an employee with external access to this account using that email address. All employees with external access must have a unique email address for login purposes. Go to the back and change the email address and resubmit.</td>
</tr>
<tr>
<td>DUP_EMPL_ENTITY_NAME</td>
<td>There is already an employee with external access to this account using that entity name. All employees with external access must have a unique entity name for login purposes. Go to the back and change the entity name and resubmit.</td>
</tr>
<tr>
<td>DUP_EMPL_TMPLT</td>
<td>There is already an employee template with that name. Go to the back and change the template name and resubmit.</td>
</tr>
<tr>
<td>DUP_ENTITY</td>
<td>This entity already exists</td>
</tr>
<tr>
<td>DUP_ENTITY_EMAIL</td>
<td>There is already an external entity (e.g., customer, vendor, or employee) with access to this account using that email address. All external entities with access must have a unique email address for login purposes.</td>
</tr>
<tr>
<td>DUP_ENTITY_NAME</td>
<td>There is already an external entity (e.g., customer, vendor, or employee) with access to this account using that entity name. All external entities with access must have a unique entity name for login purposes.</td>
</tr>
<tr>
<td>DUP_FEDEX_ACCT_NUM</td>
<td>There is an existing NetSuite registration for FedEx account number {1}.</td>
</tr>
<tr>
<td>DUP_FINANCL_STATMNT_LAYOUT</td>
<td>This financial statement layout already exists.</td>
</tr>
<tr>
<td>DUP_INFO_ITEM</td>
<td>This information item already exists</td>
</tr>
<tr>
<td>DUP_ISSUE_NAME_OR_NUM</td>
<td>You cannot set {1: issue record name} {2: issue number} to be a duplicate of itself or one of its duplicates.</td>
</tr>
<tr>
<td>DUP_ITEM</td>
<td>Uniqueness error - there is already an item with that name or name/parent combination.</td>
</tr>
<tr>
<td>DUP_ITEM_LAYOUT</td>
<td>This item/category layout already exists</td>
</tr>
<tr>
<td>DUP_ITEM_NAME</td>
<td>There is already an item with that name. &lt;br&gt; Go to the back and change the name and resubmit.</td>
</tr>
<tr>
<td>DUP_ITEM_OPTION</td>
<td>A child item child with that combination of options already exists</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DUP_ITEM_TMPLT</td>
<td>This item/category template already exists</td>
</tr>
<tr>
<td>DUP_MATRIX_OPTN_ABBRV</td>
<td>Matrix option '{1}' already uses that abbreviation. Please choose another.</td>
</tr>
<tr>
<td>DUP_MEMRZD_TRANS</td>
<td>There is already a Memorized Transaction with that name. &lt;br&gt;Go &lt;a href=&quot;javascript:history.go(-1);&quot;&gt;back&lt;/a&gt;, change the name and resubmit.</td>
</tr>
<tr>
<td>DUP_NAME</td>
<td>That name is already in use. &lt;br&gt;Go &lt;a href=&quot;javascript:history.go(-1);&quot;&gt;back&lt;/a&gt;, change the name and resubmit.</td>
</tr>
<tr>
<td>DUP_PAYROLL_ITEM</td>
<td>There is already a payroll item named {1}</td>
</tr>
<tr>
<td>DUP_PRESNTN_CAT</td>
<td>This presentation category already exists</td>
</tr>
<tr>
<td>DUP_RCRD</td>
<td>A {1} already exists with that name. &lt;br&gt;Go &lt;a href=&quot;javascript:history.go(-1);&quot;&gt;back&lt;/a&gt;, change the name and resubmit.</td>
</tr>
<tr>
<td>DUP_RCRD_LINK</td>
<td>This record already exists</td>
</tr>
<tr>
<td>DUP_SALES_TAX_ITEM</td>
<td>You have entered a duplicate Sales Tax Item. &lt;br&gt;Go &lt;a href=&quot;javascript:history.go(-1);&quot;&gt;back&lt;/a&gt;, change the name, city, state or zip code and resubmit.</td>
</tr>
<tr>
<td>DUP_SHIPPING_ITEM</td>
<td>You have entered a duplicate Shipping Item. &lt;br&gt;Go &lt;a href=&quot;javascript:history.go(-1);&quot;&gt;back&lt;/a&gt;, change the name, city, state or zip code and resubmit.</td>
</tr>
<tr>
<td>DUP_SHORT_NAME</td>
<td>Duplicate short name</td>
</tr>
<tr>
<td>DUP_SITE_THEME</td>
<td>This site theme already exists</td>
</tr>
<tr>
<td>DUP_TAX_CODE</td>
<td>You have entered a duplicate Tax Code. &lt;br&gt;Go &lt;a href=&quot;javascript:history.go(-1);&quot;&gt;back&lt;/a&gt;, change the name and resubmit.</td>
</tr>
<tr>
<td>DUP_TAX_CODE</td>
<td>You have entered a duplicate Tax Code. &lt;br&gt;Go &lt;a href=&quot;javascript:history.go(-1);&quot;&gt;back&lt;/a&gt;, change the name and resubmit.</td>
</tr>
<tr>
<td>DUP_TRACKING_NUM</td>
<td>You entered the following tracking number twice: {1}. Note that a single tracking number may not contain spaces or commas. A space or comma will be interpreted as the separator between different tracking numbers. For example, '1029 3847 465' will be interpreted as 3 different tracking numbers. It should be entered without spaces: '10293847465'.</td>
</tr>
<tr>
<td>DUP_UPS_ACCT_NUM</td>
<td>There is an existing NetSuite registration for UPS account number {1}.</td>
</tr>
<tr>
<td>DUP_VENDOR_EMAIL</td>
<td>There is already a vendor with external access to this account using that email address. All vendors with external access must have a unique email address for login purposes. Go &lt;a href=&quot;javascript:history.go(-1);&quot;&gt;back&lt;/a&gt;, change the email address and resubmit.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DUP_VENDOR_NAME</td>
<td>There is already a vendor using that entity name. All vendors must have a unique entity name. Go &lt;a href=&quot;javascript:history.go(-1);&quot;&gt;back&lt;/a&gt;, change the entity name and resubmit.</td>
</tr>
<tr>
<td>EDITION_DSNT_SUPRT_WORLDPAY</td>
<td>WorldPay is not supported in this edition.</td>
</tr>
<tr>
<td>EMAIL_ADDRS_REQD</td>
<td>Please enter your email address</td>
</tr>
<tr>
<td>EMAIL_ADDRS_REQD_TO_NOTIFY</td>
<td>Please enter an email address for this company. A notification email will be sent when this case record is saved.</td>
</tr>
<tr>
<td>EMAIL_ADDRS_REQD_TO_NOTIFY</td>
<td>The recipient you are sending this email does not have an email address. Please enter one and try again</td>
</tr>
<tr>
<td>EMAIL_REQD</td>
<td>You must enter a valid email address in order to email the transaction.</td>
</tr>
<tr>
<td>EMAIL_REQD_ACCT_PROVISION</td>
<td>Cannot provision an account without an Email address for this customer: Was external access granted?</td>
</tr>
<tr>
<td>EMPL_IN_USE</td>
<td>You can’t delete this employee, as commissions have been calculated for this employee.</td>
</tr>
<tr>
<td>EMPL_IN_USE</td>
<td>You can’t delete this employee, as it is or has been referenced by other employees as a supervisor.</td>
</tr>
<tr>
<td>ERROR_IN_TERRITORY_ASSGNMNT</td>
<td>Error Performing Initial Round_Robin Assignment for Territory: {1}</td>
</tr>
<tr>
<td>ERROR_IN_TERRITORY_ASSGNMNT</td>
<td>Error Performing Round_Robin Assignment for Territory: {1}</td>
</tr>
<tr>
<td>ERROR_PRCSSNG_TRANS</td>
<td>There were errors processing the selected transactions. Please process them individually for more information.</td>
</tr>
<tr>
<td>ERROR_SENDING_TRAN_EMAIL</td>
<td>The transaction was entered successfully, but an unexpected error occurred while sending the transaction email {1}</td>
</tr>
<tr>
<td>EVENT_ID_NOT_FOUND</td>
<td>Event ID not found</td>
</tr>
<tr>
<td>EXCEEDED_MAX_ALLWD_LOC</td>
<td>You have reached the maximum allowance of {1} location records. If you need to create additional location records, please contact our NetSuite Customer Support team for assistance</td>
</tr>
<tr>
<td>EXCEEDED_MAX_CONCUR_RQST</td>
<td>The maximum number of concurrent requests has been exceeded. Please try your request again when an existing session has completed.</td>
</tr>
<tr>
<td>EXCEEDED_MAX_EMAILS</td>
<td>The merge exceeds the number of bulk merge emails allotted to your account this year. This account has {1} more bulk emails that can be sent this year. Please contact your NetSuite account manager to purchase additional block of emails.</td>
</tr>
<tr>
<td>EXCEEDED_MAX_EMAILS</td>
<td>This campaign email event exceeds the number of emails {(1)} that can be sent per event without setting up a default campaign domain or specifying one on the campaign email template.</td>
</tr>
<tr>
<td>EXCEEDED_MAX_EMAILS</td>
<td>This merge operation exceeds the number of emails {(1)} that can be sent per execution without setting up a bulk merge domain or specifying one on the email template.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>EXCEEDED_MAX_FIELD_LENGTH</td>
<td>Address line 1 cannot exceed 35 characters. Please check the shipper and recipient address to ensure the “Address 1” field is a maximum of 35 characters.</td>
</tr>
<tr>
<td>EXCEEDED_MAX_FIELD_LENGTH</td>
<td>Address line 2 cannot exceed 35 characters. Please check the shipper and recipient address to ensure the “Address 2” field is a maximum of 35 characters.</td>
</tr>
<tr>
<td>EXCEEDED_MAX_FIELD_LENGTH</td>
<td>The field (1) contained more than the maximum number (2) of characters allowed.</td>
</tr>
<tr>
<td>EXCEEDED_MAX_FIELD_LENGTH</td>
<td>The string “[1]” contained more than the maximum number of characters allowed.</td>
</tr>
<tr>
<td>EXCEEDED_MAX_FIELD_LENGTH</td>
<td>Too many characters for a field</td>
</tr>
<tr>
<td>EXCEEDED_MAX_MATRIX_OPTNS</td>
<td>The total combination of subitems you have selected exceeds the maximum allowed of 2000. Please choose fewer options on the matrix tab.</td>
</tr>
<tr>
<td>EXCEEDED_MAX_MATRIX_OPTNS</td>
<td>The total combination of subitems you have selected exceeds the maximum allowed of 2000. Please choose fewer options on the matrix tab.</td>
</tr>
<tr>
<td>EXCEEDED_MAX_SHIP_PACKAGE</td>
<td>The maximum number of custom shipping packages has been exceeded: (1). Please reduce item quantities to generate fewer packages, or enter the packages manually.</td>
</tr>
<tr>
<td>EXCEEDED_MAX_TIME</td>
<td>The operation has exceeded maximum allowed time for completion. Operation aborted.</td>
</tr>
<tr>
<td>EXCEEDED_MAX_TRANS_LINES</td>
<td>Transactions may not contain more than (1) lines.</td>
</tr>
<tr>
<td>EXCEEDED_PER_TRANS_MAX</td>
<td>Exceeded per transaction maximum on account (1)</td>
</tr>
<tr>
<td>EXCEEDED_RQST_SIZE_LIMIT</td>
<td>You have exceeded the permitted request size limit ((1))</td>
</tr>
<tr>
<td>EXCEEDS_ALLWD_LICENSES</td>
<td>Adding access for this user exceeds the number of licenses you have purchased. To add another user, you must first remove access from an existing user or contact NetSuite to purchase additional licenses.</td>
</tr>
<tr>
<td>EXCEEDS_ALLWD_LICENSES</td>
<td>Adding a (1) would exceed the number of licenses you have purchased. Please contact NetSuite for additional licenses.</td>
</tr>
<tr>
<td>EXPIRED_SEARCH_CRITERIA</td>
<td>Your search criteria expired. The criteria for a given search generally expire after 15 minutes of inactivity. Please return to the search definition page and re-submit your search.</td>
</tr>
<tr>
<td>EXT_CAT_LINK_SETUP_REQD</td>
<td>Error - you have not properly set up links from your External Catalog Site back into (1)!</td>
</tr>
<tr>
<td>FAILED_FEDEX_LABEL_VOID</td>
<td>Failed FedEx Label Void</td>
</tr>
<tr>
<td>FAILED_FORM_VALIDATION</td>
<td>Form validation failed. You cannot submit this record.</td>
</tr>
<tr>
<td>FAILED_UPS_LABEL_VOID</td>
<td>Failed UPS Label Void</td>
</tr>
<tr>
<td>FAX_NUM_REQD</td>
<td>You must enter a fax number.</td>
</tr>
<tr>
<td>FAX_NUM_REQD</td>
<td>You must enter a fax number for this recipient before performing a fax merge operation.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>FAX_NUM_REQD</td>
<td>You must enter a valid fax number in order to fax the transaction.</td>
</tr>
<tr>
<td>FAX_SETUP_REQD</td>
<td>Before you can send faxes, you need to go to the &lt;a href='/app/setup/printing.nl'&gt;Set Up Printing, Fax &amp; Email&lt;/a&gt; page and set up the fax service.</td>
</tr>
<tr>
<td>FEATURE_DISABLED</td>
<td>The feature '{1}' required to access this page is not enabled in this account.</td>
</tr>
<tr>
<td>FEATURE_UNAVAILABLE</td>
<td>&lt;b&gt;{1} Trial does not allow access to this feature.&lt;/b&gt; If you would like more information about this feature, please contact your account manager.</td>
</tr>
<tr>
<td>FEATURE_UNAVAILABLE</td>
<td>Error - This business does not have the External Catalog Site feature enabled.</td>
</tr>
<tr>
<td>FEATURE_UNAVAILABLE</td>
<td>Test Drive does not allow access to this feature. If you would like more information about this feature, please contact your account manager.</td>
</tr>
<tr>
<td>FEATURE_UNAVAILABLE</td>
<td>That feature is only available to Plus users</td>
</tr>
<tr>
<td>FEATURE_UNAVAILABLE</td>
<td>The {1} feature is not available to your company.</td>
</tr>
<tr>
<td>FEATURE_UNAVAILABLE</td>
<td>This feature is not available to your company.</td>
</tr>
<tr>
<td>FEDEX_ACCT_REQD</td>
<td>The FedEx Account Number has not been set.</td>
</tr>
<tr>
<td>FEDEX_CANT_INTEGRATE_FULFILL</td>
<td>The fulfillment cannot be integrated with {1} because the Shipping Integration Carrier is set to UPS.</td>
</tr>
<tr>
<td>FEDEX_DROPOFF_TYP_REQD</td>
<td>The FedEx Dropoff Type has not been set.</td>
</tr>
<tr>
<td>FEDEX_INVALID_ACCT_NUM</td>
<td>This account number was not recognized by FedEx. Please re-enter your account number, or contact FedEx to open a new account.</td>
</tr>
<tr>
<td>FEDEX_ITEM_CONTENTS_REQD</td>
<td>For international shipments, {1} requires specific information about the item contents.</td>
</tr>
<tr>
<td>FEDEX_METER_NOT_RETRIEVED</td>
<td>A FedEx Meter Number was not retrieved for account number {1}. Please try your request again in a few minutes.</td>
</tr>
<tr>
<td>FEDEX_METER_REQD</td>
<td>The FedEx Meter Number has not been set.</td>
</tr>
<tr>
<td>FEDEX_ONE_PACKG_ALLWD</td>
<td>The selected FedEx service allows only one package per fulfillment. If more than one package is required, please break up the shipment into multiple fulfillments of one package each.</td>
</tr>
<tr>
<td>FEDEX_ORIGIN_COUNTRY_US_REQD</td>
<td>The origin country must be United States (US) for all Item Fulfillments when using a FedEx shipping method.</td>
</tr>
<tr>
<td>FEDEX_RATING_SRVC_UNAVAILBL</td>
<td>The FedEx rating services application is currently unavailable. Please try your request again in a few minutes.</td>
</tr>
<tr>
<td>FEDEX_REG_NOT_FOUND</td>
<td>A valid FedEx Registration was not found for the specified location:</td>
</tr>
<tr>
<td>FEDEX_SHIP_SRVC_REQD</td>
<td>The FedEx Shipping Service has not been set.</td>
</tr>
<tr>
<td>FEDEX_SHIP_SRVC_UNAVAILBL</td>
<td>The FedEx shipping services application is currently unavailable. Please try your request again in a few minutes.</td>
</tr>
<tr>
<td><strong>Error Code Returned</strong></td>
<td><strong>Long Description or Message</strong></td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FEDEX_UNSUPRTD_ORIGIN_COUNTRY</td>
<td>The origin country [1] is currently not supported for Item Fulfillments when using a FedEx shipping method.</td>
</tr>
<tr>
<td>FEDEX_USD_EXCHANGE_RATE_REQD</td>
<td>Cannot retrieve FedEx realtime rates: USD Exchange Rate is required when requesting FedEx realtime rates.</td>
</tr>
<tr>
<td>FEDEX_VOID_ERROR</td>
<td>The FedEx Void failed due to a system error.</td>
</tr>
<tr>
<td>FED_WITHHOLDING_REQD</td>
<td>Your employee record does not have current Federal Withholding information. Please contact your supervisor to set up your record with the appropriate information.</td>
</tr>
<tr>
<td>FIELD_CALL_DATE_REQD</td>
<td>Missing Required Field: Call Date</td>
</tr>
<tr>
<td>FIELD_DEFN_REQD</td>
<td>Field definition not found</td>
</tr>
<tr>
<td>FIELD_NOT_SETTABLE_ON_ADD</td>
<td>You are not allowed to set the nsKey for a record</td>
</tr>
<tr>
<td>FIELD_PARAM_REQD</td>
<td>Please enter a value for [1].</td>
</tr>
<tr>
<td>FIELD_PARAM_REQD</td>
<td>Please enter values for [1].</td>
</tr>
<tr>
<td>FIELD_REQD</td>
<td>Mandatory Field Missing</td>
</tr>
<tr>
<td>FIELD_REQD</td>
<td>You must first select a field</td>
</tr>
<tr>
<td>FILE_ALREADY_EXISTS</td>
<td>A file with the same name already exists in the selected folder.</td>
</tr>
<tr>
<td>FILE_ALREADY_EXISTS</td>
<td>Note: You are attempting to upload a file with a name matching an existing file in the selected folder. Please rename this file or select another folder, and then upload your file.</td>
</tr>
<tr>
<td>FILE_DISALLWD_IN_ROOT_FLDR</td>
<td>You attempted to copy a file to the root directory. Only folders can exist in the root directory.</td>
</tr>
<tr>
<td>FILE_DISALLWD_IN_ROOT_FLDR</td>
<td>You attempted to move a file to the root directory. Only folders can exist in the root directory.</td>
</tr>
<tr>
<td>FILE_MISSING</td>
<td>File Missing</td>
</tr>
<tr>
<td>FILE_NOT_DOWNLOADABLE</td>
<td>Illegal request for a file that isn't downloadable</td>
</tr>
<tr>
<td>FILE_NOT_FOUND</td>
<td>File/Media Item [1] not found.</td>
</tr>
<tr>
<td>FILE_NOT_FOUND</td>
<td>File not found. Please try your download again.</td>
</tr>
<tr>
<td>FILE_REQD</td>
<td>You must upload a file before creating this media item</td>
</tr>
<tr>
<td>FILE_UPLOAD_IN_PROGRESS</td>
<td>Files are currently being uploaded to this account.</td>
</tr>
<tr>
<td>FILTER_BY_AMT_REQD</td>
<td>Please enter an amount to filter by.</td>
</tr>
<tr>
<td>FINANCE_CHARGE_SETUP_REQD</td>
<td>Please set &lt;a href=&quot;/app/setup/finchargepref.nl&quot;&gt;Finance Charge Preferences&lt;/a&gt; first.</td>
</tr>
<tr>
<td>FIRST_LAST_NAMES_REQD</td>
<td>Please enter both your first and last name.</td>
</tr>
<tr>
<td>FIRST_QTY_BUCKET_MUST_BE_ZERO</td>
<td>Quantity defined for first quantity bucket must be zero</td>
</tr>
<tr>
<td>FLD_VALUE_TOO_LARGE</td>
<td>Value for field [1] is too large to be processed.</td>
</tr>
<tr>
<td>FOLDER_ALREADY_EXISTS</td>
<td>A folder with the same name already exists in the selected folder.</td>
</tr>
<tr>
<td>FORMULA_ERROR</td>
<td>Your formula has an error in it. It could resolve to the wrong datatype, use an unknown function, or have a syntax error. Please go back, correct the formula, and re-submit.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FORM_RESUBMISSION_REQD</td>
<td>You have logged in to a different user since you navigated to this form. You must re-submit this form as the new user.</td>
</tr>
<tr>
<td>FORM_UNAVAILBL_ONLINE</td>
<td>This form is not available online</td>
</tr>
<tr>
<td>FRIENDLY_NAME_REQD</td>
<td>Missing Friendly Name. Friendly Name is a required field and it cannot be null or empty.</td>
</tr>
<tr>
<td>FULFILL_REQD_FIELDS_MISSING</td>
<td>For the listed items, please edit the item record and provide values for the specified fields, and retry the fulfillment.</td>
</tr>
<tr>
<td>FULFILL_REQD_FIELDS_MISSING</td>
<td>The {1} field is required to complete this fulfillment. Please return to the International tab on the item fulfillment and provide a value for the specified field and retry the fulfillment.</td>
</tr>
<tr>
<td>FULFILL_REQD_PARAMS_MISSING</td>
<td>Could not perform operation '{1}' since (2) parameter was not set.</td>
</tr>
<tr>
<td>FULL_DISTRI_REQD</td>
<td>You must fully distribute all (1) numbers for (1) numbered items.</td>
</tr>
<tr>
<td>FULL_USERS_REQD_TO_INTEGRATE</td>
<td>Only full (1) users can integrate with partners.</td>
</tr>
<tr>
<td>FX_MALFORMED_RESPONSE</td>
<td>Received malformed response from Foreign Exchange source.</td>
</tr>
<tr>
<td>FX_RATE_REQD_FEDEX_RATE</td>
<td>Cannot retrieve (1) realtime rates: (2) Exchange Rate is required when requesting (3) realtime rates.</td>
</tr>
<tr>
<td>GETALL_RCRD_TYPE_REQD</td>
<td>The getAll record type is required.</td>
</tr>
<tr>
<td>GIFT_CERT_AMT_EXCEED_AVAILBL</td>
<td>Gift certificate redemption amount exceeds available amount on the gift certificate.</td>
</tr>
<tr>
<td>GIFT_CERT_AUTH_ALREADY_EXISTS</td>
<td>Gift certificate authorization code (1) already exists</td>
</tr>
<tr>
<td>GIFT_CERT_CAN_BE_USED_ONCE</td>
<td>A gift certificate may only be used once on a transaction.</td>
</tr>
<tr>
<td>GIFT_CERT_CODE_REQD</td>
<td>Gift certificate codes are missing</td>
</tr>
<tr>
<td>GIFT_CERT_CODE_REQD</td>
<td>Missing gift certificate authorization code(s). Please go back and enter authorization codes on the (1).</td>
</tr>
<tr>
<td>GIFT_CERT_CODE_REQD</td>
<td>You must specify a gift certificate code.</td>
</tr>
<tr>
<td>GIFT_CERT_IN_USE</td>
<td>Another user is using gift certificate (1)</td>
</tr>
<tr>
<td>GIFT_CERT_IN_USE</td>
<td>Gift certificate code (1) is already in use</td>
</tr>
<tr>
<td>GROUP_DSNT_EXIST</td>
<td>That group does not exist</td>
</tr>
<tr>
<td>GROUP_REQD</td>
<td>You cannot perform a bulk merge operation with an empty group</td>
</tr>
<tr>
<td>GROUP_TYPE_REQD</td>
<td>The group type is required.</td>
</tr>
<tr>
<td>GRTR_QTY_PRICE_LEVEL_REQD</td>
<td>Each quantity pricing level must be greater than the previous quantity pricing level.</td>
</tr>
<tr>
<td>ILLEGAL_ID</td>
<td>Illegal ID. Please enter a name.</td>
</tr>
<tr>
<td>ILLEGAL_PERIOD_STRUCTURE</td>
<td>Illegal period structure. Date (1) is in multiple periods.</td>
</tr>
<tr>
<td>INACTIVE_RCRD_FOR_ROLE</td>
<td>The record for this role has been made inactive.</td>
</tr>
<tr>
<td>INAVLIDFILE_TYP</td>
<td>A change has been made to this file's format. You cannot upload this type of file.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>INAVLID_FILE_TYP</td>
<td>You attempted to upload a restricted file type. Please try again with a selection from the list below:</td>
</tr>
<tr>
<td>INAVLID_ITEM_TYP</td>
<td>Invalid item type [1] for item [2].</td>
</tr>
<tr>
<td>INAVLID_PRICING_MTRX</td>
<td>Invalid Quantity Pricing Matrix for quantity level (1) : Quantity (2), Base Price (3)</td>
</tr>
<tr>
<td>INCOMPATIBLE_ACCT_CHANGE</td>
<td>The account change you have made is incompatible with old transactions. If you need to swap two accounts, you need to do it in 3 steps. For example, to change the income and asset accounts for an item:&lt;ul&gt;&lt;li&gt;(1) Change the income account to a temporary account and save&lt;/li&gt;&lt;li&gt;(2) Change asset account to the old income account and save&lt;/li&gt;&lt;li&gt;(3) Change the income to the old asset account and save&lt;/li&gt;&lt;/ul&gt;Please contact customer support if you need assistance with this.</td>
</tr>
<tr>
<td>INCOMPATIBLE_ACCT_CHANGE</td>
<td>The account change you have made is incompatible with old transactions. Please either change the account selection appropriately or do not request to update past transactions.</td>
</tr>
<tr>
<td>INCOMPLETE_BILLING_ADDR</td>
<td>Billing address is incomplete.</td>
</tr>
<tr>
<td>INCOMPLETE_FILE_UPLOAD</td>
<td>The upload did not complete correctly. Please try uploading the file again. If you have repeatedly received this error message, please send mail to Technical Support.</td>
</tr>
<tr>
<td>INCRCT_ORD_INFO</td>
<td>The order contains incorrect information and was not placed.</td>
</tr>
<tr>
<td>INITIALIZE_ARG_REQD</td>
<td>The initialize reference id is required.</td>
</tr>
<tr>
<td>INITIALIZE_ARG_REQD</td>
<td>The initialize reference type is required.</td>
</tr>
<tr>
<td>INITIALIZE_ARG_REQD</td>
<td>The initialize type is required.</td>
</tr>
<tr>
<td>INITIALIZE_AUXREF_REQD</td>
<td>The initialize auxReference type is required.</td>
</tr>
<tr>
<td>INSUFCNT_NUM_PRDS_FOR_REV_REC</td>
<td>Not enough accounting periods in range specified for revenue recognition.</td>
</tr>
<tr>
<td>INSUFCNT_OPEN_PRDS_FOR_REV_REC</td>
<td>Not enough open accounting periods available for revenue recognition.</td>
</tr>
<tr>
<td>INSUFFICIENT_CHARS_IN_SEARCH</td>
<td>Global searches must contain at least three characters to prevent excessive matches.</td>
</tr>
<tr>
<td>INSUFFICIENT_FLD_PERMISSION</td>
<td>You are attempting to read an unauthorized field: [1]</td>
</tr>
<tr>
<td>INSUFFICIENT_FLD_PERMISSION</td>
<td>You cannot access this search because it includes restricted fields. Please contact your administrator.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>For security reasons, only an administrator is allowed to edit an administrator record.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>Global search is not permitted from this role.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>Insufficient privileges</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>Your issue DB access has been inactivated. Please contact your issue DB administrator.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>Your current login role does not have an associated Issue Role. Please change to a different role or contact your Issue administrator.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>Only the owner can make a contact private</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>Only the super user can update or delete bug entries</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>Permission error: you may not edit this role.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>Permission Violation: partners do not have access to this report.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>Permission Violation: partners may not delete saved reports.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>Permission Violation: You cannot delete saved reports not created by yourself.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>&lt;b&gt;Test Drive does not allow access to this feature.&lt;/b&gt; If you would like more information about this feature, please contact your account manager.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>The restriction settings on your role deny you access to this item.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>This folder does not exist or you do not have permission to access this folder.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>This folder does not permit the direct addition of files</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>This order has been partially or fully processed and may not be edited by a user without permission to approve sales orders.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>User permission level could not be established</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have permissions to set a value for element {1} due to one of the following reasons: 1) The field is read-only; 2) An associated feature is disabled; 3) The field is available either when a record is created or updated, but not in both cases.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>Your role does not have permission to provision accounts.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You are not allowed to approve your own transactions.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You are not authorized to change this event’s organizer. Public events may only have their organizer changed by administrators, the event’s organizer, or delegates with edit permission to the event’s calendar. Private or busy events may only have their organizer changed by the owner.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You cannot update a system defined template.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You cannot update cases using this form.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You can only delete notes that you created.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have access to the activity history for that record.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have access to the media item you selected.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have access to this page</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have access to this template</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have permission to access this list.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have permission to access this register.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have permission to access this type of transaction.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have permission to create this type of record. Please choose a different record type.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have permission to email transactions.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have permission to perform this operation.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have permission to print (1)</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have permission to view this page.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have privileges to approve commissions.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have privileges to create commissions.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have privileges to create this transaction.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have privileges to perform that operation.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have privileges to perform this action.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have privileges to perform this operation</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have privileges to use this page.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have privileges to view this account</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You do not have privileges to view this page</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You may not create a new Liability Adjustment or edit existing Liability Adjustments.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You may not delete built-in audiences.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You may not delete built-in categories.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You may not delete built-in items.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You may not delete built-in tabs.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You must have either 'Transactions -&gt; Invoice' or 'Transactions - &gt; Cash Sale' permission to bill sales orders.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You must have either 'Transactions -&gt; Invoice' or 'Transactions - &gt; Cash Sale' permission to fulfill sales orders.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You must have 'Transactions -&gt; {1}’ permission to build work orders.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You must have 'Transactions -&gt; Fulfill Sales Orders' view permission to view sales order fulfillments.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You must have 'Transactions -&gt; Fulfill Sales Orders’ edit permission to fulfill sales orders.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>You need employee access in order to delete this record.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>(1) The (2) restrictions on your role deny you access to this record.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMISSION</td>
<td>(1) The (2) restrictions on your role prevent you from seeing this record.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMITION</td>
<td>{1} The customer restrictions on your partner role prevent you from seeing this record.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMITION</td>
<td>{1} The restrictions on your role deny you access to this record.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMITION</td>
<td>{1} The restrictions on your role do not allow you to modify this record.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMITION</td>
<td>{1} You need {2} the '{3}' permission to access this page. Please contact your account administrator.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMITION</td>
<td>{1} You need a higher level of the '{2}' permission to access this page. Please contact your account administrator.</td>
</tr>
<tr>
<td>INSUFFICIENT_PERMITION</td>
<td>{1} You need a higher permission for custom record type {2} to access this page. Please contact your account administrator.</td>
</tr>
<tr>
<td>INTEGER_REQD_FOR_QTY</td>
<td>Quantity must be an integer for numbered items.</td>
</tr>
<tr>
<td>INTL_FEDEX_ONE_PKG_ALLWD</td>
<td>International FedEx fulfillments allow only one package. If more than one package is required, please break up the shipment into multiple fulfillments of one package each.</td>
</tr>
<tr>
<td>INTL_SHIP_EXCEED_MAX_ITEM</td>
<td>The maximum number of items for FedEx International shipping has been exceeded: {1}</td>
</tr>
<tr>
<td>INVALID_ABN</td>
<td>Invalid ABN registration number {1}.</td>
</tr>
<tr>
<td>INVALID_ACCT</td>
<td>Invalid login. No such account.</td>
</tr>
<tr>
<td>INVALID_ACCT</td>
<td>Invalid account number.</td>
</tr>
<tr>
<td>INVALID_ACCT_NUM_CSTM_FIELD</td>
<td>The account number custom field does not exist!! Consult billing cell.</td>
</tr>
<tr>
<td>INVALID_ACCT_PRD</td>
<td>You can not create an accounting period that is not a year or does not belong to a year.</td>
</tr>
<tr>
<td>INVALID_ACCT_TYP</td>
<td>Invalid account type [{1}].</td>
</tr>
<tr>
<td>INVALID_ACCT_TYP</td>
<td>There is no account of type: {1}</td>
</tr>
<tr>
<td>INVALID_ACCT_TYP</td>
<td>The account and its parent have different account type.</td>
</tr>
<tr>
<td>INVALID_ACCT_TYP</td>
<td>You cannot change an account to or from A/R or A/P</td>
</tr>
<tr>
<td>INVALID_ACTION</td>
<td>You have attempted an unsupported action.</td>
</tr>
<tr>
<td>INVALID_ADDRESS_OR_SHIPPER_NO</td>
<td>An error has occurred. Please ensure that the address information and shipper number are correct, then resubmit the form.</td>
</tr>
<tr>
<td>INVALID_ADJUSTMENT_ACCT</td>
<td>The account you selected in Adjustment Account is the same as the asset account for one of the items you are adjusting. Please go back and change the account. Normally, the adjustment account would be an expense account.</td>
</tr>
<tr>
<td>INVALID_AES_FTSR_EXEMPTN_NUM</td>
<td>The AES/FTSR Exemption Number is invalid.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INVALID_ALLOCTN_METHOD</td>
<td>You have attempted to allocate landed costs to a transaction using an allocation method that results in no allocation for any lines in the transaction. The allocation method you chose is {1}. To correct this problem, go back to the transaction and choose a different allocation method, or modify the items/lines on the transaction so that there will be some cost allocated to the lines.</td>
</tr>
<tr>
<td>INVALID_AMT</td>
<td>Amount applied greater than total payments and credits</td>
</tr>
<tr>
<td>INVALID_APP_ID</td>
<td>Invalid application id: (1)</td>
</tr>
<tr>
<td>INVALID_ASSIGN_STATUS_COMBO</td>
<td>Invalid assignee/status combination((1)/(2))</td>
</tr>
<tr>
<td>INVALID_ASSIGN_STATUS_COMBO</td>
<td>Invalid assignee/status combination (assignee (1), status (2), issue #(3)). No default owner for issue role?</td>
</tr>
<tr>
<td>INVALID_AUTH_CODE</td>
<td>You have entered an invalid authorization code for this campaign email address. Please check the authorization code in the email message, and enter it again.</td>
</tr>
<tr>
<td>INVALID_AUTOAPPLY_VALUE</td>
<td>Ambiguous data: &lt;autoApply&gt; has been selected and lines have been selected in the &lt;applyList&gt; element.</td>
</tr>
<tr>
<td>INVALID_BALANCE_RANGE</td>
<td>Your balance is not within the allowed range.</td>
</tr>
<tr>
<td>INVALID_BILLING_SCHDUL</td>
<td>The billing schedule definition is incompatible with this transaction. Please modify the current billing schedule or select a different one.</td>
</tr>
<tr>
<td>INVALID_BILLING_SCHDUL_ENTRY</td>
<td>You cannot create a billing schedule with two entries on the same date. Please go back and edit the billing schedule or start date.</td>
</tr>
<tr>
<td>INVALID_BIN_NUM</td>
<td>Bin numbers may not contain the '{1}' character</td>
</tr>
<tr>
<td>INVALID_BOM_QTY</td>
<td>Inventory/Assembly quantities cannot be negative</td>
</tr>
<tr>
<td>INVALID_BOOLEAN_VALUE</td>
<td>Checkbox / boolean data must be either 'T' or 'F'</td>
</tr>
<tr>
<td>INVALID_BUG_NUM</td>
<td>Bug number specified was incorrect. &quot;{(1)}&quot; isn't a number.</td>
</tr>
<tr>
<td>INVALID_CAMPAIGN_CHANNEL</td>
<td>You cannot use this channel to setup this event</td>
</tr>
<tr>
<td>INVALID_CAMPAIGN_GROUP_SIZE</td>
<td>While in (1), you can only send (2) emails per campaign event. Please modify one or more of your target groups to contain (2) members or less. All campaign emails will be sent to your (1) login email address.</td>
</tr>
<tr>
<td>INVALID_CAMPAIGN_STATUS</td>
<td>You cannot set the status of this campaign event back to 'In Progress' because it already has some activity.</td>
</tr>
<tr>
<td>INVALID_CASE_FORM</td>
<td>You cannot create cases using this form.</td>
</tr>
<tr>
<td>INVALID_CATGRY_TAX_AGENCY_REQ</td>
<td>A Vendor must be created in a category with the Tax Agency checkbox checked.</td>
</tr>
<tr>
<td>INVALID_CC_EMAIL_ADDRESS</td>
<td>The email address to CC store orders to is invalid. Please go back and correct it.</td>
</tr>
<tr>
<td>INVALID_CC_NUM</td>
<td>Credit card numbers must contain between 13 and 20 digits.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INVALID_CC_NUM</td>
<td>Credit card number is not valid. Please check that all digits were entered correctly.</td>
</tr>
<tr>
<td>INVALID_CC_NUM</td>
<td>Credit card number must contain only digits.</td>
</tr>
<tr>
<td>INVALID_CHARS_IN_EMAIL</td>
<td>Email address contains invalid characters.</td>
</tr>
<tr>
<td>INVALID_CHARS_IN_NAME</td>
<td>The From Name field cannot contain apostrophes, quotation marks, commas, or greater than or less than signs.</td>
</tr>
<tr>
<td>INVALID_CHARS_IN_NAME</td>
<td>You cannot use the colon <code>:</code> character in the topic name - please remove it.</td>
</tr>
<tr>
<td>INVALID_CHARS_IN_PARAM_FIELD</td>
<td>The Additional Parameters field can not contain any of the following characters: &quot;?&lt;&gt;@#$%^*()+,.:;'&quot;. Please remove them and try again</td>
</tr>
<tr>
<td>INVALID_CHARS_IN_URL</td>
<td>Spaces are not allowed in the {1}url.&lt;b&gt;Examples of a valid {1}url are:&lt;b&gt;<a href="http://www.mydomain.com/image.gif">http://www.mydomain.com/image.gif</a>&lt;/b&gt;  or  &lt;b&gt;<a href="https://one.two.org/username/test.jpg">https://one.two.org/username/test.jpg</a>&lt;/b&gt;</td>
</tr>
<tr>
<td>INVALID_CHARS_IN_URL</td>
<td>The URL component you have chosen contains a space or one of the following prohibited character: &quot;?&lt;&gt;@#$%^&amp;*()+=,.:;'&quot;. Please remove them and try again</td>
</tr>
<tr>
<td>INVALID_COLUMN_NAME</td>
<td>Invalid column name in get_invitem_col_sum_all_locs: {1} [ {2} ]</td>
</tr>
<tr>
<td>INVALID_COSTING_METHOD</td>
<td>SERIAL and LOT are the only costing methods that may be passed as parameters to this page.</td>
</tr>
<tr>
<td>INVALID_CSTM_FIELD_DATA_TYP</td>
<td>The customfield [{1}] reference object does not match its data type.</td>
</tr>
<tr>
<td>INVALID_CSTM_FIELD_RCRD_TYP</td>
<td>Invalid custom field record type</td>
</tr>
<tr>
<td>INVALID_CSTM_FIELD_REF</td>
<td>The specified custom field reference {1} is invalid.</td>
</tr>
<tr>
<td>INVALID_CSTM_FORM</td>
<td>{1} is an invalid custom form</td>
</tr>
<tr>
<td>INVALID_CSTM_RCRD_KEY</td>
<td>Invalid custom record key [{1}].</td>
</tr>
<tr>
<td>INVALID_CSTM_RCRD_QUERY</td>
<td>Invalid custom record object in query.</td>
</tr>
<tr>
<td>INVALID_CSTM_RCRD_TYPE_KEY</td>
<td>Invalid custom record type</td>
</tr>
<tr>
<td>INVALID_CSTM_RCRD_TYPE_KEY</td>
<td>Invalid custom record type key.</td>
</tr>
<tr>
<td>INVALID_CSTM_RCRD_TYPE_KEY</td>
<td>(1) refers to a custom list. To get the contents of this list, use the 'get' or 'getAll' operation with a RecordRef of type 'customList'</td>
</tr>
<tr>
<td>INVALID_CSTM_RCRD_TYP_KEY</td>
<td>Invalid custom record type key in query.</td>
</tr>
<tr>
<td>INVALID_CUSTOMER_RCRD</td>
<td>This customer record {1} is not valid. Please create the customer first.</td>
</tr>
<tr>
<td>INVALID_DATA</td>
<td>Invalid data combination, can not set {1} to {2} and {3} to {4}</td>
</tr>
<tr>
<td>INVALID_DATE</td>
<td>The date &lt; {1} &gt; is invalid. You must specify a date after &lt; {2} &gt;.</td>
</tr>
<tr>
<td>INVALID_DATE_FORMAT</td>
<td>Date field not in your preferred date format.</td>
</tr>
<tr>
<td>INVALID_DATE_RANGE</td>
<td>Invalid time range. The {1} &quot;{2}&quot; start time ({3}) must be earlier than its end time ({4}).</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INVALID_DATE_RANGE</td>
<td>Invalid Date Range - the To Date value must be on or after the From Date value.</td>
</tr>
<tr>
<td>INVALID_DATE_RANGE</td>
<td>The date range you specified does not fall inside that of the parent period.</td>
</tr>
<tr>
<td>INVALID DEAL_RANGE</td>
<td>Invalid Deal Range - low must be less than projected and high must be greater than projected.</td>
</tr>
<tr>
<td>INVALID DEAL_RANGE</td>
<td>Invalid Deal Range - low must be less than projected and high must be greater than projected.</td>
</tr>
<tr>
<td>INVALID_DELETE_REF</td>
<td>Either RecordRef or CustomRecordRef should be used for 'delete' operation.</td>
</tr>
<tr>
<td>INVALID_DESTINATION_FLDR</td>
<td>The destination folder is the same as the current folder.</td>
</tr>
<tr>
<td>INVALID_DESTNTN_COUNTRY</td>
<td>The destination Country is invalid or has not been set.</td>
</tr>
<tr>
<td>INVALID_DESTNTN_POST_CODE</td>
<td>The destination Postal Code is invalid or has not been set.</td>
</tr>
<tr>
<td>INVALID_DESTNTN_STATE</td>
<td>The destination State is invalid or has not been set.</td>
</tr>
<tr>
<td>INVALID_DETACH_RECORD_TYP</td>
<td>Missing or Invalid RecordType for DetachFrom.</td>
</tr>
<tr>
<td>INVALID_DETACH_RECORD_TYP</td>
<td>Detaching of record type {1} from {2} is not supported.</td>
</tr>
<tr>
<td>INVALID_DOMAIN_KEY</td>
<td>The private domain key is invalid, please enter a valid private domain key.</td>
</tr>
<tr>
<td>INVALID_DOMAIN_NAME</td>
<td>Invalid domain name {1}, please enter a valid domain name.</td>
</tr>
<tr>
<td>INVALID DUP ISSUE_REF</td>
<td>Cannot set this issue to be a duplicate of itself or of an issue that is a duplicate of this issue.</td>
</tr>
<tr>
<td>INVALID_EMAIL</td>
<td>Email address is not valid.</td>
</tr>
<tr>
<td>INVALID_EMAIL</td>
<td>Your email or code is invalid. Please try again.</td>
</tr>
<tr>
<td>INVALID_EMAIL</td>
<td>You have entered an invalid email address. Please try again.</td>
</tr>
<tr>
<td>INVALID_EMAIL_ADDR</td>
<td>Some of the email addresses you have entered are invalid: {1:list of invalid email addresses}</td>
</tr>
<tr>
<td>INVALID_EMAIL_ADDR</td>
<td>The email address for the web store is invalid. Please go back and correct it.</td>
</tr>
<tr>
<td>INVALID_END_DATE</td>
<td>You entered an end date ([1]) that is before the start date ([2])</td>
</tr>
<tr>
<td>INVALID_END_DATE</td>
<td>([1] [2]) recurrence end date is invalid</td>
</tr>
<tr>
<td>INVALID_END_TIME</td>
<td>invalid 'end' time</td>
</tr>
<tr>
<td>INVALID ENTITY_INTERNALID</td>
<td>Attempt to insert entity with nkey -1 or 0</td>
</tr>
<tr>
<td>INVALID ENTITY_STATUS</td>
<td>You entered an invalid entity status.</td>
</tr>
<tr>
<td>INVALID EVENT_TIME</td>
<td>You cannot make the time that close to the start or end of the day, because it shifts the event across a day boundary.</td>
</tr>
<tr>
<td>INVALID_EXPRESSION</td>
<td>ERROR: Invalid Expression</td>
</tr>
<tr>
<td>INVALID FAX NUM</td>
<td>The Fax Number is invalid.</td>
</tr>
<tr>
<td>INVALID FAX PHONE_FORMAT</td>
<td>Invalid FaxPhoneNumber. The format of FaxPhoneNumber must contain area code plus seven digit number.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>INVALID_FIELD_FOR_RCRD_TYP</td>
<td>Record type (1) does not support field (2)</td>
</tr>
<tr>
<td>INVALID_FIELD_NAME_FOR_NULL</td>
<td>The specified name ([1]) must exactly match an existing field name.</td>
</tr>
<tr>
<td>INVALID_FILE</td>
<td>Verify that you have a valid file to upload.</td>
</tr>
<tr>
<td>INVALID_FILE_TYP</td>
<td>Invalid file type. File is not a compressed/zip file.</td>
</tr>
<tr>
<td>INVALID_FILE_TYP</td>
<td>Invalid file type. File is not a compressed zip file.</td>
</tr>
<tr>
<td>INVALID_FILE_TYP</td>
<td>The media file type you uploaded was not recognized. Please try again.</td>
</tr>
<tr>
<td>INVALID_FLDR_SIZE</td>
<td>Error in update_folder_size</td>
</tr>
<tr>
<td>INVALID_FLD_RANGE</td>
<td>Value outside of valid min/max range for this field</td>
</tr>
<tr>
<td>INVALID_FLD_VALUE</td>
<td>You have entered an Invalid Field Value (1) for the following field: (2)</td>
</tr>
<tr>
<td>INVALID_FORMAT_IN_PARAM_FIELD</td>
<td>The Additional Parameters field is not formatted correctly. Please reformat and try again</td>
</tr>
<tr>
<td>INVALID_FORMULA</td>
<td>Your formula contains a reference to an encrypted field. This is not allowed.</td>
</tr>
<tr>
<td>INVALID_FORMULA</td>
<td>Your formula could result in a divide by zero error. Please go back, correct the formula and resubmit.</td>
</tr>
<tr>
<td>INVALID_FORMULA_FIELD</td>
<td>Your formula has an unrecognized field in it. Please go back and correct the formula and resubmit.</td>
</tr>
<tr>
<td>INVALID_FROM_DATE</td>
<td>invalid 'from' date</td>
</tr>
<tr>
<td>INVALID_FROM_TIME</td>
<td>invalid 'from' time</td>
</tr>
<tr>
<td>INVALID_FULFILMNT_ITEM</td>
<td>You have an invalid item (1) in the fulfillment request.</td>
</tr>
<tr>
<td>INVALID_FX_RATE</td>
<td>Exchange Rate must be 1 for vendors in your currency.</td>
</tr>
<tr>
<td>INVALID_GET_REF</td>
<td>Either RecordRef or CustomRecordRef should be used for 'get' operation.</td>
</tr>
<tr>
<td>INVALID_GIFT_CERT</td>
<td>Invalid gift certificate</td>
</tr>
<tr>
<td>INVALID_GIFT_CERT_AMT</td>
<td>The remaining amount on a gift certificate can not be negative</td>
</tr>
<tr>
<td>INVALID_GIFT_CERT_CODE</td>
<td>Gift certificate code must contain only letters and digits.</td>
</tr>
<tr>
<td>INVALID_GROUP_TYP</td>
<td>This type of group cannot be defined based on another group of the same type.</td>
</tr>
<tr>
<td>INVALID_GROUP_TYP</td>
<td>You cannot define this group type using this search.</td>
</tr>
<tr>
<td>INVALID_GROUP_TYPE</td>
<td>The group type (1) is invalid.</td>
</tr>
<tr>
<td>INVALID_GST_PST_AGENCIES</td>
<td>The GST or PST agencies are not valid. Please review your company preferences</td>
</tr>
<tr>
<td>INVALID_ID</td>
<td>Identifiers can contain only digits, alphabetic characters, or &quot;_&quot; with no spaces</td>
</tr>
<tr>
<td>INVALID_ID</td>
<td>You have provided an invalid script id or internal id: [1]</td>
</tr>
<tr>
<td>INVALID_ID</td>
<td>The externalId attribute is not supported for (1)</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INVALID_INITIALIZE_ARG</td>
<td>The reference type {1} and initialize type {2} are not matched.</td>
</tr>
<tr>
<td>INVALID_INITIALIZE_ARG</td>
<td>InitializeRef should be used for 'initialize' operation.</td>
</tr>
<tr>
<td>INVALID_INITIALIZE_AUXREF</td>
<td>Invalid initialize operation argument 'auxReference'.</td>
</tr>
<tr>
<td>INVALID_INITIALIZE_REF</td>
<td>You cannot initialize {1}: invalid reference {2}.</td>
</tr>
<tr>
<td>INVALID_INITIALIZE_REF</td>
<td>Can not initialize customerPayment: invalid customer reference {1}.</td>
</tr>
<tr>
<td>INVALID_INITIALIZE_REF</td>
<td>Can not initialize customerPayment: invalid invoice reference {1}.</td>
</tr>
<tr>
<td>INVALID_INITIALIZE_REF</td>
<td>You have an invalid sales order {1} or the order is already billed.</td>
</tr>
<tr>
<td>INVALID_INITIALIZE_REF</td>
<td>You have an invalid sales order {1} or the order is already closed.</td>
</tr>
<tr>
<td>INVALID_INSURED_VALUE</td>
<td>The Insured Value cannot exceed the total sum of the items being shipped.</td>
</tr>
<tr>
<td>INVALID_INVENTORY_NUM</td>
<td>Invalid set of inventory numbers: values must be separated by commas, spaces, tabs, or line feeds.</td>
</tr>
<tr>
<td>INVALID_INV_DATE</td>
<td>Invoice date on billing schedule may not be after {1}.</td>
</tr>
<tr>
<td>INVALID_IP_ADDRESS_RULE</td>
<td>The following IP Address rule is not valid: {1}.</td>
</tr>
<tr>
<td>INVALID_ISSUE_PRIORITY</td>
<td>Severity 1 issues must have priority 1.</td>
</tr>
<tr>
<td>INVALID_ITEM_OPTION</td>
<td>Invalid item option {1} for item {2}.</td>
</tr>
<tr>
<td>INVALID_ITEM_OPTIONS</td>
<td>The options for item '{1}' are no longer available. Please change your order and try again.</td>
</tr>
<tr>
<td>INVALID_ITEM_SUBTYP</td>
<td>Invalid item subtype [{1}] for item [{2}].</td>
</tr>
<tr>
<td>INVALID_ITEM_TYP</td>
<td>The item [{1}] does not have a valid item type.</td>
</tr>
<tr>
<td>INVALID_ITEM_WEIGHT</td>
<td>The total item weight must be &gt; 0.0</td>
</tr>
<tr>
<td>INVALID_JOB_ID</td>
<td>You have specified an invalid Job Id</td>
</tr>
<tr>
<td>INVALID_KEY_OR_REF</td>
<td>The specified key is invalid.</td>
</tr>
<tr>
<td>INVALID_KEY_OR_REF</td>
<td>Invalid {1} reference key {2}.</td>
</tr>
<tr>
<td>INVALID_KEY_OR_REF</td>
<td>Invalid {1} reference key {2} for {3} {4}.</td>
</tr>
<tr>
<td>INVALID_LINK_SUM</td>
<td>Links sum to more than applied transaction amount</td>
</tr>
<tr>
<td>INVALID_LINK_SUM</td>
<td>Links sum to more than original transaction amount</td>
</tr>
<tr>
<td>INVALID_LIST_ID</td>
<td>You must specify a valid line ID. Please set {1}.</td>
</tr>
<tr>
<td>INVALID_LIST_KEY</td>
<td>Could not perform operation &quot;{1}&quot; on an invalid line {2}.</td>
</tr>
<tr>
<td>INVALID_LIST_KEY</td>
<td>Could not perform operation 'add' on an existing line {1}.</td>
</tr>
<tr>
<td>INVALID_LOGIN</td>
<td>Invalid login. Online Form access is disabled.</td>
</tr>
<tr>
<td>INVALID_LOGIN</td>
<td>Invalid login. Supplier access is disabled.</td>
</tr>
<tr>
<td>INVALID_LOGIN_ATTEMPT</td>
<td>Invalid login attempt.</td>
</tr>
<tr>
<td>INVALID_LOGIN_CREDENTIALS</td>
<td>A problem occurred verifying the presented email address, password, roleName or account number, please verify these pieces of information and try again.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INVALID_LOGIN_CREDENTIALS</td>
<td>You have entered an invalid email address or account number. Please try again.</td>
</tr>
<tr>
<td>INVALID_LOGIN_CREDENTIALS</td>
<td>You have entered an invalid email address or password. Please try again.</td>
</tr>
<tr>
<td>INVALID_LOGIN_CREDENTIALS</td>
<td>You have entered an invalid login password. Please try again.</td>
</tr>
<tr>
<td>INVALID_LOGIN_CREDENTIALS</td>
<td>You have entered an invalid password. Please try again.</td>
</tr>
<tr>
<td>INVALID_LOGIN_IP</td>
<td>Invalid login. IP Address does not match any of the IP Address rules specified for this entity.</td>
</tr>
<tr>
<td>INVALID_LOT_NUM_FORMAT</td>
<td>Lot numbers must be entered using this format: LOT#(Quantity). For example, to enter a quantity of 100 items as Lot number ABC1234, enter &quot;ABC1234(100)&quot; in the Lot Numbers field.</td>
</tr>
<tr>
<td>INVALID_MARKUP_DISCOUNT</td>
<td>Markup/Discount % must be between -999% and 999%</td>
</tr>
<tr>
<td>INVALID_MEMRZD_TRANS</td>
<td>A memorized transaction may not contain any serial or lot numbers. Go back, remove the numbers, and try to re-Memorize the transaction. Posting transactions such as Bills or Cash Sales may not use serial or lot numbered items. Non-Posting transactions such as Purchase Orders or Sales Orders may use serial or lot numbered items but may not contain serial or lot numbers.</td>
</tr>
<tr>
<td>INVALID_NUMBER</td>
<td>Invalid Decimal Number</td>
</tr>
<tr>
<td>INVALID_NUMBER</td>
<td>Invalid Integer</td>
</tr>
<tr>
<td>INVALID_NUMBER</td>
<td>Invalid integer {1}</td>
</tr>
<tr>
<td>INVALID_NUMBER</td>
<td>Invalid number {1}</td>
</tr>
<tr>
<td>INVALID_NUMBER</td>
<td>You entered &quot;{(1)}&quot; into a field where a numeric value was expected. Please go back and change this value to a number.</td>
</tr>
<tr>
<td>INVALID_NUMBER</td>
<td>You entered an invalid number: &lt;br&gt;Go &lt;a href=&quot;javascript:history.go(-1);&quot;&gt;back&lt;/a&gt;, change this value and resubmit.</td>
</tr>
<tr>
<td>INVALID_OBJ</td>
<td>There are no objects of this type</td>
</tr>
<tr>
<td>INVALID_ONLINE_FORM</td>
<td>This online form is inactive or not available online.</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
<td>That operation is not supported for this record type: {1}</td>
</tr>
<tr>
<td>INVALID_ORD_STATUS</td>
<td>This order has been partially or fully processed and may not be reset to 'Pending Approval'.</td>
</tr>
<tr>
<td>INVALID_ORIGIN_COUNTRY</td>
<td>The origin Country is invalid or has not been set.</td>
</tr>
<tr>
<td>INVALID_ORIGIN_POSTCODE</td>
<td>The origin Postal Code is invalid or has not been set.</td>
</tr>
<tr>
<td>INVALID_ORIGIN_STATE</td>
<td>The origin State is invalid or has not been set.</td>
</tr>
<tr>
<td>INVALID_PAGER_NUM</td>
<td>The Pager Number is invalid.</td>
</tr>
<tr>
<td>INVALID_PAGE_INDEX</td>
<td>Job {1} does not have a page {2}</td>
</tr>
<tr>
<td>INVALID_PARENT</td>
<td>An account cannot be its own parent</td>
</tr>
<tr>
<td>INVALID_PARTNER_ID</td>
<td>Invalid partner id: {1}</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INVALID_PHONE_FAX_PAGER_NUM</td>
<td>The Phone, Fax, or Pager Number is invalid.</td>
</tr>
<tr>
<td>INVALID_PHONE_NUM</td>
<td>The Phone Number is invalid.</td>
</tr>
<tr>
<td>INVALID_PICKUP_POSTAL_CODE</td>
<td>An error has occurred. Pickup Postal Code (1) is not the postal code associated with Shipper Number (2).</td>
</tr>
<tr>
<td>INVALID_POST</td>
<td>Invalid Post</td>
</tr>
<tr>
<td>INVALID_PROBABILITY_RANGE</td>
<td>Probability must be between 0 and 100.</td>
</tr>
<tr>
<td>INVALID_PST_TAX_VALUE</td>
<td>PST tax value is not a valid number: [1]</td>
</tr>
<tr>
<td>INVALID_PSWD</td>
<td>Invalid Password. The password must be between 6 and 10 character with at least one numeric and one alphabetic character.</td>
</tr>
<tr>
<td>INVALID_PSWD</td>
<td>Password must be at least 6 characters long.</td>
</tr>
<tr>
<td>INVALID_PSWD</td>
<td>Password must be at least 6 characters long and contain at least one number or special character.</td>
</tr>
<tr>
<td>INVALID_PSWD</td>
<td>Password must contain at least one letter (A-Z).</td>
</tr>
<tr>
<td>INVALID_PSWD</td>
<td>Password must contain at least one number or special character.</td>
</tr>
<tr>
<td>INVALID_PSWD</td>
<td>The current password you supplied is incorrect.</td>
</tr>
<tr>
<td>INVALID_PSWD</td>
<td>Your new password must be at least [1] characters, contain at least one non-letter, and be substantially different from the current password.</td>
</tr>
<tr>
<td>INVALID_PSWD</td>
<td>Your new password must be at least 6 characters, contain at least one non-letter, and be substantially different from the current password.</td>
</tr>
<tr>
<td>INVALID_PSWD</td>
<td>Your password cannot be the same as your login. Please choose a new password.</td>
</tr>
<tr>
<td>INVALID_PSWD</td>
<td>Your password must be at least 6 characters</td>
</tr>
<tr>
<td>INVALID_PSWD</td>
<td>You've used that password before. Please choose a new password.</td>
</tr>
<tr>
<td>INVALID_PSWD_HINT</td>
<td>Your hint is too similar to your password. Please choose something less obvious.</td>
</tr>
<tr>
<td>INVALID_PSWD_ILLEGAL_CHAR</td>
<td>Password contains an illegal character.</td>
</tr>
<tr>
<td>INVALID_PURCHASE_TAX_CODE</td>
<td>Purchase tax code not defined properly for item</td>
</tr>
<tr>
<td>INVALID_QUANTITY</td>
<td>Serial and lot number quantities must be integers</td>
</tr>
<tr>
<td>INVALID_QUANTITY</td>
<td>Serial and lot number quantities must be positive.</td>
</tr>
<tr>
<td>INVALID_RCRD_ID</td>
<td>Invalid id (1) to create a record.</td>
</tr>
<tr>
<td>INVALID_RCRD_OBJ</td>
<td>You do not have a valid record object.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INVALID_RCRD_REF</td>
<td>Invalid RecordRef internalId {1} for field {2}</td>
</tr>
<tr>
<td>INVALID_RCRD_REF</td>
<td>Invalid record reference.</td>
</tr>
<tr>
<td>INVALID_RCRD_REF</td>
<td>Invalid record reference</td>
</tr>
<tr>
<td>INVALID_RCRD_TRANSFRM</td>
<td>You have entered an invalid default value for this record transformation operation.</td>
</tr>
<tr>
<td>INVALID_RCRD_TRANSFRM</td>
<td>That type of record transformation is not allowed. Please see the documentation for a list of supported transformation types.</td>
</tr>
<tr>
<td>INVALID_RCRD_TRANSFRM</td>
<td>That is not a valid record transformation.</td>
</tr>
<tr>
<td>INVALID_RCRD_TYPE</td>
<td>Invalid Record Type</td>
</tr>
<tr>
<td>INVALID_RCRD_TYPE</td>
<td>{1}: type argument {2} is not a valid record or is not available in your account. Please see the documentation for a list of supported record types.</td>
</tr>
<tr>
<td>INVALID_RCRD_TYPE</td>
<td>The record type [{1}] is invalid.</td>
</tr>
<tr>
<td>INVALID_RCRD_TYPE</td>
<td>The record type is invalid.</td>
</tr>
<tr>
<td>INVALID_RECIPIENT</td>
<td>Recipient internal id does not match an existing entity.</td>
</tr>
<tr>
<td>INVALID_RECR_REF</td>
<td>Could not update {1} to {2} because referenced record does not exist.</td>
</tr>
<tr>
<td>INVALID_RECUR_DATE_RANGE</td>
<td>This event recurrence is invalid because its duration is either negative or longer than one day. {1}</td>
</tr>
<tr>
<td>INVALID_RECUR_DATE_RANGE</td>
<td>This event recurrence is invalid because its end-by date is before its start date. {1}</td>
</tr>
<tr>
<td>INVALID_RECUR_DATE_RANGE</td>
<td>This event recurrence is invalid because its end time and duration do not match. {1}</td>
</tr>
<tr>
<td>INVALID_RECUR_DATE_RANGE</td>
<td>This event recurrence is invalid because its end time is more than one day after its start time. {1}</td>
</tr>
<tr>
<td>INVALID_RECUR_DATE_RANGE</td>
<td>This event recurrence is invalid because its end time is not after its start time. {1}</td>
</tr>
<tr>
<td>INVALID_RECUR_DATE_RANGE</td>
<td>This event recurrence is invalid because its start time or end time/duration is empty. {1}</td>
</tr>
<tr>
<td>INVALID_RECUR_DATE_RANGE</td>
<td>This event recurrence is invalid because its times are not in order. {1}</td>
</tr>
<tr>
<td>INVALID_RECUR_DESC_REQD</td>
<td>This event recurrence is invalid because it has no description. {1}</td>
</tr>
<tr>
<td>INVALID_RECUR_DOW</td>
<td>This event recurrence has an invalid day-of-week field. {1}</td>
</tr>
<tr>
<td>INVALID_RECUR_DOWIM</td>
<td>This event recurrence has an invalid day-of-week-in-month value. {1}</td>
</tr>
<tr>
<td>INVALID_RECUR_DOWMASK</td>
<td>This event recurrence is invalid because its day-of-month mask is not 7 characters long. {1}</td>
</tr>
<tr>
<td>INVALID_RECUR_FREQUENCY</td>
<td>This event recurrence has an invalid frequency. {1}</td>
</tr>
<tr>
<td>INVALID_RECUR_PATTERN</td>
<td>This event does not have a valid recurrence pattern.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INVALID_RECUR_PATTERN</td>
<td>This event recurrence is invalid because it is not a monthly or yearly event and it has day-of-week and day-of-week-in-month field values. [1]</td>
</tr>
<tr>
<td>INVALID_RECUR_PATTERN</td>
<td>This event recurrence is invalid because it only has one recurrence time and it must have either none or at least two. [1]</td>
</tr>
<tr>
<td>INVALID_RECUR_PATTERN</td>
<td>This event recurrence is invalid because one of its times is out of the range 0 to 86399. [1]</td>
</tr>
<tr>
<td>INVALID_RECUR_PATTERN</td>
<td>This event recurrence is invalid because only one of the day-of-week and day-of-week-in-month fields is set. Both must be set or both must be unset. [1]</td>
</tr>
<tr>
<td>INVALID_RECUR_PATTERN</td>
<td>This event recurrence is invalid either because it is not weekly and it has a day-of-week mask, or it is weekly and it has no day-of-week mask. [1]</td>
</tr>
<tr>
<td>INVALID_RECUR_PATTERN</td>
<td>This single day event is invalid since it contains a recurrence pattern.</td>
</tr>
<tr>
<td>INVALID_RECUR_PERIOD</td>
<td>This event recurrence has an invalid period. [1]</td>
</tr>
<tr>
<td>INVALID_RECUR_TIME_ZONE_REQD</td>
<td>This event recurrence is invalid because it has no time zone. [1]</td>
</tr>
<tr>
<td>INVALID_REFFERER_EMAIL</td>
<td>The referer email address you have entered is not valid. Please try again.</td>
</tr>
<tr>
<td>INVALID_REF_CANT_INITIALIZE</td>
<td>Cannot initialize customerRefund: invalid creditMemo reference [1].</td>
</tr>
<tr>
<td>INVALID_REF_CANT_INITIALIZE</td>
<td>Cannot initialize customerRefund: invalid customer reference [1].</td>
</tr>
<tr>
<td>INVALID_REF_CANT_INITIALIZE</td>
<td>You can not initialize [1] by referencing [2].</td>
</tr>
<tr>
<td>INVALID_REF_KEY</td>
<td>Invalid externalId [1].</td>
</tr>
<tr>
<td>INVALID_REF_KEY</td>
<td>Invalid reference key [1].</td>
</tr>
<tr>
<td>INVALID_RESULT_SUMMARY_FUNC</td>
<td>The result field [1] cannot be grouped. Please edit the search and omit this field or use a different summary function.</td>
</tr>
<tr>
<td>INVALID_REV_REC_DATE_RANGE</td>
<td>Rev rec end date can not be before rev rec start date.</td>
</tr>
<tr>
<td>INVALID_ROLE</td>
<td>The specified role is invalid.</td>
</tr>
<tr>
<td>INVALID_ROLE</td>
<td>Your role does not give you permission to view this page.</td>
</tr>
<tr>
<td>INVALID_ROLE_FOR_EVENT</td>
<td>You seem to have been invited to this [1] in a different role. Please change your role to view the [1].</td>
</tr>
<tr>
<td>INVALID_RQST_CONTACTS_EXIST</td>
<td>It has associated primary contacts.</td>
</tr>
<tr>
<td>INVALID_RQST_PARENT_REQD</td>
<td>It has associated contact records that would be left with no parent company.</td>
</tr>
<tr>
<td>INVALID_RQST_SBCUST_JOBS_EXIST</td>
<td>It has associated sub-customers or jobs.</td>
</tr>
<tr>
<td>INVALID_SAVEDSEARCH</td>
<td>A saved search with the internal ID [1] does not exist.</td>
</tr>
<tr>
<td>INVALID_SAVEDSEARCH</td>
<td>We cannot return search columns for summary saved search [1].</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INVALID_SCHEDUL_FORMAT</td>
<td>To create a valid schedule, please enter the bracket values in ascending orders without gaps.</td>
</tr>
<tr>
<td>INVALID_SCRIPT_ID</td>
<td>A saved search with the script ID {1} does not exist.</td>
</tr>
<tr>
<td>INVALID_SEARCH</td>
<td>That search or mass update does not exist.</td>
</tr>
<tr>
<td>INVALID_SEARCH_CRITERIA</td>
<td>Can’t search transactions: invalid cross reference key</td>
</tr>
<tr>
<td>INVALID_SEARCH_CRITERIA</td>
<td>Global Search supports at most three keywords and requires at least one. Keywords are composed of only letters, digits, and dashes.</td>
</tr>
<tr>
<td>INVALID_SEARCH_FIELD_KEY</td>
<td>search field keys are not consistent({1}/ {2})</td>
</tr>
<tr>
<td>INVALID_SEARCH_FIELD_NAME</td>
<td>search field names are not consistent({1}/ {2})</td>
</tr>
<tr>
<td>INVALID_SEARCH_FIELD_OBJ</td>
<td>{1} is not a valid search custom field</td>
</tr>
<tr>
<td>INVALID_SEARCH_FIELD_OBJ</td>
<td>{1} must be used to search custom field {2}</td>
</tr>
<tr>
<td>INVALID_SEARCH_FIELD_OBJ</td>
<td>Server application error: invalid search customfield object.</td>
</tr>
<tr>
<td>INVALID_SEARCH_FIELD_OBJ</td>
<td>Invalid search field object: {1}</td>
</tr>
<tr>
<td>INVALID_SEARCH_JOIN_ID</td>
<td>Invalid Search More</td>
</tr>
<tr>
<td>INVALID_SEARCH_MORE</td>
<td>Invalid searchMore operation. Please make sure that you have had a successful search operation before you can perform any searchMore operation.</td>
</tr>
<tr>
<td>INVALID_SEARCH_OPERATOR</td>
<td>You need to provide a valid search field operator.</td>
</tr>
<tr>
<td>INVALID_SEARCH_OPERATOR</td>
<td>You can not use this operator '{1}' for internalId search.</td>
</tr>
<tr>
<td>INVALID_SEARCH_PAGE_INDEX</td>
<td>Invalid search page index.</td>
</tr>
<tr>
<td>INVALID_SEARCH_PAGE_SIZE</td>
<td>Invalid search page size.</td>
</tr>
<tr>
<td>INVALID_SEARCH_PREF</td>
<td>You cannot set returnSearchColumns to false while you specify search columns.</td>
</tr>
<tr>
<td>INVALID_SEARCH_PREF</td>
<td>You cannot set returnSearchColumns to true without specifying search columns or referencing a saved search.</td>
</tr>
<tr>
<td>INVALID_SEARCH_SELECT_OBJ</td>
<td>Invalid search select field object: {1}</td>
</tr>
<tr>
<td>INVALID_SEARCH_VALUE</td>
<td>You need to provide a search value.</td>
</tr>
<tr>
<td>INVALID_SEARCH_VALUE</td>
<td>You need to provide search values.</td>
</tr>
<tr>
<td>INVALID_SECONDARY_EMAIL</td>
<td>Invalid secondary email address. The email address must be in a valid format.</td>
</tr>
<tr>
<td>INVALID_SECPAY_CREDENTIALS</td>
<td>The username or password used to process the transaction with SECPay was not valid. Please make sure you have entered the correct username, password, and remote password in your SECPay account setup.</td>
</tr>
<tr>
<td>INVALID_SERIAL_NUM</td>
<td>No items match the entered serial number</td>
</tr>
<tr>
<td>INVALID_SERIAL_OR_LOT_NUMBER</td>
<td>Serial and lot numbers may not contain the '{1}' character.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INVALID_SHIPPER_STATE</td>
<td>The Shipper State/Province Code is missing or invalid. Please enter the 2 to 5 character abbreviation for the state or province of the address that contains it.</td>
</tr>
<tr>
<td>INVALID_SHIP_FROM_STATE</td>
<td>The Ship From State/Province Code is missing or invalid. Please enter the 2 to 5 character abbreviation for the state or province of the address that contains it.</td>
</tr>
<tr>
<td>INVALID_SHIP_TO_STATE</td>
<td>The Ship To State/Province Code is missing or invalid. Please enter the 2 to 5 character abbreviation for the state or province of the address that contains it.</td>
</tr>
<tr>
<td>INVALID_STATE</td>
<td>Signup prospect state '{1}' is invalid.</td>
</tr>
<tr>
<td>INVALID_STATUS</td>
<td>You may not change this issue’s status from '{1}' to '{2}'.</td>
</tr>
<tr>
<td>INVALID_SUB</td>
<td>The subsidiary restrictions on this record are incompatible with those defined for account: '{1}'. Subsidiary access on this record must be a subset of those permitted by the account.</td>
</tr>
<tr>
<td>INVALID_SUB</td>
<td>The subsidiary restrictions on this record are incompatible with those defined for account: '{1}'. Subsidiary access on this record must be a superset of those permitted by the account.</td>
</tr>
<tr>
<td>INVALID_SUB</td>
<td>The subsidiary restrictions on this record are incompatible with those defined for department: '{1}'. Subsidiary access on this record must be a subset of those permitted by the department.</td>
</tr>
<tr>
<td>INVALID_SUB</td>
<td>The subsidiary restrictions on this record are incompatible with those defined for item: '{1}'. Subsidiary access on this record must be a superset of those permitted by the item.</td>
</tr>
<tr>
<td>INVALID_SUB</td>
<td>The subsidiary restrictions on this record are incompatible with those defined for location: '{1}'. Subsidiary access on this record must be a subset of those permitted by the location.</td>
</tr>
<tr>
<td>INVALID_SUB</td>
<td>The Subsidiary selected doesn't match the bank account selected.</td>
</tr>
<tr>
<td>INVALID_SUB</td>
<td>This record does not support multiple subsidiary restrictions. You must choose a single subsidiary.</td>
</tr>
<tr>
<td>INVALID_SUB</td>
<td>Transaction references multiple subsidiaries</td>
</tr>
<tr>
<td>INVALID_SUB</td>
<td>You may not add members to a group/kit/assembly unless the subsidiaries for those members completely contain the subsidiaries of the group/kit/assembly.</td>
</tr>
<tr>
<td>INVALID_SUB</td>
<td>'{1}' can not be used with the selected subsidiary</td>
</tr>
<tr>
<td>INVALID_SUBSCRIPTION_STATUS</td>
<td>You cannot change the global subscription status from its current value of '{1:status name}'.</td>
</tr>
<tr>
<td>INVALID_SUBSCRIPTION_STATUS</td>
<td>You cannot set the global subscription status to the value '{1:status name}'.</td>
</tr>
<tr>
<td>INVALID_SUPERVISOR</td>
<td>Employees can not be their own supervisor.</td>
</tr>
<tr>
<td>INVALID_SUPERVISOR</td>
<td>You can’t insert this employee record as it would create a loop in the supervisor hierarchy.</td>
</tr>
<tr>
<td>INVALID_TASK_ID</td>
<td>The task ID: '{1}' is not valid. Please refer to the documentation for a list of supported task IDs.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INVALID_TASK_ID</td>
<td>You have specified an invalid task Id</td>
</tr>
<tr>
<td>INVALID_TXR_CODE</td>
<td>Invalid Canadian Tax Code: {1}</td>
</tr>
<tr>
<td>INVALID_TXR_CODES</td>
<td>Invalid Tax Code(s): {1}</td>
</tr>
<tr>
<td>INVALID_TXR_CODE_FOR_SUB</td>
<td>The selected tax code is not available in subsidiary.</td>
</tr>
<tr>
<td>INVALID_TXR_VALUE</td>
<td>GST and PST amount cannot be negative!</td>
</tr>
<tr>
<td>INVALID_TXR_VALUE</td>
<td>GST tax value is not a valid number: {1}</td>
</tr>
<tr>
<td>INVALID_TIME_FORMAT</td>
<td>(1) is not a valid time and it should use the following format h:mm a.</td>
</tr>
<tr>
<td>INVALID_TO_DATE</td>
<td>invalid 'to' date</td>
</tr>
<tr>
<td>INVALID_TRACKING_NUM</td>
<td>The tracking number is not valid.</td>
</tr>
<tr>
<td>INVALID_TRACKING_NUM</td>
<td>You have entered a tracking number that exceeds the maximum size of {1} characters: (2). Multiple tracking numbers must be separated by spaces, tabs, or commas. Slash (/), semicolon (;), colon (:), or any other character that is not a space or a comma will be interpreted as a part of the tracking number.</td>
</tr>
<tr>
<td>INVALID_TRANS</td>
<td>This transaction is not valid.</td>
</tr>
<tr>
<td>INVALID_TRANSACTION_DATE</td>
<td>Transaction date {1} is not valid. Transaction dates may be at most (2) years in the past and (3) years in the future.</td>
</tr>
<tr>
<td>INVALID_TRANACTIO_DATE</td>
<td>There are no Accounting Periods that cover this transaction date.</td>
</tr>
<tr>
<td>INVALID_TRANS_COMPNT</td>
<td>You have entered an invalid component for this transaction.</td>
</tr>
<tr>
<td>INVALID_TRANS_SUB_ACCT</td>
<td>Transaction subsidiary {1} is not valid for account {2}. Please choose a different account.</td>
</tr>
<tr>
<td>INVALID_TRANS_SUB_CLASS</td>
<td>Transaction subsidiary {1} is not valid for class {2}. Please choose a different class.</td>
</tr>
<tr>
<td>INVALID_TRANS_SUB_DEPT</td>
<td>Transaction subsidiary {1} is not valid for department {2}. Please choose a different department.</td>
</tr>
<tr>
<td>INVALID_TRANS_SUB_ENTITY</td>
<td>Transaction subsidiary {1} is not valid for entity {2}. Please choose a different entity.</td>
</tr>
<tr>
<td>INVALID_TRANS_SUB_ITEM</td>
<td>Transaction subsidiary {1} is not valid for item {2}. Please choose a different item.</td>
</tr>
<tr>
<td>INVALID_TRANS_SUB_LOC</td>
<td>Transaction subsidiary {1} is not valid for location {2}. Please choose a different location.</td>
</tr>
<tr>
<td>INVALID_TRANS_TYP</td>
<td>Transaction type specified is incorrect.</td>
</tr>
<tr>
<td>INVALID_UNSUPRTD_RCRD_TYP</td>
<td>Invalid or unsupported record type: {1}</td>
</tr>
<tr>
<td>INVALID_UPS_ACCT</td>
<td>An invalid UPS Account Number was entered. Please verify you have entered the correct Shipper Number and re-submit the form.</td>
</tr>
<tr>
<td>INVALID_UPS_PACKG_WEIGHT</td>
<td>UPS requires a minimum package weight of .1 LBS and a maximum package weight of 150 LBS. Please adjust the package weights accordingly and resubmit the fulfillment.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INVALID_UPS_VALUES</td>
<td>UPS did not accept the entered values for the following fields. Please go back and correct these values:</td>
</tr>
<tr>
<td>INVALID_URL</td>
<td>Please begin the (1) url with &lt;b&gt;http://&lt;/b&gt;  or  &lt;b&gt;https://&lt;/b&gt;  Examples of a valid (1)url are:&lt;b&gt;<a href="http://www.mydomain.com/image.gif">http://www.mydomain.com/image.gif</a>&lt;/b&gt;  or  &lt;b&gt;<a href="https://one.two.org/user-name/test.jpg">https://one.two.org/user-name/test.jpg</a>&lt;/b&gt;</td>
</tr>
<tr>
<td>INVALID_URL</td>
<td>Request for invalid URL: (1)</td>
</tr>
<tr>
<td>INVALID_VAT_AMOUNT</td>
<td>VAT amount cannot be negative</td>
</tr>
<tr>
<td>INVALID_VAT_REGSTRTN_NUM</td>
<td>Invalid VAT registration number (1).</td>
</tr>
<tr>
<td>INVALID_WO</td>
<td>You have an invalid work order (1) or the order is already closed.</td>
</tr>
<tr>
<td>INVALID_WORLDPAY_ID</td>
<td>Exchange source does not recognize your WorldPay ID. Please check that it is correct.</td>
</tr>
<tr>
<td>INVALID_WO_ITEM</td>
<td>Special Work Order Items can not be Drop Ship or Special Order</td>
</tr>
<tr>
<td>INVALID_YEAR</td>
<td>Invalid year (1)</td>
</tr>
<tr>
<td>INVALID_YEAR_FORMAT</td>
<td>Illegal year format or value. Examples: 1999, 2000, 2001, etc.</td>
</tr>
<tr>
<td>INVALID_ZIP_FILE</td>
<td>Invalid archive. Zip file must contain at least one file.</td>
</tr>
<tr>
<td>INVALID_ZIP_POST_CODE</td>
<td>The submitted Zip/Postal Code is invalid. This field may only contain a maximum of 16 digits, spaces, and the dash character (-).</td>
</tr>
<tr>
<td>INVENTORY_NUM_DISALLWD</td>
<td>Inventory numbers are only allowed on items with serial numbered or lot numbered items.</td>
</tr>
<tr>
<td>ISSUE_ASSIGNEE_DISALLWD</td>
<td>The specified assignee is disallowed for this issue's status.</td>
</tr>
<tr>
<td>ISSUE_PRODUCT_VERSION_MISMATCH</td>
<td>Cannot set issue (1) to (2) (3) and (4) (5) because that product is not associated with that version.</td>
</tr>
<tr>
<td>ISSUE_VERSION_BUILD_MISMATCH</td>
<td>Issue version and build do not match.</td>
</tr>
<tr>
<td>ITEM_ACCT_REQD</td>
<td>One of the items on this transaction has an amount but no account. Please fix the item and resubmit the transaction.</td>
</tr>
<tr>
<td>ITEM_ACCT_REQD</td>
<td>One of the items on this transaction has an amount but no account. Please fix the item and resubmit the transaction. It might be that you have recently elected to charge for shipping and have not assigned an account to the shipping item that is included in this transaction.</td>
</tr>
<tr>
<td>ITEM_ACCT_REQD</td>
<td>You must specify asset and COGS accounts for this inventory item.</td>
</tr>
<tr>
<td>ITEM_COUNT_MISMATCH</td>
<td>COGS_CORRECTION: 2 means of calculating the item count do not match for item: (1) vs (2))</td>
</tr>
<tr>
<td>ITEM_COUNT_MISMATCH</td>
<td>COGS_CORRECTION: 2 means of calculating the item count do not match for item: (1) vs (2)) There are transactions in the system in which this item is used but the asset account for that item is not the current Asset Account in the item record</td>
</tr>
<tr>
<td>ITEM_IS_UNAVAILABLE</td>
<td>(Item is unavailable)</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ITEM_NAME_MUST_BE_UNIQUE</td>
<td>An item with that name already exists. Please choose another name</td>
</tr>
<tr>
<td>ITEM_NOT_UNIQUE</td>
<td>The item [1] is not unique.</td>
</tr>
<tr>
<td>ITEM_PARAM_REQD_IN_URL</td>
<td>Error - Item parameter (id=nmn) was not provided on the URL</td>
</tr>
<tr>
<td>ITEM_TYP_REQS_UNIT</td>
<td>Items of type (1) require (2) unit</td>
</tr>
<tr>
<td>ITEM_TYP_REQS_UNIT</td>
<td>Items of type (1) subtype (2) require (3) unit</td>
</tr>
<tr>
<td>JE_AMOUNTS_MUST_BALANCE</td>
<td>The amounts in the journal entry must balance.</td>
</tr>
<tr>
<td>JE_LINE_MISSING_REQD_DATA</td>
<td>(1) are mandatory on all lines of the journal entry.</td>
</tr>
<tr>
<td>JE_REV_REC_IN_PROGRESS</td>
<td>This account is currently processing Revenue Recognition Journal Entries. Only one such process is allowed at a time.</td>
</tr>
<tr>
<td>JE_UNEXPECTED_ERROR</td>
<td>Journal Entries failed to be created due to unexpected error.</td>
</tr>
<tr>
<td>JOB_NOT_COMPLETE</td>
<td>The specified job is not complete yet</td>
</tr>
<tr>
<td>JS_EXCEPTION</td>
<td>A JavaScript Exception was thrown</td>
</tr>
<tr>
<td>KEY_REQD</td>
<td>Empty key not allowed for [1]</td>
</tr>
<tr>
<td>LABEL_REQD</td>
<td>Please enter a value for Label</td>
</tr>
<tr>
<td>LANGUAGE_SETUP_REQD</td>
<td>Please go to company preference to add language to translate.</td>
</tr>
<tr>
<td>LINKED_ACCT_DONT_MATCH</td>
<td>You are attempting to link transaction line items, but items on the lines do not match. This can happen when you create a fulfillment from a sales order, a receipt from a purchase order, an invoice from a sales order, a vendor bill from a purchase order, or a reimbursement from a purchase. Please verify that items in the transaction you are creating match the items in the originating transaction.</td>
</tr>
<tr>
<td>LINKED_ITEMS_DONT_MATCH</td>
<td>Linked items don’t match</td>
</tr>
<tr>
<td>LIST_ID_REQD</td>
<td>Required field missing in a related list. You must set [1].</td>
</tr>
<tr>
<td>LIST_KEY_REQD</td>
<td>There is no list key for field (1) of list (2). Please assign a key and resubmit your task.</td>
</tr>
<tr>
<td>LOCATIONS_IN_USE</td>
<td>Your classes cannot be converted to locations because your existing location records are referred to by transactions or other records. These location records cannot be overwritten.</td>
</tr>
<tr>
<td>LOCATIONS_SETUP_REQD</td>
<td>You must first define locations (Lists-&gt;Locations-&gt;New) before you can distribute inventory.</td>
</tr>
<tr>
<td>LOCATIONS_SETUP_REQD</td>
<td>You must first define locations (Lists-&gt;Locations-&gt;New) before you can transfer inventory.</td>
</tr>
<tr>
<td>LOCATION_REQD</td>
<td>You must specify a location in order to use (1) numbers when Multi-Location Inventory is enabled</td>
</tr>
<tr>
<td>LOCKED_DASHBOARD</td>
<td>Your dashboard has been set up and locked by an administrator. Please contact them for details.</td>
</tr>
<tr>
<td>LOGIN_DISABLED</td>
<td>Invalid login. Customer access is disabled.</td>
</tr>
<tr>
<td>LOGIN_DISABLED</td>
<td>Login access has been disabled for this role.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LOGIN_DISABLED</td>
<td>Your access to {1} has been deactivated. Please contact the company's administrator to re-activate your access.</td>
</tr>
<tr>
<td>LOGIN_DISABLED</td>
<td>Your access to this account has been removed or disabled. Please contact the account administrator.</td>
</tr>
<tr>
<td>LOGIN_DISABLED_PARTNER_CTR</td>
<td>Disabled login: Advanced Partner Center access has been disabled by the account administrator.</td>
</tr>
<tr>
<td>LOGIN_DISABLED_PARTNER_CTR</td>
<td>Disabled login: Standard Partner Center access has been disabled by the account administrator.</td>
</tr>
<tr>
<td>LOGIN_EMAIL_REQD</td>
<td>Invalid login. You must provide an email address.</td>
</tr>
<tr>
<td>LOGIN_NAME_AND_PSWD_REQD</td>
<td>Please enter both a user name and a password.</td>
</tr>
<tr>
<td>LOGIN_REQD</td>
<td>You must <code>&lt;a href='/pages/login.jsp' target='_self'&gt;log in&lt;/a&gt;</code> before accessing this page.</td>
</tr>
<tr>
<td>LOST_UPSELL_CRITERIA</td>
<td>Your upsell criteria were lost. This is probably due to a transient condition such as a server reboot. Click <code>&lt;a href='#onclick='history.go(-1);'&gt;here&lt;/a&gt;</code> to go back and try again.</td>
</tr>
<tr>
<td>MACHN_LIST_KEY_NAMES_REQD</td>
<td>Server application error: no list key names are defined for field {1} of record of type {2}.</td>
</tr>
<tr>
<td>MANDATORY_PRD_TYPE_REQD</td>
<td>Please select the mandatory period type...</td>
</tr>
<tr>
<td>MATRIX_INFO_TEMP_LOST</td>
<td>Matrix item information was lost. This was probably due to a transient condition like a server reboot. Please try again.</td>
</tr>
<tr>
<td>MATRIX_SUBITEM_NAME_TOO_LONG</td>
<td>The following matrix sub-item name is too long (80 character max):&lt;p&gt; {1} &lt;p&gt; Please shorten your parent item name or your option abbreviations.</td>
</tr>
<tr>
<td>MAX_16_LINES_ALLWD_PER_BILLPAY</td>
<td>A maximum of 16 lines per payee can be applied per online bill payment.</td>
</tr>
<tr>
<td>MAX_200_LINES_ALLWD_ON_TRANS</td>
<td>Journal Entries can have a maximum of 200 lines.</td>
</tr>
<tr>
<td>MAX_BULK_MERGE_RCRDS_EXCEEDED</td>
<td>You cannot perform a bulk merge operation with a group larger than {1} records.</td>
</tr>
<tr>
<td>MAX_EMAILS_EXCEEDED</td>
<td>This campaign email event exceeds the number of emails ({1}) that can be sent per month without setting up a default campaign domain or specifying one on the campaign email template.</td>
</tr>
<tr>
<td>MAX_EMAILS_EXCEEDED</td>
<td>This merge operation exceeds the number of emails ({1}) that can be sent per month without setting up a bulk merge domain or specifying one on the email template.</td>
</tr>
<tr>
<td>MAX_MERGE_LIMIT_EXCEEDED</td>
<td>You can merge a maximum of 25 records at a time.</td>
</tr>
<tr>
<td>MAX_MERGE_RCRDS_EXCEEDED</td>
<td>You can merge a maximum of {1} records at a time.</td>
</tr>
<tr>
<td>MAX_RCRDS_EXCEEDED</td>
<td>The maximum number of records allowed for a (1) operation has been exceeded.</td>
</tr>
<tr>
<td>MEDIA_FILE_INVALID_JSCRIPT</td>
<td>Media file was of type javascript and would not compile. Error on line:</td>
</tr>
<tr>
<td>MEDIA_NOT_FOUND</td>
<td>Media item not found {1}</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MEDIA_NOT_INITIALIZED</td>
<td>Media Item cannot be initialized</td>
</tr>
<tr>
<td>MERGE_OPERATION_DISALWD</td>
<td>You cannot perform merge operations on records that belong to your group.</td>
</tr>
<tr>
<td>MERGE_RCRD_REQD</td>
<td>You must specify a record to merge into</td>
</tr>
<tr>
<td>METAVANTE_SECRET_ANSWER_REQD</td>
<td>Missing Secret Answer. A secret answer is required by the Metavante CSP service. It cannot be null or empty.</td>
</tr>
<tr>
<td>METAVANTE_SECRET_QUESTION_REQD</td>
<td>Missing or invalid Secret Question ID. A valid Secret Question is required by the Metavante CSP service. Please refer to Table 2, above, for a list of valid Secret Question IDs.</td>
</tr>
<tr>
<td>METAVANTE_SETUP_REQD</td>
<td>Your NetSuite account is not currently integrated with a Metavante Online Bill Pay account. &lt;br&gt;To set up an active account, you need to reapply to Metavante. &lt;br&gt;Go to Setup &gt; Set Up Online Bill Pay and follow the instructions on that page to apply for a Metavante account.</td>
</tr>
</tbody>
</table>
| METAVANTE_TEMP_UNAVAILBL                    | Metavante is temporarily unavailable. Please try again later.  
If you would like to print the payment to mail yourself, click Back, and then click the date of the payment on the Approve Online Bill Payments page. When the payment's detail record appears, clear the Bill Pay box and either check the To Be Printed box and click Submit or click the Print button. |
<p>| MISMATCHED_CURRENCY                         | The transaction currency does not match the names currency                                                                                                    |
| MISMATCHED_QTY_PRICING                      | Quantities do not match accross pricings                                                                                                                     |
| MISMATCHED_SEARCH_PARENTHESIS               | Search error: Parentheses are unbalanced.                                                                                                                     |
| MISMATCH_EVENT_ISSUE_STATUS                 | Event status ([1]) and issue base status ([2]) do not match                                                                                                   |
| MISMATCH_ISSUE_PRODUCT_VERSION              | Issue product and version do not match.                                                                                                                      |
| MISSING_ACCT_PRD                            | You are attempting to create an amortization or revenue recognition schedule outside the range of available accounting periods. Please adjust the periods on this transaction or go to Setup&gt;Accounting&gt;Manage Accounting Periods to set up more periods. |
| MISSING_ENUM                                | No Enumerated Value [1] for Enumerated Type [2]                                                                                                               |
| MISSING_REQD_FLD                            | Missing required value for mandatory field: [1]                                                                                                               |
| MISSING_REQD_FLD                            | The Company record does not have all required fields set. Please ensure the State, Zip/Postal Code, and Country fields are set and try your request again.        |
| MISSING_ACCT_PRD                            | Unable to find an Accounting Period for the allocation date.                                                                                                 |
| MISSNG_REV_REC_RCRD                         | Unable to locate Revenue Recognition records.                                                                                                                 |
| MISSNG_SO_REV_REC_PARAMS                    | Unable to get Revenue Recognition parameters from originating sales order.                                                                                  |
| MISSNG_SO_START_END_DATES                   | Unable to acquire start and end date from Sales Order.                                                                                                       |</p>
<table>
<thead>
<tr>
<th>Error Code Returned</th>
<th>Long Description or Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLI_REQD</td>
<td>Multi-location Inventory Error (MLI_LOCATION_REQUIRED): this transaction or its items must have locations.</td>
</tr>
<tr>
<td>MLTIPLE_TAX_LINES_DISALLWD</td>
<td>Multiple Tax lines for line item in transaction:</td>
</tr>
<tr>
<td>MSNG_FIELD_OWRTE_MUST_BE_TRUE</td>
<td>The missingFieldOverwrite attribute must be true when updating a salesOrder.</td>
</tr>
<tr>
<td>MST_UPDATE_ITEMS_THEN_RATES</td>
<td>You cannot update items and shipping rates at the same time on transactions that have multiple shipping routes enabled. You must first update the items, then get the transaction and update the shipping rates separately.</td>
</tr>
<tr>
<td>MULTISELECT_TYPE_REQD</td>
<td>Server application error: no multiselect type is defined for field (1) of (2) record type.</td>
</tr>
<tr>
<td>MULTI_ACCT_CANT_CHANGE_PSWD</td>
<td>The password cannot be changed here because the email address is associated with multiple accounts. The user must change their password via the link in the settings portal of the home page.</td>
</tr>
<tr>
<td>MULTI_SHIP_ROUTES_REQD</td>
<td>(1) (2) has multiple shipping routes enabled, which is only supported in version 2008_2 and newer. You are not allowed to update any shipping fields on this record.</td>
</tr>
<tr>
<td>NAME_ALREADY_IN_USE</td>
<td>A mass update has already been saved with that name. Please use a different name.</td>
</tr>
<tr>
<td>NAME_ALREADY_IN_USE</td>
<td>A search has already been saved with that name. Please use a different name.</td>
</tr>
<tr>
<td>NAME_REQQ</td>
<td>Missing Name. Name is a required field and it cannot be null or empty.</td>
</tr>
<tr>
<td>NAME_TYPE_FLDR_FIELDS_REQD</td>
<td>missing required fields : name, type, and folder</td>
</tr>
<tr>
<td>NARROW_KEYWORD_SEARCH</td>
<td>Please provide more detailed keywords so your search does not return too many results.</td>
</tr>
<tr>
<td>NEGATIVE_PAYMENT_DISALLWD</td>
<td>Negative payments not allowed</td>
</tr>
<tr>
<td>NEGATIVE_TAX_RATE_DISALLWD</td>
<td>A Tax rate cannot be negative</td>
</tr>
<tr>
<td>NEW_CONNECTION_DISALLWD</td>
<td>Not allowed to create new connections.</td>
</tr>
<tr>
<td>NONMATCHING_EMAILS</td>
<td>Email addresses don't match</td>
</tr>
<tr>
<td>NONZERO_AMT_REQD</td>
<td>You did not enter non-zero amounts for any accounts.</td>
</tr>
<tr>
<td>NON_ADMIN_CANT_INITIATE_LINK</td>
<td>This user cannot integrate with a partner.</td>
</tr>
<tr>
<td>NOT_IN_INVNT</td>
<td>You may not distribute (1) numbers that are not currently in inventory. You attempted to distribute the following (1) numbers that were not in inventory: (2)</td>
</tr>
<tr>
<td>NO_DATA_FOUND</td>
<td>No data was found</td>
</tr>
<tr>
<td>NO_EXPENSES_FOR_PRD</td>
<td>The Allocation sources or destinations did not have any expenses associated with them for the selected period.</td>
</tr>
<tr>
<td>NO_ITEMS_TO_PRINT</td>
<td>There are no items to print</td>
</tr>
<tr>
<td>NO_MASS_UPDATES_RUNNING</td>
<td>There are currently no mass updates running.</td>
</tr>
<tr>
<td>NO_MTRX_ITEMS_TO_UPDATE</td>
<td>There are no matrix subitems to update.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NO_ORD_SHPMNT</td>
<td>There is no shipment on that order.</td>
</tr>
<tr>
<td>NO_RCRDS_MATCH</td>
<td>No Records matched your request.</td>
</tr>
<tr>
<td>NO_RCRD_FOR_USER</td>
<td>There is no record for this user in the company's entity table. (emaillogin.semail='{1}', kentity={2})</td>
</tr>
<tr>
<td>NO_SCHDUL_APPLIED</td>
<td>There were no schedules that need to be applied to the given period.</td>
</tr>
<tr>
<td>NO_SCHDUL_APPLIED</td>
<td>There were no schedules that need to be applied to the input accounting period.</td>
</tr>
<tr>
<td>NULL_CHECK_NUMBER</td>
<td>Null Check Number</td>
</tr>
<tr>
<td>NUMERIC_CHECK_NUM_REQD</td>
<td>Invalid Check Number. Check number must be a numeric value and can be at most 7 digits long.</td>
</tr>
<tr>
<td>NUM_ITEMS_GRTR_THAN_QTY</td>
<td>The number of {1} entered ({2}) is greater than the item quantity ({3}).</td>
</tr>
<tr>
<td>NUM_ITEMS_NOT_EQUAL_TO_QTY</td>
<td>The number of {1} entered ({2}) is not equal to the item quantity ({3}).</td>
</tr>
<tr>
<td>NUM_REQD_FOR_FIRST_LABEL</td>
<td>No number was specified for the first label.</td>
</tr>
<tr>
<td>OI_FEATURE_REQD</td>
<td>You have not enabled Outlook Integration feature for your account.</td>
</tr>
<tr>
<td>OI_PERMISSION_REQD</td>
<td>You do not have permission to access Outlook Integration feature.</td>
</tr>
<tr>
<td>ONE_ADMIN_REQD_PER_ACCT</td>
<td>This operation would leave your account without an active Administrator. In order to successfully perform the mass update, please deselect at least one entity with an Administrator role.</td>
</tr>
<tr>
<td>ONE_ADMIN_REQD_PER_ACCT</td>
<td>You can't delete this employee. No administrators for this account would remain.</td>
</tr>
<tr>
<td>ONE_ADMIN_REQD_PER_ACCT</td>
<td>You can't inactivate {1}. The account would be left with no active administrators.</td>
</tr>
<tr>
<td>ONE_ADMIN_REQD_PER_ACCT</td>
<td>You can't remove the administrator role from this user. No administrators for this account would remain.</td>
</tr>
<tr>
<td>ONE_EMPL_REQD</td>
<td>At least one employee is required to process payroll</td>
</tr>
<tr>
<td>ONE_POSITIVE_VALUE_REQD</td>
<td>You must enter at least one positive value for at least one item.</td>
</tr>
<tr>
<td>ONE_RCRD_REQD_FOR_MASS_UPDATE</td>
<td>Please create at least one {1} before using this mass update.</td>
</tr>
<tr>
<td>ONE_ROLE_REQD</td>
<td>You can't inactivate all roles. You would not be able to log in.</td>
</tr>
<tr>
<td>ONLINE_BILL_PAY_SETUP_REQD</td>
<td>&lt;b&gt;{1}&lt;/b&gt; is not set up for Online Bill Pay. To set up this payee, click Go Back. When the Approve Online Bill Payments page appears, click Enable Payee in the Enabled for Billpay column. When the payee's record appears, check Enable Online Bill Pay and submit these required fields: &lt;li&gt;Legal Name&lt;/li&gt; &lt;li&gt;Print on Check As&lt;/li&gt; &lt;li&gt;Phone&lt;/li&gt; &lt;li&gt;Billing Address&lt;/li&gt; Then, go to Transactions &gt; Approve Online Bill Payments to approve the payment.</td>
</tr>
<tr>
<td>ONLINE_FORM_DSNT_EXIST</td>
<td>This online form does not exist.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ONLINE_FORM_ID_REQD</td>
<td>Missing required online form ID</td>
</tr>
<tr>
<td>ONLINE_FORM_USER_ACCESS_ONLY</td>
<td>This form is only accessible to online form users.</td>
</tr>
<tr>
<td>ONLINE_ORD_FEATURE_DISABLED</td>
<td>Can't open store for {1}. This company does not have the &lt;b&gt;Use Sales Orders&lt;/b&gt; feature enabled. The feature is required for customers to make online purchases.</td>
</tr>
<tr>
<td>ONLY_ONE_CONTRIB_ITEM_REQD</td>
<td>Only one instance of a company contribution item is allowed on an employee record.</td>
</tr>
<tr>
<td>ONLY_ONE_DEDCT_ITEM_REQD</td>
<td>Only one instance of a deduction item is allowed on an employee record.</td>
</tr>
<tr>
<td>ONLY_ONE_DISTRIBUT_ALLOWED</td>
<td>You may not distribute {1} numbers more than once. You attempted to distribute the following {1} numbers more than once: {2}</td>
</tr>
<tr>
<td>ONLY_ONE_EARNING_ITEM_REQD</td>
<td>Only one instance of an earning item is allowed on an employee record.</td>
</tr>
<tr>
<td>ONLY_ONE_LOT_NUM_ALLWD</td>
<td>You may not enter more than a single serial/lot number before an item is selected.</td>
</tr>
<tr>
<td>ONLY_ONE_PREF_BIN_ALLWD</td>
<td>There may be at most one preferred bin per location for an item. The following location has more than one preferred bin for this item: {1}</td>
</tr>
<tr>
<td>ONLY_ONE_PREF_BIN_ALLWD</td>
<td>You may not have more than one preferred bin per item.</td>
</tr>
<tr>
<td>ONLY_ONE_UNIT_AS_BASE_UNIT</td>
<td>Only one unit may be designated as the base unit.</td>
</tr>
<tr>
<td>ONLY_ONE_UPLOAD_ALLWD</td>
<td>You cannot upload more than one file at a time.</td>
</tr>
<tr>
<td>ONLY_ONE_WITHLD_ITEM_REQD</td>
<td>Only one instance of a withholding item is allowed on an employee record.</td>
</tr>
<tr>
<td>ORDER_DSNEXIST</td>
<td>That order does not exist.</td>
</tr>
<tr>
<td>ORD_ALREADY_APPROVED</td>
<td>You cannot cancel this order because it has already been approved.</td>
</tr>
<tr>
<td>OVERAGE_DISALLOWED</td>
<td>Overage is not allowed.</td>
</tr>
<tr>
<td>OVERLAPPING_PRDS_DISALLOWED</td>
<td>Illegal period structure. Overlapping periods.</td>
</tr>
<tr>
<td>OVERLAPPING_PRDS_DISALLOWED</td>
<td>There is an overlapping period. Please check your Active or Inactive Periods to ensure that there is not an existing period.</td>
</tr>
<tr>
<td>OVER_FULFILL_DISALLOWED</td>
<td>You can not over-fulfill an item unless you have selected the 'Allow Overage on Item Fulfillments' preference.</td>
</tr>
<tr>
<td>OWNER_REQD</td>
<td>You cannot make a contact private without an owner</td>
</tr>
<tr>
<td>PACKAGE_WEIGHT_REQD</td>
<td>Attempted to create a package without specifying a nonzero package weight.</td>
</tr>
<tr>
<td>PACKG_LEVEL_REF_DISALLOWED</td>
<td>Package level reference numbers are not allowed for shipments whose origin/destination pair is not US/US or Puerto Rico/ Puerto Rico.</td>
</tr>
<tr>
<td>PACKG_VALUE_TOO_LARGE</td>
<td>Package declared value cannot be greater than $999.00 USD</td>
</tr>
<tr>
<td>PARENT_CANT_ITSELF_BE_MEMBER</td>
<td>Parent item cannot be a member of itself</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PARENT_MUST_BE_MATRIX_ITEM</td>
<td>A Child matrix item’s parent must be a matrix item</td>
</tr>
<tr>
<td>PARENT_REQD</td>
<td>A Child matrix item must have its parent specified</td>
</tr>
<tr>
<td>PARTNER_ACCESS_DENIED</td>
<td>Partners do not have access to this item.</td>
</tr>
<tr>
<td>PARTNER_CODE_ALREADY_USED</td>
<td>A partner with that partner code ({1}) already exists.</td>
</tr>
<tr>
<td>PAYCHECK_IN_USE</td>
<td>You cannot clear this paycheck because it is linked to by one or more liability payments.</td>
</tr>
<tr>
<td></td>
<td>You must delete or void those transactions first</td>
</tr>
<tr>
<td>PAYEE_REQD_FOR_PMT</td>
<td>Your payment has been recorded, but an online bill pay payment will not be made because no</td>
</tr>
<tr>
<td></td>
<td>payee was specified. You should return to the payment screen if you wish to print the</td>
</tr>
<tr>
<td></td>
<td>check.</td>
</tr>
<tr>
<td>PAYPAL_FUND_SOURCE_REQD</td>
<td>Please return to PayPal to select a different funding source.</td>
</tr>
<tr>
<td>PAYPAL_INVALID_PMT_METHOD</td>
<td>Paypal is unable to process this payment. Please select an alternate payment method.</td>
</tr>
<tr>
<td>PAYPAL_INVALID_PMT_METHOD</td>
<td>Your PayPal account is not configured to use Express Checkout. Please follow directions</td>
</tr>
<tr>
<td></td>
<td>on the PayPal payment method record.</td>
</tr>
<tr>
<td>PAYPAL_PMT_NOTIFICATION</td>
<td>PayPal Payment Notification</td>
</tr>
<tr>
<td>PAYPAL_SETUP_REQD</td>
<td>The account referenced by this paypal id is not setup to use express checkout. Please</td>
</tr>
<tr>
<td></td>
<td>return to the paypal setup page and follow directions for setting up paypal express</td>
</tr>
<tr>
<td></td>
<td>checkout.</td>
</tr>
<tr>
<td>PAYROLL_EXPENSE_ACCT_REQD</td>
<td>Please select an expense account for payroll item &lt;a href='/'app/common/item/payrollitem.nl?id={1}&amp;e=T}&gt;{2}&lt;/a&gt;</td>
</tr>
<tr>
<td>PAYROLL_EXPENSE_ACCT_REQD</td>
<td>Please select an expense account for payroll item &lt;a href='/app/common/item/payrollitem.nl?id={1}&amp;e=T}&gt;{2}&lt;/a&gt;</td>
</tr>
<tr>
<td>PAYROLL_FEATURE_DISABLED</td>
<td>You have not enabled the Payroll feature.</td>
</tr>
<tr>
<td>PAYROLL_FEATURE_UNAVAILABLE</td>
<td>You are trying to edit a Pay Cheque - Payroll is not available in NetSuite Canada.</td>
</tr>
<tr>
<td>PAYROLL_ITEM_DELETE_DISALLWD</td>
<td>Unable to remove payroll item: {1} - There are existing transactions for this payroll item.</td>
</tr>
<tr>
<td></td>
<td>You may mark it inactive instead.</td>
</tr>
<tr>
<td>PAYROLL_LIABILITY_ACCT_REQD</td>
<td>Please select a liability account for payroll item &lt;a href='/app/common/item/payrollitem.nl?id={1}&amp;e=T}&gt;{2}&lt;/a&gt;</td>
</tr>
<tr>
<td>PERMISSION_VIOLATION</td>
<td>Permission Violation: you may not access this record.</td>
</tr>
<tr>
<td>PERMISSION_VIOLATION</td>
<td>Permission Violation: you may no longer edit this record.</td>
</tr>
<tr>
<td>PHONE_NUM_REQD</td>
<td>Please provide a phone number.</td>
</tr>
<tr>
<td>PLAN_IN_USE</td>
<td>This plan has already been used to generate commission calculations and can't be deleted.</td>
</tr>
<tr>
<td>PLAN_OVERLAP_DISALLWD</td>
<td>Plan overlap is not permitted. You have attempted to assign someone to this plan for a</td>
</tr>
<tr>
<td></td>
<td>time period that overlaps with another plan.</td>
</tr>
<tr>
<td>PMT_ALREADY_APPRVD</td>
<td>The payment has already been approved and sent to the bill pay carrier for processing.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PMT_ALREADY_EXISTS</td>
<td>A payment with the same amount and date already exists for this payee.</td>
</tr>
<tr>
<td>PMT_ALREADY_SBMTD</td>
<td>This payment has already been submitted for online bill pay.</td>
</tr>
<tr>
<td>PMT_EDIT_DISALLWD</td>
<td>Access to this Bill Pay transaction is restricted, and it cannot be modified. Transactions can only be modified until 3PM CST on the payment date.</td>
</tr>
<tr>
<td>PMT_EDIT_DISALLWD</td>
<td>This liability payment cannot be edited while it has an Automated Clearing House transmission in process.</td>
</tr>
<tr>
<td>POSITIVE_QTY_REQD</td>
<td>Assembly member items must have positive quantities</td>
</tr>
<tr>
<td>POSTING_PRD_SETUP_REQD</td>
<td>Creation of Journal Entries require a single Accounting Period value across all Revenue Recognition events. Please setup a 'Posting Period' filter.</td>
</tr>
<tr>
<td>PRDS_DISALLWD NAMES_NOT_UNIQUE</td>
<td>After adding new periods, not all names would be unique.</td>
</tr>
<tr>
<td>PRD_SETUP_REQD</td>
<td>You must change your period definitions to contain fiscal years. Please visit 'Setup-&gt;Manage Accounting Periods' and click 'Set Up Year'.</td>
</tr>
<tr>
<td>PRD_SETUP_REQD</td>
<td>You must define the periods of the prior fiscal year. Please visit 'Setup-&gt;Manage Accounting Periods' and click 'Set Up Year'.</td>
</tr>
<tr>
<td>PREFERRED_TAX_AGENCY_REQD</td>
<td>A preferred Tax Agency has been deleted - Please choose a new one in '&lt;a href=&quot;/app/setup/acctsetup.nl&quot;&gt;Set Up Accounting&lt;/a&gt;'</td>
</tr>
<tr>
<td>PREFERRED_TAX_AGENCY_REQD</td>
<td>Error: No preferred Tax Agencies have been set up (go to '&lt;a href=&quot;/app/setup/acctsetup.nl&quot;&gt;Set Up Accounting&lt;/a&gt;')</td>
</tr>
<tr>
<td>PREF_VENDOR_COST_REQD</td>
<td>Drop ship/Special Order items must have a preferred vendor and a purchase price.</td>
</tr>
<tr>
<td>PREF_VENDOR_REQD</td>
<td>Drop ship/Special Order items must have a preferred vendor for each of the {1} the item is accessible to.</td>
</tr>
<tr>
<td>PRIVATE_RCRD_ACCESS_DISALLWD</td>
<td>You cannot view or edit this record because it is marked private.</td>
</tr>
<tr>
<td>PRIVATE_STATUS_CHNG_DISALLWD</td>
<td>You cannot make this contact private.</td>
</tr>
<tr>
<td>PSWD_EXPIRED</td>
<td>Password has expired. Please change your NetSuite password before continuing.</td>
</tr>
<tr>
<td>PSWD_REQD</td>
<td>A password must be entered when granting login access privileges to this record.</td>
</tr>
<tr>
<td>PSWD_REQD</td>
<td>Password is empty.</td>
</tr>
<tr>
<td>PSWD_REQD</td>
<td>Please type your password into both fields.</td>
</tr>
<tr>
<td>PSWD_REQD</td>
<td>You must provide a password to give this person access to your account.</td>
</tr>
<tr>
<td>PWSDS_DONT_MATCH</td>
<td>New passwords don't match.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PWDS_DONT_MATCH</td>
<td>The passwords you entered do not match. Please reenter your passwords.</td>
</tr>
<tr>
<td>PWDS_DONT_MATCH</td>
<td>The Passwords you entered do not match. Please reenter your passwords.</td>
</tr>
<tr>
<td>PWDS_DONT_MATCH</td>
<td>The passwords you have entered do not match.</td>
</tr>
<tr>
<td>QTY_EXCEEDED_QTY_BUCKETS</td>
<td>More quantities defined than there are quantity buckets</td>
</tr>
<tr>
<td>QTY_REQD</td>
<td>Quantities must be defined</td>
</tr>
<tr>
<td>RATE_REQUEST_SHPMNT_REQD</td>
<td>The rate request shipment value has not been set.</td>
</tr>
<tr>
<td>RATE_SRVC_UNAVAILBL</td>
<td>The rate for this service is not available for the specified source and destination addresses.</td>
</tr>
<tr>
<td>RCRD_DSNT_EXIST</td>
<td>Group Record Not Found</td>
</tr>
<tr>
<td>RCRD_DSNT_EXIST</td>
<td>That record does not exist.</td>
</tr>
<tr>
<td>RCRD_DSNT_EXIST</td>
<td>There are no records of this type.</td>
</tr>
<tr>
<td>RCRD_HAS_Been_CHANGED</td>
<td>Record has been changed</td>
</tr>
<tr>
<td>RCRD_ID_NOT_INT</td>
<td>Record id is not integer: {1}</td>
</tr>
<tr>
<td>RCRD_NOT_FOUND</td>
<td>Could not find any records by this name.</td>
</tr>
<tr>
<td>RCRD_PREVSLY_DELETED</td>
<td>This record has already been deleted.</td>
</tr>
<tr>
<td>RCRD_PREVSLY_DELETED</td>
<td>This record has been deleted since the list was generated.</td>
</tr>
<tr>
<td>RCRD_REF_RCRD_TYP_MISMATCH</td>
<td>The record type and its object reference are not matched.</td>
</tr>
<tr>
<td>RCRD_SUB_MISMATCH_WITH_CLASS</td>
<td>The subsidiary restrictions on this record are incompatible with those defined for class: {1}. Subsidiary access on this record must be a subset of those permitted by the class.</td>
</tr>
<tr>
<td>RCRD_TYPE_REQD</td>
<td>The record type is required.</td>
</tr>
<tr>
<td>RCRD_UNEDITABLE</td>
<td>That record is not editable.</td>
</tr>
<tr>
<td>RECALCING_PLAN_SCHDUL</td>
<td>Cannot inactivate a plan when schedules in the plan are recalculating. Try again when recalculation is complete.</td>
</tr>
<tr>
<td>RECURSV_REF_DISALLWD</td>
<td>ERROR: Recursive Reference</td>
</tr>
<tr>
<td>RECUR_EVENT_DISALLWD</td>
<td>A yearly event cannot be on the 29th of February</td>
</tr>
<tr>
<td>REQD_LOC_FIELDS_MISSING</td>
<td>Location {1} does not have all required fields set. Please ensure the State, Zip/Postal Code, and Country fields are set and try your request again.</td>
</tr>
<tr>
<td>REQD_SUB_FIELDS_MISSING</td>
<td>The Subsidiary {1} does not have all required fields set. Please ensure the State, Zip/Postal Code, and Country fields are set and try your request again.</td>
</tr>
<tr>
<td>REQUEST_PARAM_REQD</td>
<td>This request is missing a required parameter.</td>
</tr>
<tr>
<td>REVERSAL_DATE WARNING</td>
<td>Reversal Date is in a closed accounting period. Please go to Manage Accounting Periods and re-open the accounting period.</td>
</tr>
<tr>
<td>REV_REC_DATE_REQD</td>
<td>No Revenue Recognition Start Date Specified</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>REV_REC_TMPLT_DATA_MISSING</td>
<td>One or more line items on this transaction have Variable Revenue Recognition Templates, but do not have the required {1} also populated. Please either change the Template for these items or indicate which {1} will be used to schedule the recognition of revenue.</td>
</tr>
<tr>
<td>REV_REC_UPDATE_DISALLWD</td>
<td>Modification of revenue recognition related information on this item is not allowed because revenue has been recognized for this or related lines.</td>
</tr>
<tr>
<td>ROLE_REQD</td>
<td>Please specify a role to which access should be granted</td>
</tr>
<tr>
<td>ROLE_REQUIRED</td>
<td>In order to login, a role is required unless a default has been previously set.</td>
</tr>
<tr>
<td>ROUNDBING_DIFF_TOO_BIG</td>
<td>rounding difference too big -&gt; tax1: {1} /tax2: {2}</td>
</tr>
<tr>
<td>ROUNDBING_ERROR</td>
<td>Rounding Error: {1}</td>
</tr>
<tr>
<td>ROUTING_NUM_REQD</td>
<td>Missing Routing Number. Bank routing number is a required field and it cannot be null or empty.</td>
</tr>
<tr>
<td>SAME_ACCT_TYP_REQD_FOR_PARENT</td>
<td>Parent account must be of same account type.</td>
</tr>
<tr>
<td>SCHEDUL_EDIT_DISALLWD</td>
<td>This schedule cannot be edited as it has already been used for commission calculations. Please go back and select 'save as new' instead.</td>
</tr>
<tr>
<td>SEARCH_DATE_FILTER_REQD</td>
<td>The search must have a date column as an available filter</td>
</tr>
<tr>
<td>SEARCH_INTEGER_REQD</td>
<td>Please enter an integer number to search on.</td>
</tr>
<tr>
<td>SEARCH_TIMED_OUT</td>
<td>Your search has timed out. If your search includes the '[1]' operator, try using '(2)' instead. If your search includes broad search criteria, try narrowing the criteria.</td>
</tr>
<tr>
<td>SEARCH_TIMED_OUT</td>
<td>Your search has timed out. This might be avoided by using a smaller page size.</td>
</tr>
<tr>
<td>SECURE_TRANS_REQD_ON_CHECKOUT</td>
<td>Store Server Error: As configured, this server does not permit secure transactions, required by store checkout.</td>
</tr>
<tr>
<td>SERIAL_NUM_MATCH_MULTI_ITEMS</td>
<td>{1} different items match this serial number. Select an item from the item dropdown.</td>
</tr>
<tr>
<td>SESSION_TERMD_2ND_LOGIN_DECTD</td>
<td>You can have a maximum of {1} active users at a time in {2}. If you would like to add active users, please contact your account manager to discuss your upgrade options. Or, you may choose to inactivate an existing user before adding a new one. Note that Employee Center users are not included in this total.</td>
</tr>
<tr>
<td>SESSION_TERMD_2ND_LOGIN_DECTD</td>
<td>You can have a maximum of {1} active users at a time that are enabled for Offline Sales Client. If you would like to add active users, please contact your account manager to discuss your upgrade options. Or, you may choose to disable Offline Sales Client on an existing user before enabling a new one.</td>
</tr>
<tr>
<td>SESSION_TIMED_OUT</td>
<td>Your connection has timed out. Please log in again.</td>
</tr>
<tr>
<td>SESSION_TIMED_OUT</td>
<td>Your connection has timed out. Please &lt;a href=&quot;/pages/login.jsp&quot; target=&quot;_self&quot;&gt;log in&lt;/a&gt; again.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SESSION_TIMED_OUT</td>
<td>Your session has timed out. Please re-enter your information and try again.</td>
</tr>
<tr>
<td>SETUP_METER_REQD</td>
<td>Please set up this meter.</td>
</tr>
<tr>
<td>SET_SHIPPING_PICKUP_TYP</td>
<td>Please verify that you have selected a pickup type under setup shipping.</td>
</tr>
<tr>
<td>SHIPMNT_INSURANCE_NOT_AVAILABLE</td>
<td>Insurance is not available when shipping to the destination country: {1}</td>
</tr>
<tr>
<td>SHIP_ADDR_REQD</td>
<td>Shipping address is incomplete.</td>
</tr>
<tr>
<td>SHIP_MANIFEST_ALREADY_PRCSSD</td>
<td>A Shipping Manifest has already been processed for the requested date/time {1}.</td>
</tr>
<tr>
<td>SHIP_MANIFEST_ERROR</td>
<td>No Shipments found to generate a Shipping Manifest for close date {1} for meter {2}.</td>
</tr>
<tr>
<td>SHIP_MANIFEST_ERROR</td>
<td>No Shipping Manifest files found in FedEx Directory for report only request for meter {1}.</td>
</tr>
<tr>
<td>SHIP_SETUP_REQD</td>
<td>No {1} registration was found for the location selected. Please select a different shipping item, or go to Setup &gt; Set Up Shipping to register a {2} account for this location.</td>
</tr>
<tr>
<td>SHIP_TALBE_UNBALNCD</td>
<td>The Shipping Table is not balanced. Please review the table and ensure there is a Charge for every Range Value, and that there are no duplicates.</td>
</tr>
<tr>
<td>SITE_DOMAIN_NAME_REQD</td>
<td>Notice: URL Components cannot be used until you have established a domain name for your site.</td>
</tr>
<tr>
<td>SO_HAS_CHILD_TRANS</td>
<td>This salesOrder has a one or more child transactions associated with it, and cannot be updated.</td>
</tr>
<tr>
<td>SO_LINE_HAS_PO</td>
<td>Error: A Drop Ship/Special Order already exists for sales order {1}, line {2}.</td>
</tr>
<tr>
<td>SRVC_UNAVAILBL_FOR_LOC</td>
<td>The requested service is unavailable between the selected locations.</td>
</tr>
<tr>
<td>START_DATE_AFTER_END_DATE</td>
<td>The start date must preceed the end date.</td>
</tr>
<tr>
<td>START_DATE_REQD</td>
<td>Please enter a value for {1} Start Date</td>
</tr>
<tr>
<td>STATE_REQD</td>
<td>State is a required field and it cannot be null or empty.</td>
</tr>
<tr>
<td>STATUS_ASSIGNEE_REQD</td>
<td>The issue status {1} does not define an assignee issue role. That status may not be used until this is corrected.</td>
</tr>
<tr>
<td>STORAGE_LIMIT_EXCEEDED</td>
<td>You entered a value that will exceed the internal storage limit of {1}. Please reduce the number.</td>
</tr>
<tr>
<td>STORE_ALIAS_UNAVAILABLE</td>
<td>The Store alias you chose &quot;{1}&quot; is already taken. Please go back and choose another.</td>
</tr>
<tr>
<td>STORE_DOMAIN_UNAVAILABLE</td>
<td>The store domain name you chose '{1}' is already taken. Please go back and choose another.</td>
</tr>
<tr>
<td>SUBITEM_REQD</td>
<td>You must first select the new subitems on the matrix tab you want to add.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SUBITEM_REQD</td>
<td>You must first select the subitems on the matrix tab you want to create.</td>
</tr>
<tr>
<td>SUBSIDIARY_MISMATCH</td>
<td>The employee and billable customer must be in the same subsidiary.</td>
</tr>
<tr>
<td>SUB_TAX_AGENCY_REQD</td>
<td>No tax agency defined for subsidiary</td>
</tr>
<tr>
<td>SUCCESS_TRANS</td>
<td>The transaction was entered (1) successfully, (2)</td>
</tr>
<tr>
<td>SUPRT_CNTR_LOGIN_ERROR</td>
<td>(1) Support Center login error: we are unable to find the customer record for account=account</td>
</tr>
<tr>
<td>TAX_ACCT_SETUP_REQD</td>
<td>Tax Accounts Not Defined.</td>
</tr>
<tr>
<td>TAX_CODES_SETUP_PROBLEM</td>
<td>The tax codes haven't been set properly</td>
</tr>
<tr>
<td>TAX_CODES_SETUP_REQD</td>
<td>Can't open store for (1). This company does not have its tax codes fully set up. This is required to properly calculate taxes on international, other-province and same-province orders.</td>
</tr>
<tr>
<td>TAX_CODES_SETUP_REQD</td>
<td>The company is not usable. Administrator hasn't set up the tax codes.</td>
</tr>
<tr>
<td>TAX_CODE_REQD</td>
<td>No default tax code is defined for country (1)</td>
</tr>
<tr>
<td>TAX_GROUP_SETUP_REQD</td>
<td>You have not created tax groups in your NetSuite account. To ensure that your customers are charged the correct amount of sales tax, you must create tax groups by entering them manually at Lists &gt; Accounting &gt; Tax Groups &gt; New.</td>
</tr>
<tr>
<td>TAX_PRD_REQD</td>
<td>No Current Tax Period is defined. &lt;a href=&quot;/app/setup/period/taxperiods.nl&quot;&gt;Click here&lt;/a&gt; to create a tax period.</td>
</tr>
<tr>
<td>TAX_SETUP_REQD</td>
<td>The tax period range (1) has not been defined. Please visit &lt;A href='{2}'&gt;Setup &gt; Accounting &gt; Manage Tax Periods&lt;/A&gt;' to define this period or set up your year.</td>
</tr>
<tr>
<td>TEMPLATE_NOT_FOUND</td>
<td>Template not found</td>
</tr>
<tr>
<td>THIRD_PARTY_BILLING_ACCT_REQD</td>
<td>A 3rd Party Billing Account Number must be provided when selecting a 3rd Party Billing Type.</td>
</tr>
<tr>
<td>TICKET_NOT_LOCATED</td>
<td>The ticket (1) cannot be located in the error database. If this is from a customer logged case, the error may not yet be inserted into the system.</td>
</tr>
<tr>
<td>TOPIC_REQD</td>
<td>You must select and add a topic to this solution.</td>
</tr>
<tr>
<td>TRANSACTION_DELETED</td>
<td>The transaction you are attempting to access has been deleted.</td>
</tr>
<tr>
<td>TRANS_AMTS_UNBALNCD</td>
<td>Transaction is not in balance! amounts+taxes+shipping: (1), total amount: (2)</td>
</tr>
<tr>
<td>TRANS_APPLIED_AMTS_UNBALNCD</td>
<td>Transaction is not in balance! Total to apply of $(1) does not equal sum of applied $2 and unapplied $3</td>
</tr>
<tr>
<td>TRANS_APPLIED_AMTS_UNBALNCD</td>
<td>Transaction is not in balance! Total to apply of $(1) does not equal sum of payment $2 and credits $3 and deposits $4</td>
</tr>
<tr>
<td>TRANS_CLASS_UNBALNCD</td>
<td>Transaction out of balance for class (1) total = (2).</td>
</tr>
<tr>
<td>TRANS_DEPT_UNBALNCD</td>
<td>Transaction out of balance for department (1) total = (2).</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>TRANS DOES NOT EXIST</td>
<td>No transaction exists for that entity.</td>
</tr>
<tr>
<td>TRANS_DSNT_EXIST</td>
<td>The transaction you are attempting to access does not exist.</td>
</tr>
<tr>
<td>TRANS_EDIT_DISALLWD</td>
<td>This transaction is in a period that has been closed. You may not edit it.</td>
</tr>
<tr>
<td>TRANS_EDIT_DISALLWD</td>
<td>You cannot edit this transaction. (1) does not support the imported transaction.</td>
</tr>
<tr>
<td>TRANS_FORGN_CRNKY_MISMATCH</td>
<td>Transaction and foreign currency account use different currencies.</td>
</tr>
<tr>
<td>TRANS_FORGN_CUR_UNBALNCD</td>
<td>Transaction was not in balance (Foreign currency). Posting total = (1)</td>
</tr>
<tr>
<td>TRANS_FORGN_CUR_UNBALNCD</td>
<td>Transaction was not in balance (Foreign currency). Total = (1)</td>
</tr>
<tr>
<td>TRANS_IN_USE</td>
<td>This transaction cannot be deleted because it is linked to by one or more transactions. Click <code>&lt;a href='/app/accounting/transactions/payments.nl?id={1}&amp;label={2}&amp;type={3}&amp;alllinks=T'&gt;here&lt;/a&gt;</code> to see the list of linked transactions.</td>
</tr>
<tr>
<td>TRANS_LINES_UNBALNCD</td>
<td>Transaction is not in balance! Line item sum of $1 does not equal amount of $2.</td>
</tr>
<tr>
<td>TRANS_LINES_UNBALNCD</td>
<td>Transaction is not in balance! Line item sum of $1 does not equal applied amount of $2.</td>
</tr>
<tr>
<td>TRANS_LINE_AND_PMT_UNBALNCD</td>
<td>Transaction is not in balance! Line item sum of $1 not equal to payment amount $2.</td>
</tr>
<tr>
<td>TRANS_LOC_UNBALNCD</td>
<td>Transaction out of balance for location (1) total = (2).</td>
</tr>
<tr>
<td>TRANS_NOT_CLEANED</td>
<td>Transaction not cleaned up.</td>
</tr>
<tr>
<td>TRANS_NOT_COMPLETED</td>
<td>Transaction was not complete.</td>
</tr>
<tr>
<td>TRANS_UNBALNCD</td>
<td>Transaction is not in balance! (1), (2), othercount = (3)</td>
</tr>
<tr>
<td>TRANS_UNBALNCD</td>
<td>Transaction was not in balance. Posting total = (1)</td>
</tr>
<tr>
<td>TRANS_UNBALNCD</td>
<td>Transaction was not in balance. Total = (1)</td>
</tr>
<tr>
<td>TRAN_DATE_REQD</td>
<td>Missing transaction date.</td>
</tr>
<tr>
<td>TRAN_LINE_FX_AMT_REQD</td>
<td>Missing foreign currency amount on non-variance transaction line</td>
</tr>
<tr>
<td>TRAN_LINK_FX_AMT_REQD</td>
<td>Missing foreign currency amount on non-variance transaction link</td>
</tr>
<tr>
<td>TRAN_PRD_CLOSED</td>
<td>This action cannot be completed because it requires modification of the transaction in a closed period due to foreign exchange variance. You may either open the period for this transaction or use the same rate ((1)) between the transactions that will be linked.</td>
</tr>
<tr>
<td>TWO_FA_REQD</td>
<td>Two-Factor Authentication required</td>
</tr>
<tr>
<td>UNABLE_TO_PRINT_CHECKS</td>
<td>Unable to print checks.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UNABLE_TO_PRINT_DEPOSITS</td>
<td>Unable to print deposits.</td>
</tr>
<tr>
<td>UNAUTH_CAMPAIGN_RSPNS_RQST</td>
<td>Unauthorized campaign response request</td>
</tr>
<tr>
<td>UNAUTH_UNSUBSCRIBE_RQST</td>
<td>Unauthorized unsubscribe request</td>
</tr>
<tr>
<td>UNDEFINED_ACCTNG_PRD</td>
<td>The accounting period range {1} has not been defined. Please visit '&lt;A href='/app/setup/period/fiscalperiods.nl'&gt;Setup &gt; Accounting &gt; Manage Accounting Periods&lt;/A&gt;' to define this period or set up your year.</td>
</tr>
<tr>
<td>UNDEFINED_ACCTNG_PRD</td>
<td>The comparison accounting period range {1} has not been defined. Please visit '&lt;A href='/app/setup/period/fiscalperiods.nl'&gt;Setup &gt; Accounting &gt; Manage Accounting Periods&lt;/A&gt;' to define this period or set up your year.</td>
</tr>
<tr>
<td>UNDEFINED_ACCTNG_PRD</td>
<td>The default accounting period for this report has not been defined. Please visit '&lt;A href='/app/setup/period/fiscalperiods.nl'&gt;Setup &gt; Accounting &gt; Manage Accounting Periods&lt;/A&gt;' to define this period or set up your year.</td>
</tr>
<tr>
<td>UNDEFINED_CSTM_FIELD</td>
<td>Undefined customfield.</td>
</tr>
<tr>
<td>UNDEFINED_TAX_PRD</td>
<td>The default tax period for this report has not been defined. Please visit '&lt;A href='/app/setup/period/taxperiods.nl'&gt;Setup &gt; Accounting &gt; Manage Tax Periods&lt;/A&gt;' to define this period or set up your year.</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>An error occurred while processing item options.</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>An unexpected error has occurred.</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>An unexpected error has occurred. A FedEx Shipping Label was not generated.</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>An unexpected error has occurred while generating this content.&lt;p&gt;Our Customer Support staff have been notified and are looking into the problem.&lt;/p&gt;</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>An unexpected error has occurred while synching a record. Click [OK] to skip the record and continue.</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>An unexpected error has occurred. Technical Support has been alerted to this problem.</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>An unexpected error occurred while extracting email from SMTP server.</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>An unexpected error occurred while logging email request completion.</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>An unexpected error occurred while logging email request failure</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>An unexpected error occurred while logging email request start</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>An unexpected error occurred while processing the payment.</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>An unexpected error occurred with the group SQL.</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>An Unexpected JavaScript Error has occurred</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>Error</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>Error: {1}</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>Please specify an scompid</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>Problem during commission calculation</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>An unexpected error occurred.</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>Dto java class is not defined for {1}.</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>Server error: no dto class is defined for record of type {1}.</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>Server error: missing database entries in WSRecordElement and WSNameSpace table for object of {1}</td>
</tr>
<tr>
<td>UNEXPECTED_ERROR</td>
<td>Application error: no form request class is defined for record of type {1}</td>
</tr>
<tr>
<td>UNIQUE_CONTACT_NAME_REQD</td>
<td>Contact names must be unique</td>
</tr>
<tr>
<td>UNIQUE_CUST_EMAIL_REQD</td>
<td>A customer record with this email address already exists. You must enter a unique customer email address for each record you create.</td>
</tr>
<tr>
<td>UNIQUE_CUST_EMAIL_REQD</td>
<td>A customer record with this email address already exists. You must enter a unique customer email address for each record you create. To correct this record, click &lt;a href='javascript:history.go(-1);'&gt;back&lt;/a&gt; and enter a new customer email address in the Customer field. Then, click Submit.</td>
</tr>
<tr>
<td>UNIQUE_CUST_ID_REQD</td>
<td>A customer record with this ID already exists. You must enter a unique customer ID for each record you create.</td>
</tr>
<tr>
<td>UNIQUE_CUST_ID_REQD</td>
<td>A customer record with this ID already exists. You must enter a unique customer ID for each record you create. To correct this record, click &lt;a href='javascript:history.go(-1);'&gt;back&lt;/a&gt; and enter a new customer ID in the Customer field. Then, click Submit.</td>
</tr>
<tr>
<td>UNIQUE_ENTITY_NAME_REQD</td>
<td>multiple sub-customers or jobs have name '{1}' which would create a naming conflict upon merge. All names must be unique. Before merging, you must change one of the subs named '{2}' to something else.</td>
</tr>
<tr>
<td>UNIQUE_GROUPID_REQD</td>
<td>You must specify exactly one numeric groupId</td>
</tr>
<tr>
<td>UNIQUE_PARTNER_CODE_REQD</td>
<td>{1:name of partner record} Code &quot;{2:partner code}&quot; already exists. Please select a unique code for each record.</td>
</tr>
<tr>
<td>UNIQUE_QTY_REQD</td>
<td>Quantities must be unique</td>
</tr>
<tr>
<td>UNIQUE_RCRD_ID_REQD</td>
<td>A record with this ID already exists. You must enter a unique ID in order to create or update this record.</td>
</tr>
<tr>
<td>UNIQUE_SOLUTION_CODE_REQD</td>
<td>A solution with this particular solution code already exists. Please assign a different code.</td>
</tr>
<tr>
<td>UNITS_TYP_IN_USE</td>
<td>This units type is used by {1} [2]. You must delete the [2] and all associated transactions in order to delete this units type.</td>
</tr>
<tr>
<td>UNKNOWN_CARRIER</td>
<td>Package Tracking is not available for id (1). Unknown carrier.</td>
</tr>
<tr>
<td>UNKNOWN_RCRD_TYPE</td>
<td>Unknown record type</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UNKNOWN_SCRIPT_TYP</td>
<td>Unknown Script Type</td>
</tr>
<tr>
<td>UNKWN_ALLOCNT_SCHED_FREQ_TYP</td>
<td>Unable to determine allocation schedule frequency type.</td>
</tr>
<tr>
<td>UNKNWN_EMAIL_AUTHOR</td>
<td>The author of this email cannot be found.</td>
</tr>
<tr>
<td>UNRECOGNIZED_METHOD</td>
<td>unrecognized method '{1}'</td>
</tr>
<tr>
<td>UNSUBSCRIBE_REQD</td>
<td>Unsubscribe is mandatory, please enter a value for this field.</td>
</tr>
<tr>
<td>UNSUPRTD_DOC_TYP</td>
<td>You attempted to upload an unsupported document type. Please try again with a selection from the list below:</td>
</tr>
<tr>
<td>UPDATE_DISALLWD</td>
<td>Update is not allowed</td>
</tr>
<tr>
<td>UPDATE_PRICE_AMT_REQD</td>
<td>Please specify an amount to update prices.</td>
</tr>
<tr>
<td>UPGRADE_WS_VERSION</td>
<td>Could not set '{1}' to field '{2}' of record number '{3}' due to schema enumeration restriction.</td>
</tr>
<tr>
<td>UPGRADE_WS_VERSION</td>
<td>Please consider upgrading to endpoint '{1}'</td>
</tr>
<tr>
<td>UPGRADE_WS_VERSION</td>
<td>Sales order &lt;id '{1}'&gt; contains item serial/lot numbers that are not supported in your client application. You are not allowed to update serial/lot numbers on this sales order. Contact your software vendor for the latest Web Services upgrade.</td>
</tr>
<tr>
<td>UPGRADE_WS_VERSION</td>
<td>Sales order &lt;id '{1}'&gt; has items with more than one serial/lot numbers that is not supported in your client application. The serial/lot numbers have been removed to successfully return the sales order. Contact your software vendor for the latest Web Services upgrade.</td>
</tr>
<tr>
<td>UPGRADE_WS_VERSION</td>
<td>This '{1}' has multiple '{2}'s. Web Services schema version '{3}' or greater is required to modify '{2}' for this '{1}'</td>
</tr>
<tr>
<td>UPGRADE_WS_VERSION</td>
<td>'{1} '{2}' has multiple shipping routes enabled, which is only supported in version 2008_2 and newer. The shipping information has been omitted to successfully return this record.</td>
</tr>
<tr>
<td>UPS_CANT_INTEGRATE_FULFILL</td>
<td>The fulfillment cannot be integrated with UPS because the Shipping Integration Carrier is not set to UPS.</td>
</tr>
<tr>
<td>UPS_CONFIG_ERROR</td>
<td>A UPS configuration error occurred. Please contact tech support.</td>
</tr>
<tr>
<td>UPS_LICENSE_AGREEMNT_REQD</td>
<td>You must agree to the UPS license agreement</td>
</tr>
<tr>
<td>UPS_ONLINE_RATE_UNAVAILBL</td>
<td>The UPS Online Realtime Rates System is temporarily unavailable. Please resubmit your rate request in a few minutes.</td>
</tr>
<tr>
<td>UPS_ONLINE_RATE_UNAVAILBL</td>
<td>UPS did not return any rates for the specified origin and destination addresses.</td>
</tr>
<tr>
<td>UPS_ONLINE_SHIP_UNAVAILBL</td>
<td>The UPS Online Shipping System is temporarily unavailable. Please resubmit your fulfillment in a few minutes.</td>
</tr>
<tr>
<td>UPS_REG_NUM_IN_USE</td>
<td>The submitted UPS Registration Number, '{1}', is already in use. Please resubmit the registration with a different UPS registration Number.</td>
</tr>
<tr>
<td>UPS_SETUP_REQD</td>
<td>No UPS registration was found. Please register your UPS account with NetSuite before attempting to send a fulfillment request to UPS.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UPS_VOID_ERROR</td>
<td>The UPS Void failed due to a system failure.</td>
</tr>
<tr>
<td>UPS_XML_ERROR</td>
<td>XML Sent to UPS. UPS returned error code/text.</td>
</tr>
<tr>
<td>URL_ID_PARAM_REQD</td>
<td>URL is missing the id parameter. The file could not be retrieved.</td>
</tr>
<tr>
<td>URL_REQD</td>
<td>You must enter a URL for this media item.</td>
</tr>
<tr>
<td>USERDISABLED</td>
<td>user disabled</td>
</tr>
<tr>
<td>USER_ERROR</td>
<td>An error occurred during your last update.</td>
</tr>
<tr>
<td>USER_ERROR</td>
<td>A User Error Has Occurred</td>
</tr>
<tr>
<td>USER_ERROR</td>
<td>Detach requires an AttachBasicReference</td>
</tr>
<tr>
<td>USERERROR</td>
<td>Either internalId or externalId is required.</td>
</tr>
<tr>
<td>USER_ERROR</td>
<td>Folder cannot be made a subfolder of itself.</td>
</tr>
<tr>
<td>USER_ERROR</td>
<td>Gift Certificate From, Recipient Name, and Recipient Email are required.</td>
</tr>
<tr>
<td>USER_ERROR</td>
<td>Invalid Attachment record combination</td>
</tr>
<tr>
<td>USER_ERROR</td>
<td>Missing Item Weight or Weight Unit.</td>
</tr>
<tr>
<td>USER_ERROR</td>
<td>Missing or Invalid RecordType for AttachTo</td>
</tr>
<tr>
<td>USER_ERROR</td>
<td>Must submit a non-abstract instance of baseRef (eg RecordRef, CustomRecordRef) NOT a baseRef</td>
</tr>
<tr>
<td>USER_ERROR</td>
<td>Must submit a non-abstract instance of record or searchRecord (eg customer or customerSearchBasic).</td>
</tr>
<tr>
<td>USER_ERROR</td>
<td>{1}</td>
</tr>
<tr>
<td>USPS_ACCT_NUM_ALREADY_EXISTS</td>
<td>There is an existing NetSuite registration for Endicia account number {1}.</td>
</tr>
<tr>
<td>USPS_INVALID_INSURED_VALUE</td>
<td>Insured value exceeds the {1} maximum allowed by Endicia.</td>
</tr>
<tr>
<td>USPS_INVALID_PACKAGING</td>
<td>The Carrier Packaging that you have selected is not valid for this item fulfillment.</td>
</tr>
<tr>
<td></td>
<td>Usually this indicates the selected packaging cannot be used with the selected USPS shipping</td>
</tr>
<tr>
<td></td>
<td>method, or the package weight is invalid. Please check the documentation for more details.</td>
</tr>
<tr>
<td>USPS_INVALID_PSWD</td>
<td>The Endicia Web Password does not match the Web Password for this USPS Registration account</td>
</tr>
<tr>
<td></td>
<td>number.</td>
</tr>
<tr>
<td>USPS_LABEL_VOIDED</td>
<td>This error required 1 or more labels created for this transaction to be voided at Endicia.</td>
</tr>
<tr>
<td>USPS_MAX_ITEM_EXCEEDED</td>
<td>International USPS fulfillments allow a maximum of 5 unique items per package, due to customs</td>
</tr>
<tr>
<td></td>
<td>documentation. If more than one package is required, please break up the shipment into the</td>
</tr>
<tr>
<td></td>
<td>multiple fulfillments.</td>
</tr>
<tr>
<td>USPS_ONE_PACKAGE_ALLWD</td>
<td>International USPS fulfillments allow only one package. If more than one package is required,</td>
</tr>
<tr>
<td></td>
<td>please break up the shipment into multiple fulfillments of one package each.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>USPS_PASS_PHRASE_NOT_UPDATED</td>
<td>The Endica Pass Phrase was not updated: {1}</td>
</tr>
<tr>
<td>USPS_REFUND_FAILED</td>
<td>Failed Endicia Refund Request</td>
</tr>
<tr>
<td>USPS_REFUND_FAILED</td>
<td>The Endicia Refund Request failed due to a system error.</td>
</tr>
<tr>
<td>USPS_RETRY</td>
<td>A response was not received for the USPS funding request. Please try again in a few minutes.</td>
</tr>
<tr>
<td>USPS_VALIDATE_ADDR</td>
<td>The address you entered could not be validated. Please verify the city, state, and/or zip code.&lt;br&gt;You can validate an address by visiting the <a href="http://zip4.usps.com/zip4/welcome.jsp">U.S. Postal Service</a> or the <a href="http://www.endicia.com/Developers/ZipLookup/">Endicia</a> web site.</td>
</tr>
<tr>
<td>USPS_VERIFY_TRACKING_NUM</td>
<td>Please verify that the following tracking numbers were created and voided in your Endicia account before proceeding.</td>
</tr>
<tr>
<td>USPS_VOID_ERROR</td>
<td>An error was detected during the Endicia Void operation:</td>
</tr>
<tr>
<td>VALID_EMAIL_REQD FOR_LOGIN</td>
<td>Please enter a valid email address when granting login access privileges to this record.</td>
</tr>
<tr>
<td>VALID_FIRST_NAME_REQD</td>
<td>Missing or invalid First Name. Users first name is a required field and cannot be null or empty.</td>
</tr>
<tr>
<td>VALID_LAST_NAME_REQD</td>
<td>Missing or invalid Last Name. Users last name is a required field and cannot be null or empty.</td>
</tr>
<tr>
<td>VALID_PHONE_NUM_REQD</td>
<td>Missing or invalid Home phone number. The Home phone number is a required field and it cannot be null or empty. The format of the Home phone number must contain area code plus seven digit number.</td>
</tr>
<tr>
<td>VALID_PRD_REQD</td>
<td>Insert Transaction Failure: No valid, open, posting period for date - {1}. Please visit Setup &gt; Manage Accounting Periods to set up a new accounting period.</td>
</tr>
<tr>
<td>VALID_PRD_REQD</td>
<td>Insert Transaction Failure: No valid, open, tax period for date - {1}. Please visit Setup &gt; Manage Tax Periods to set up a new tax period.</td>
</tr>
<tr>
<td>VALID_PRD_REQD</td>
<td>Update Transaction Failure: No valid, open, {1} period for date - {2}</td>
</tr>
<tr>
<td>VALID_VERSION_REQD_IN_URL</td>
<td>If the version parameter is passed through the URL, it MUST contain a valid version in a phased release environment. Valid: {1}</td>
</tr>
<tr>
<td>VALID_WORK_PHONE_REQD</td>
<td>Missing or invalid Work phone number. The Work phone number is a required field and it cannot be null or empty. The format of the Work phone number must contain area code plus seven digit number.</td>
</tr>
<tr>
<td>Error Code Returned</td>
<td>Long Description or Message</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>VALID_ZIPCODE_REQD</td>
<td>Missing or invalid ZIP code field. ZIP code is a required field and it cannot be null or empty. ZIP code and state values are checked against an internal database to make sure that ZIP code specified exists in state specified.</td>
</tr>
<tr>
<td>VENDOR_TYPE_REQD</td>
<td>No Vendor Type was specified. If creating a Tax Agency, please ensure that the vendor type is active and marked as a tax agency.</td>
</tr>
<tr>
<td>VERIFY_DESTNTN_ZIP_CODE</td>
<td>Please verify that the destination zipcode is correctly specified.</td>
</tr>
<tr>
<td>VERIFY_ZIP_CODE_SETUP</td>
<td>Please verify that you have correctly set your zip code under setup company. If you have multi-location enabled, verify that you have set a correct zipcode for each location.</td>
</tr>
<tr>
<td>VOIDING_REVERSAL_DISALLWD</td>
<td>You may not create a voiding reversal for transactions with inventory impact. To reverse the inventory impact of the transaction, you will need to create an inventory adjustment.</td>
</tr>
<tr>
<td>VSOE_CANT_ADD_ITEM_GROUP</td>
<td>When the <code>&lt;b&gt;Is VSOE bundle&lt;/b&gt;</code> box is checked, Items for Purchase cannot be added to item groups.</td>
</tr>
<tr>
<td>VSOE_REV_REC_TMPLT_REQD</td>
<td>All Lines in a VSOE Bundle with a VSOE Allocation must have a revenue recognition template.</td>
</tr>
<tr>
<td>VSOE_TOTAL_ALLOCATION_ERROR</td>
<td>The total vsoe allocation in a bundle must equal the total bundle sales amount.</td>
</tr>
<tr>
<td>VSOE_TRAN_VSOE_BUNDLE_ERROR</td>
<td>You have indicated that you would like this transaction to be treated as a Bundle (multi-element arrangement) for VSOE purposes. Please either uncheck the 'Transaction Is VSOE Bundle' checkbox or remove the Item Groups that have the 'Is VSOE Bundle' option specified.</td>
</tr>
<tr>
<td>WRITE_OFF_ACCT_REQD</td>
<td>In order to receive items without restocking, you must first set a value for the write-off account. To set the value of the write-off account, go to Accounting &gt; Accounting Preferences &gt; Order Management &gt; Write-Off Account for Returns.</td>
</tr>
<tr>
<td>WS_CONCUR_SESSION_DISALLWD</td>
<td>Someone has logged in as this user from a different web services session. Only one person may login as a given user at a time. As a consequence, this session has been terminated.</td>
</tr>
<tr>
<td>WS_CONCUR_SESSION_DISALLWD</td>
<td>Only one request may be made against a session at a time.</td>
</tr>
<tr>
<td>WS_EXCEEDED_CONCUR_USERS_ALLWD</td>
<td>You can have a maximum of {1} active concurrent WS users at a time in {2}</td>
</tr>
<tr>
<td>WS_EXCEEDED_MAX_CONCUR_RQST</td>
<td>The maximum number of eBay Order Imports has exceeded the provisioned quantity. Please contact Customer Support for further assistance.</td>
</tr>
<tr>
<td>WS_FEATURE_REQD</td>
<td>You have not enabled web services feature for your account.</td>
</tr>
<tr>
<td>WS_INVALID_SEARCH_OPERATN</td>
<td>When using request-level credentials, you must use the {1} operation instead of {2}</td>
</tr>
<tr>
<td>WS_LOG_IN_REQD</td>
<td>You must log in before performing a web service operation.</td>
</tr>
<tr>
<td>WS_PERMISSION_REQD</td>
<td>You do not have permission to access web services feature.</td>
</tr>
<tr>
<td>ZIP_FILE_CONTAINS_VIRUS</td>
<td>The zip file contains a virus {1}. Upload abort.</td>
</tr>
</tbody>
</table>
Chapter 81 SuiteScript Governance

**Important:** SuiteScript thresholds are based on the volume of activity that a company’s users can manually generate, as well as on a provision for automated functions. 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).

In order 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 given 60 minute time period. For complete details, see Governance on Script Logging.

**Note:** Given that there are no time limits imposed on script execution, NetSuite has put internal mechanisms in place to detect “runaway scripts” that include infinite loops. Once 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.

<table>
<thead>
<tr>
<th>API</th>
<th>Unit Usage per API</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlapiDeleteFile</td>
<td>20</td>
<td>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 Script Usage Unit Limits for the total units allowed for a user event script.)</td>
</tr>
<tr>
<td>nlapiInitiateWorkflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nlapiScheduleScript</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nlapiSubmitConfiguration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nlapiSubmitFile</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nlapiDeleteRecord</td>
<td></td>
<td>When used on standard transactions: 20</td>
</tr>
<tr>
<td>nlapiSubmitRecord</td>
<td></td>
<td>When used on standard non-transactions: 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When used on custom records: 4</td>
</tr>
</tbody>
</table>
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 Usage Unit Limits

<table>
<thead>
<tr>
<th>Script Type</th>
<th>Total Units Allowed per Script</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduled Scripts</td>
<td>10,000</td>
<td>A scheduled script that includes two calls to nlapiTransformRecord, one call to nlapiMergeRecord, and one call to nlapiSendEmail consumes 40 units out of a possible 10,000 available.</td>
</tr>
<tr>
<td>User Event Scripts</td>
<td>1,000</td>
<td>Regardless of the 1,000 unit limit for user event scripts, developers should design their scripts so that they are responsive to users, otherwise user experience may be impacted.</td>
</tr>
</tbody>
</table>
### SuiteScript Governance
### Script Usage Unit Limits

<table>
<thead>
<tr>
<th>Script Type</th>
<th>Total Units Allowed per Script</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client Scripts</td>
<td>1,000</td>
<td>Be aware that client scripts are metered on a per-script basis. 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. Usage units are not shared by all the client scripts associated with a form or record. <strong>Note:</strong> For information on form- and record-level client scripts, see Form-level and Record-level Client Scripts.</td>
</tr>
<tr>
<td>Suitelets</td>
<td>1,000</td>
<td>For example, a Suitelet that calls nlapiCreateRecord and nlapiRequestURL consumes 20 units out of a possible 1,000 units available. <strong>Note:</strong> 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.</td>
</tr>
<tr>
<td>RESTlets</td>
<td>5,000</td>
<td></td>
</tr>
<tr>
<td>Portlet Scripts</td>
<td>1,000</td>
<td>You can have 1000 units per record/invocation of the script.</td>
</tr>
<tr>
<td>Mass Update Scripts</td>
<td>1,000</td>
<td></td>
</tr>
<tr>
<td>Bundle Installation Scripts</td>
<td>10,000</td>
<td>Bundle installation scripts are governed by a maximum of 10,000 units per execution.</td>
</tr>
<tr>
<td>Workflow Action Scripts</td>
<td>1,000</td>
<td>Use workflow action scripts to create custom actions in your workflow. Note that within one workflow state, all actions combined cannot not 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. <strong>See</strong> Workflow Action Usage Units for a list of all workflow actions and the units they consume when executed within a state.</td>
</tr>
<tr>
<td>(also referred to as Custom Actions in SuiteFlow)</td>
<td>1,000</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** NetSuite also enforces a usage limit of 1000 units for SSP application scripts. For more information, see 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.

```javascript
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.

```javascript
var context = nlapiGetContext();
if (parseInt(context.getRemainingUsage()) > 50)
{
    //execute code here
}
```

Governance on Script Logging

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 given 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, however, NetSuite databases, shared by multiple customers, are not flooded with an inordinate amount of logging from only one company.

**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 Deployment](image)

**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 given 60 minute time period).

NetSuite will send notifications through email and will also add a log entry to the script's Execution Log (see figure). 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.
Chapter 82 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?

The Multiple Shipping Routes (MSR) feature allows you to 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.

The following figure shows a sales order with three items. When MSR is enabled in your account, and the Enable Item Line Shipping checkbox 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 the figure below.
### Multiple Shipping Routes and SuiteScript

#### SuiteScript Developer and Reference Guide

![Multiple Shipping Routes and SuiteScript](image)

**Ship from:**
- **Ship to:**
- **Ship Via:**

**Fields:**
- shipaddress
- shipmethod

<table>
<thead>
<tr>
<th>Project</th>
<th>Item</th>
<th>Quantity</th>
<th>Units</th>
<th>Description</th>
<th>Serial/Lot Numbers</th>
<th>Price Level</th>
<th>Rate</th>
<th>Amount</th>
<th>License Code</th>
<th>Tax Code</th>
<th>Commit</th>
<th>Tax Rate</th>
<th>Options</th>
<th>Gift Certificate</th>
<th>Ship To</th>
<th>Carrier</th>
<th>Ship Via</th>
</tr>
</thead>
<tbody>
<tr>
<td>#10</td>
<td>Envelopes</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>130.00</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#10</td>
<td>Envelopes</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>130.00</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assorted Bandages - Large</td>
<td>Assorted Bandages</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>10.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assorted Bandages - Large</td>
<td>Assorted Bandages</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>10.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Autopsy Saver</td>
<td>Stryker Autopsy Saver</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>850.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Autopsy Saver</td>
<td>Stryker Autopsy Saver</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>850.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Discount Rate: 5%*
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).

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.

Note: For additional information on this feature, as well as steps for enabling MSR in your account, see Multiple Shipping Routes in the NetSuite Help Center.
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:
  ```javascript
  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.

<table>
<thead>
<tr>
<th>UI Label</th>
<th>Element Name</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Item Line Shipping</td>
<td>ismultishipto</td>
<td>Set to T to allow for multiple items with separate shipping address/methods</td>
</tr>
<tr>
<td>Ship To</td>
<td>shipaddress</td>
<td>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. <strong>Note:</strong> The Show Internal ID preference must be enabled for address IDs to show.</td>
</tr>
</tbody>
</table>
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.

**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 checkbox to the Items sublist. For steps on adding this checkbox to a custom form, see *Multiple Shipping Routes* in the NetSuite Help Center.

**Note:** Once 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.

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,

```javascript
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');
```
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, once you enable MSR on individual transactions (by selecting the *Enable Item Line Shipping* checkbox 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,
   ```javascript
   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:
   ```javascript
   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.
You can use SuiteScript to create custom addresses that can be used as one-time ship-to and bill-to addresses on transactions. In the UI, this is equivalent to 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 (see Figure 1). The field values you set in SuiteScript are those that appear in the Custom Address popup.

Developers may want to set custom addresses in situations where the bill-to is standard but the ship-to is custom or one-time. This 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.

**Code Sample**

The following is an example of how to set a custom ship-to address on a sales order. Note the following:

1. The 'shipaddresslist' field must be set to an empty string to indicate custom.
2. The 'shipaddress' field is set to the full address, separated with line return characters.
3. If same sample can be applied to setting custom bill-to addresses. The only difference between ship-to and bill-to scripts is that the word ship is substituted for bill, for example shipaddresslist changes to billaddresslist, shipattention changes to billattention, and so on.

```javascript
var record = nlapiCreateRecord('salesorder');
record.setFieldValue('entity', 967);
record.selectNewLineItem('item');
```
record.setCurrentLineItemValue('item', 'item', 14);
record.setCurrentLineItemValue('item', 'quantity', 1);
record.setCurrentLineItemValue('item', 'location', 1);
record.setCurrentLineItemValue('item', 'amount', '19.99');
record.commitLineItem('item');

record.setFieldValue('shipmethod', 92);
record.setFieldValue('exchangerate', '1');
record.setFieldValue('ccapproved', 'T');

// The shipaddresslist field must be set to empty string to indicate custom.
// If setting custom bill to address, change the shipaddresslist internal ID to billtoaddresslist.
record.setFieldValue('shipaddresslist', '');

// The next set of fields are the UI equivalent of the fields that appear in the Custom Address popup
record.setFieldValue('shipattention', 'Bill Hicks');
record.setFieldValue('shipaddressee', 'RantCo Inc.');
record.setFieldValue('shipphone', '999-888-7777');
record.setFieldValue('shipaddr1', '123 Main Street');
record.setFieldValue('shipaddr2', 'Suite 100');
record.setFieldValue('shipcity', 'New York');
record.setFieldValue('shipstate', 'NY');
record.setFieldValue('shipzip', '10004');
record.setFieldValue('shipcountry', 'US');
record.setFieldValue('shipoverride', 'F');
record.setFieldValue('shipisresidential', 'T');

// This represents the Ship Address text field on the Shipping or Billing sublist
record.setFieldValue('shipaddress', 'Bill Hicks
123 Main Street
New York
NY
10004
United States');

id = nlapiSubmitRecord(record, true);
Chapter 84 Referencing the currencyname Field in SuiteScript

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');
```
Part 12  SuiteScript Developer Resources
Chapter 85 SuiteScript Developer

Resources

See these links for SuiteScript developer resources:

- SuiteScript Samples
- SuiteScript Tutorials
- SuiteScript FAQ
- NetSuite User Group
- NetSuite Developer Portal
Chapter 86 SuiteScript Samples

SuiteScript code samples are organized into the following categories:

- Client SuiteScript Samples
- User Event Script Samples
- Scheduled Script Samples
- Suitelets Samples
- Portlet Scripts Samples
Chapter 87 SuiteScript Tutorials

The following tutorials are provided in this section:

- Getting Started with Your SuiteScript Development Environment Tutorial (If you are new to NetSuite, it is recommended that you start with this tutorial.)
- Client SuiteScript Tutorials
- User Event SuiteScript Tutorial

Getting Started with Your SuiteScript Development Environment Tutorial

This tutorial shows you how to write a “Hello World!” Suitelet. You will programmatically create a NetSuite form, add an alert, add a button, and configure the Suitelet so that it can be launched from a NetSuite menu. After launching the Suitelet, you can click the button on the form to receive the “Hello World!” popup message.

Note: If you are new to NetSuite and would like to learn more general information about Suitelets, see Suitelets.

This tutorial guides you through the following tasks, all of which are required for your Suitelet to run in NetSuite:

1. Install and Configure the SuiteCloud IDE.
2. Write a “Hello World!” Suitelet.
5. Configure Suitelet Script Record and Deployment.
6. Test Your script.

Note: You can use any text editor (including Notepad) to write SuiteScript. However, NetSuite strongly recommends you use the SuiteCloud IDE, which offers code completion, script validation, and access to Help on each API. Therefore, for the purpose of the “Getting Started with Your SuiteScript Development Environment” tutorial, basic steps are provided to help you install and configure the SuiteCloud IDE. For more information about working with SuiteCloud IDE, see Working with Your SuiteScript Development Environment (SuiteCloud IDE).
To write a Hello World! Suitelet using the SuiteCloud IDE:

**Install and Configure the SuiteCloud IDE**

1. Download the SuiteCloud IDE.
   
   **Note:** For more information about the download location and available versions for SuiteCloud IDE, see Installing SuiteCloud IDE.

2. Launch the executable file.

3. Select a workspace.
   
   **Important:** NetSuite recommends that you create a new workspace for SuiteCloud IDE instead of reusing an existing workspace. This is to avoid the possibility of carrying over incompatible settings from an old workspace.

4. Accept the terms of the license agreement.

5. In SuiteCloud IDE, set your master password by going to NetSuite > Master Password > Set Master Password.

6. Add your NetSuite accounts by going to NetSuite > Accounts and clicking Add.


8. In the project name field, enter **GettingStarted**.

9. In the project type field, select **SuiteScript Project**.

10. Select **Use default location**.
   
   **Note:** If you do not want to use the default location, deselect **Use default location** and navigate to your desired location.

11. Click Finish.

**Write a “Hello World!” Suitelet**


13. In the script type field, select **Suitelet Script**.

14. In the parent folder field, select **GettingStarted**.

15. In the script filename field, enter **GettingStarted_SS_HelloWorld.js**.

16. Click Finish. A Suitelet script is automatically created.

17. Change the function name to **processHelloWorld**.

18. Edit the function to create a form, use an alert statement, and add a button to display a “Hello World!” popup message when the button is clicked.

```javascript
function processHelloWorld(request, response) {
    var form = nlapiCreateForm('Getting Started with Your SuiteScript Development Environment');
```
var script = 'alert('Hello World!')';

form.addButton('custombutton', 'Click Me', script);
response.writePage(form); //The content of the parameter becomes the http response sent to the browser.
}

19. Click Save.

Upload Your Script into NetSuite

20. Next, right-click in the SuiteCloud IDE editor area and then go to NetSuite > Upload File in Editor. The Upload File in Editor window opens.

   Note: You can also use the shortcut, Ctrl + U, to upload file in editor.

21. Wait for the progress bar to complete. The Upload File in Editor window closes upon completion.

Verify Your Script Upload

22. Right-click in the editor area and then go to NetSuite > Log in to Project Account. A browser loads with your NetSuite account logged in.

   Note: You can also use the shortcut, Ctrl + B, to log in to project account.

23. In NetSuite, go to Documents > Files > File Cabinet. The File Cabinet Folders page of your NetSuite account loads.

24. Navigate to the SuiteScripts directory and check your file in the GettingStarted subdirectory.

Configure Suitelet Script Record and Deployment

25. In NetSuite, go to Setup > Customization > Scripts > New. The Select Type page loads.

26. In the Type list, click Suitelet.

27. In the Name field, enter GettingStarted_SS_HelloWorld.

28. In the ID field, enter _gs_ss_helloworld.

29. In the Scripts subtab, select GettingStarted_SS_HelloWorld.js in the Script File dropdown list.

30. In the Function field, enter processHelloWorld.

   Important: Make sure that you enter the function name without the parentheses and with the correct case. Also, the function name you enter MUST match the name of the executable function in your script.

31. Hover over the dropdown arrow on the Save button, and then click Save and Deploy. The New Script Deployment page loads.

32. In the Title field, enter GettingStarted_SS_HelloWorld.

33. In the ID field, enter _gs_ss_helloworld.

34. In the Status dropdown list, select Testing.
35. Click the Links subtab.
36. In the Center dropdown list, select Classic Center.
37. In the Section dropdown list, select Setup.
38. In the Category dropdown list, select Custom.
39. In the Label field, enter Getting Started.
40. Click Add.
41. Click Save.

Test Your Script


43. Click Click Me. The Hello World! popup message appears.

44. Click OK on the popup message.

Related Topics

- Working with Your SuiteScript Development Environment (SuiteCloud IDE)
- Working with IDEs Other Than SuiteCloud IDE
- Installing SuiteCloud IDE
- Launching SuiteCloud IDE
- Selecting a Workspace
- Setting a Master Password
- Authenticating a Master Password
- Adding an Account
Client SuiteScript Tutorials

These tutorials provide detailed examples for how to use client SuiteScript to control the behavior of item discounts in a custom Cash Sale Transaction Form.

The tutorials start by describing the code to be used when the page first loads, then covers field changes, and then validation and page save code. These tutorials are broken out into the following sections:

- Customizing the Page Load
- Validating Field Values
- Retrieving the Customer Discount
- Triggering Events When a Field is Changed
- Validating a Line Item on a Sublist
- Recalculating Field Totals
- Prompting Before Save
- Client Tutorial (Complete Code)

For each section, you may need to create custom elements described in the Setup sections. If you are familiar with creating custom items, lists and fields, simply refer to the table provided for the required parameters for each element. Otherwise, refer to the SuiteBuilder Guide in the NetSuite Help Center for detailed instructions. This guide is available in both Help and PDF formats.

The fields in these samples must exist in your NetSuite account prior to using these examples. If the items and fields do not exist, you will receive an error when attempting to use your custom cash sale.

Customizing the Page Load

[UE=HELP_TOPIC_OUT_OF_DATE_ALERT=UE]

In the Page Init example, we will cause a 5% transaction discount to be automatically applied to the cash sale price of an item whenever our custom Cash Sale Transaction form is loaded.

Setup

Before creating a script that applies a discount, we must first create a discount item in NetSuite and then use this discount item in our custom code.

<table>
<thead>
<tr>
<th>Element</th>
<th>Parameters</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discount Item</td>
<td>Item Name/Number: In-store Discount</td>
<td>Defines the discount item that you will refer to in your custom code.</td>
</tr>
<tr>
<td></td>
<td>Income Account: 4000 Sales</td>
<td>To create, go to Lists &gt; Accounting &gt; Items &gt; New &gt; Discount.</td>
</tr>
<tr>
<td></td>
<td>Rate: 5%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Apply Before Sales Tax: No</td>
<td></td>
</tr>
</tbody>
</table>

SuiteScript Developer and Reference Guide
Create the Code

Next we will create a javascript file called *customCashSales.js* and enter the following code. This code sets the discount item field to the value created in Setup.

```javascript
function samplePageInit() {
    nlapiSetFieldValue('discountitem', 144);
}
```

The code refers to the internal ID value of In-Store Discount item created in the preceding section. You can determine the discount item ID by navigating to the item, and inspecting the id= parameter of the URL or, if you have enabled the display of Internal IDs, the value is displayed in the internal ID column.

Test the Code

Next we will upload the javascript file to NetSuite and apply the code to our custom Cash Sale Transaction form.

**To upload the javascript file to NetSuite:**

1. Go to Documents > Files > File Cabinet.
2. Select the folder you want to add your file to.
3. At the bottom of the page, select Computer in the Attach From field.
4. Click Browse next to the File field and select the *customCashSale.js* file you just created.
5. Click Add This File.

**To apply the custom code to a transaction form:**

1. Go to Setup > Customization > Transaction Forms.
2. Click Customize next to Standard Cash Sale.
3. In the Name field, enter **Custom Code Sample**.
   Leave all standard settings the same.
4. Click the Custom Code subtab.
5. In the Script File field, browse to the **customCashSale.js** file you uploaded to your file cabinet.
6. In the Page Init Function field, enter **samplePageInit**.

![Custom Code](image)

7. Click Save.

You can now go to Transactions > Sales > Enter Cash Sales to test your custom code. Select the **Custom Code Sample** form in the Custom Form field on the Cash Sale page.

Notice that the Discount is automatically set to **In-Store Discount** with the rate at -5%. This cash sale will have the 5% discount applied to it.

![Custom Form](image)

### Validating Field Values

[UE=HELP_TOPIC_OUT_OF_DATE_ALERT=UE]

Of course, in some cases, there may be customer discounts already assigned for certain customers, in which case we wouldn't want to automatically assign the lower 5% discount — we will want to consider the customer discount instead. In this section we will create a validation script to ensure that this is the behavior.

#### Setup

Before creating our code, we will need to setup several customers with custom discounts. The following table lists all of the elements you will need to create in order to do this.
Create the Code

In this section we will add the validation code to the customCashSales.js file created for the Page Init function.

Note: When applying custom code, you can select up to two javascript files to associate with a given custom form. One file should contain all of the code specific to the current form. The other, library file, should contain any code that may be used across multiple forms. For the purposes of this tutorial, we will include all of the custom code in a single file — customCashSales.js.

Retrieving the Customer Discount

First, we’ll need to create a function that determines whether a customer discount rate has been set. The following code uses nlapiGetFieldText to retrieve the value of the custom transaction body field we created — custbody_cust_discount. If it exists, we convert the value to a negative...
number to ensure that the total is decreased by the specified amount and the value is then returned.

```javascript
function calcCustomerDiscountRate()
{
    if( !isNaN(parseFloat(nlapiGetFieldText('custbody_cust_discount'))))
        return ((parseFloat(nlapiGetFieldText('custbody_cust_discount')))) * -1;
    else
        return 0;
}
```

### Determining Which Discount to Use

Next, we'll want to make sure that the Page Init function does not cause the discount to be set to 5% if a customer already has a pre-defined higher discount rate.

1. Create a new function.
   ```javascript
   function sampleValidateField(type, name)
   {  
   
   }
   ```

2. Set the variables
   ```javascript
   var useCustomerDiscount = true;
   var instore_discount = -5;
   var cust_discount = calcCustomerDiscountRate();
   ```

3. Determine if In-Store discount is higher than the customer discount and return an alert that allows you to choose whether to accept the higher value.
   ```javascript
   if ( instore_discount <= cust_discount )
   {
       useCustomerDiscount = false;
   }
   if( useCustomerDiscount )
   {
       alert("custbody_cust_discount: " +
       nlapiGetFieldText('custbody_cust_discount') + ",");
   nlapiSetFieldText('discountitem',
   nlapiGetFieldText('custbody_cust_discount'));
   }
   else
   {
       nlapiSetFieldText('discountitem', 'In-store Discount');
   }
```

4. Close the function.

### Test the Code

Now upload the modified customCashSale.js file to the file cabinet and then assign the sampleValidateField function for the Validate Field function on the Custom Code Sample form.
Now when we enter a cash sale using the Custom Code Sample form, if a customer is selected that has a pre-defined discount higher than the In-store discount set by the Page Init function, then the In-store discount is NOT used.

**More Validation**

This is a very simple case. In reality, we'd need to add some further validation to ensure that the field types are in the correct format and we may want to check for other discount restrictions, etc. The complete code included in Client Tutorial (Complete Code) includes javascript validation for the following:

- If the transaction discount is changed to a rate greater than $50 or 30%, a box pops up asking for confirmation of manager approval for the discount.
  
  The user can choose to keep the current discount or abort the discount. If the discount is aborted, a message is displayed to notify the user that the discount level is not approved and it will be reset.

- If more than 20 of any one item is added to the transactions, a box pops up asking if you are sure you want to add the item. The user can choose to add the items or return to the item list without adding the items.

**Triggering Events When a Field is Changed**

In the Field Changed example, we will cause a popup box display that indicates if a customer has had order problems in the past and reminds the user to verify all items before submitting the order.

**Setup**

This script requires the creation of custom entity and transaction body fields.
Create the Code

In this section we will add the field changed code to the customCashSales.js file.

1. First, we will set this code to fire on any change to the transaction body field created during Setup.

   ```javascript
   function sampleFieldChanged(type, name)
   {
     if (name == 'custbody_had_order_problems')
     {
       Remember, the custbody_had_order_problems is sourced from the entity field selected. Therefore, this code will be executed each time a new customer is selected in the form. Since the custbody_had_order_problems field is locked, it can NOT be edited directly.
     }
   }
   return true;
   }
   
   Test the Code

   Upload the modified customCashSale.js file to the file cabinet and then assign the sampleFieldChanged function for the Field Change Validation function on the Custom Code Sample form.
Now when we enter a cash sale using the Custom Code Sample form, if a customer is selected where Past Order Problems checkbox is enabled, an alert is displayed.

### Validating a Line Item on a Sublist

In the Validate Line example, we will include validation that prevents the addition of a service line item without a service employee being chosen for that line.

#### Setup

This script requires the creation of a custom entity and custom transaction column field.

<table>
<thead>
<tr>
<th>Element</th>
<th>Parameters</th>
<th>Purpose</th>
</tr>
</thead>
</table>
| Entity Field             | Description: Service Rep  
Type: Check Box  
Subtab: Human Resources  
Applies To: Employee     | Provides a checkbox on employee records where you can define an employee as a service rep.  
To create, go to Setup > Customization > Entity Field > New. |
| Transaction Column Field | Description: Service Rep  
Type: List/Record  
List/Record: Employee  
ID: _service_rep  
Applies To: Sale Item  
Filter Using: Service Rep Is Checked: True   | To create, go to Setup > Customization > Transaction Column Field > New. |
| Employee Records         | Assign various employees as service reps.                                    |                                                                         |

#### Create the Code

Again, we will modify the customCashSale.js file to add the following code. The code simply checks the value of the custcol_service_rep custom field whenever a custcol_service_item item is selected. If the value is T, then an alert is displayed and the event is aborted. The user is forced to first select a service rep before entering any line items.
function sampleValidateLine(type) {
    if ( (nlapiGetCurrentLineItemValue('item', 'custcol_service_item') == true) &&
        (!nlapiGetCurrentLineItemText('item', 'custcol_service_rep')) )
    {
        alert("You must choose a Service Rep for this service item.");
        return false;
    }
    return true;
}

**Test the Code**

Upload the modified customCashSale.js file to the file cabinet and then assign the sampleValidateLine function for the Validate Line function on the Custom Code Sample form.

Now when we enter a cash sale using the Custom Code Sample form, if a service line item is selected when a sales rep has not been assigned, the user is forced to return and select a sales rep for the customer before selecting any service items.

**Recalculating Field Totals**

Now when we enter a cash sale using the Custom Code Sample form, if a service line item is selected when a sales rep has not been assigned, the user is forced to return and select a sales rep for the customer before selecting any service items.
Create the Code

Again, we will modify the customCashSale.js file to add the following code.

```javascript
function sampleRecalc(type)
{
    var total = 0;
    for (i = 1; i <= nlapiGetLineItemCount('item'); i++)
    {
        var item_amount = parseFloat(nlapiGetLineItemValue('item', 'amount', i));
        if (nlapiGetLineItemValue('item', 'custcol_service_item', i) == 'T')
        {
            total += item_amount;
        }
    }
    nlapiSetFieldValue('custbody_service_bookings', nlapiFormatCurrency(total));
}
```

Test the Code

Upload the modified customCashSale.js file to the file cabinet and then assign the sampleRecalc function for the Validate Line function on the Custom Code Sample form.
Now when we save a cash sale using the Custom Code Sample form, any service items amounts are added to the service bookings field.

**Prompting Before Save**

In the Save example, we will cause a pop-up box to display when a record is saved that prompts the user with “Are you sure you want to save this record?” The code will execute the save if the user clicks OK and abort the save if the user clicks Cancel.

**Note:** No record customization (setup) is required for this script to work properly.

**Create the Code**

Again, we will modify the customCashSale.js file to add the following code. This simple script returns a dialog box prompting the user to confirm the save.

```javascript
function sampleSaveRecord()
{
    return confirm("Are you sure you want to save this record?");
}
```

**Test the Code**

Upload the modified customCashSale.js file to the file cabinet and then assign the sampleSaveRecord function for the Save Record function on the Custom Code Sample form.

Now when we save a cash sale using the Custom Code Sample form, a confirmation prompt is returned.

**Client Tutorial (Complete Code)**

The following is the complete code used throughout the Client SuiteScript Tutorials exercise. Copy this information exactly as it appears here. Comments are included in the code to help illustrate the purpose of each portion of the code.
function samplePageInit()
{
    // Customers that make purchases in the store, as opposed to from a Web site,
    // automatically receive a 5% in-store transaction discount. This discount is
    // NOT in addition to other discounts they may have available to them.
    //
    // NOTE: you could also have used nlapiSetFieldText('discountitem', 'In-store
    // Discount');
    // The 92 is the ID of the item in your NetSuite account. You can determine the
    // ID of an item by navigating to the item, and inspecting the id= parameter of
    // the URL.

    nlapiSetFieldValue('discountitem', 92);
}

function sampleSaveRecord()
{
    return confirm("Are you sure you want to save this record?");
}

function sampleValidateField(type, name)
{
    // This script does not allow discounts equal to or greater than $50 or greater
    // than 30%, without manager approval.
    // If the transaction discount rate is positive, which adds to the order total,
    // prompt to reset the transaction discount to the better of the In-store or
    // Cust. Discount.
    //
    // Also, if the rate is a percent and is not as high as the Cust. Discount or
    // In-store Discount, check to see if they would like to reset the discount
    // to the higher rate.
    // The three variable initialization functions that are called from this
    // function (calcIsPercent, calcTransactionDiscountRate and
    // calcCustomerDiscountRate) are defined at the end of this script file.

    if (name == 'discountrate')
    {
        // initialize variables
        // Customers that purchase in the store, as opposed to from a Web site,
        // automatically receive a 5% in-store transaction discount.
        var instore_discount = -5;

        // Start by assuming the customer discount is better than the in-store
        // discount.
        var useCustomerDiscount = true;

        // Is the transaction discount field a percent?
        var isPercent = calcIsPercent();

        // If a transaction discount is present, set tran_discount_rate,
// otherwise initialize to zero.
var tran_discount_rate = calcTransactionDiscountRate();

// If a customer discount is present, set cust_discount,
// otherwise initialize to zero.
var cust_discount = calcCustomerDiscountRate();

// If the In-store discount is higher than the customer discount, use it
// instead.
if (instore_discount <= cust_discount )
  useCustomerDiscount = false;

// Check to see that the discount is not a percent.
if (!isPercent)
{
  // If the discount is not a percent...
  // Is the transaction discount rate > 0 (which adds to the order total)?
  // If so, prompt to reset Transaction Discount to the higher of In-house or
  // Cust. Discount
  if (tran_discount_rate > 0)
  {
    if (confirm("The Transaction Discount Rate is positive, which adds to
    the order total:\n\nReset Transaction Discount to the better of Cust.
    Discount or In-store Discount?\nOK = Yes, Cancel = No") )
    {
      // Use the higher of the In-store or the Cust. Discount rate.
      if( useCustomerDiscount )
      {
        // alert("custbody_cust_discount: "+
        nlapiGetFieldText('custbody_cust_discount') + "");
        nlapiSetFieldText('discountitem',
        nlapiGetFieldText('custbody_cust_discount'));
      }
      else
      {
        nlapiSetFieldText('discountitem', "In-store Discount");
      }
    }
  }
// If the discount is not percent, is the discount equal to or more than
// $50?
else if (parseInt(nlapiGetFieldValue('discountrate')) <= -50)
{
  if (!confirm("Please confirm that you have a manager’s approval for this
discount amount:\n\nOK = Yes, Cancel = No") )
  {
    // Use the higher of the In-store or the Cust. Discount rate
    if( useCustomerDiscount )
    {
      alert("Discount level not approved, resetting to Customer
      Discount.");
    }
    // alert("custbody_cust_discount: "+
    nlapiGetFieldText('custbody_cust_discount') + ");
  }
nlapiSetFieldText('discountitem',
nlapiGetFieldText('custbody_cust_discount'));
} else {
    alert("Discount level not approved, resetting to In-store Discount.");
    nlapiSetFieldText('discountitem', 'In-store Discount');
}
}

// If the discount is a percent...
// Is that percent more than 30% off?
else {
    if (parseFloat(nlapiGetFieldValue('discountrate')) < -30 ) {
        // If so, prompt...
        if (!confirm("Please confirm that you have a manager’s approval for this discount percent.\n\nOK = Yes, Cancel = No\n")) {
            // Reset to the higher of In-store discount or Cust. Discount.
            if (useCustomerDiscount) {
                alert("Discount level not approved, resetting to Customer Discount.");
                nlapiSetFieldText('discountitem',
nlapiGetFieldText('custbody_cust_discount'));
            } else {
                alert("Discount level not approved, resetting to In-store Discount.");
                nlapiSetFieldText('discountitem', 'In-store Discount');
            }
        }
    }
}

// If the Transaction Discount Rate is a percent and is lower than Cust.
// Discount or In-Store Discount, then prompt to reset Transaction Discount
// to the higher of the two.
if (tran_discount_rate > cust_discount || tran_discount_rate > instore_discount) {
    if (confirm("The Transaction Discount Rate is not as good as the Cust. Discount or the
    In-store Discount.\n\nReset Transaction Discount to the better of the two?\n\nOK = Yes, Cancel = No\n")) {
// Use the higher of the In-store or Cust. Discount rate.
if(useCustomerDiscount)
{
    // alert("custbody_cust_discount: " +
    nlapiGetFieldText('custbody_cust_discount') + "");
    nlapiSetFieldText('discountitem',
    nlapiGetFieldText('custbody_cust_discount'));
}
else
{
    nlapiSetFieldText('discountitem', "In-store Discount");
}

// If any single line-item quantity exceeds 20, ask if this is correct.
else if ((type == 'item') && (name == 'quantity'))
{
    var count = parseFloat(nlapiGetCurrentLineItemValue('item', 'quantity'));
    if(count > 20)
    {
        return confirm("Do you really want to add " + count + " of this item?
        OK = Yes,
        Cancel = No");
    }
}

// Always return true at this level to validate all the fields that you
// are not specifically validating...
return true;

//=====================================================}
function sampleFieldChanged(type, name)
{
    // A custom checkbox field for customers records whether the customer has "had
    // order problems."
    // A locked transaction body field (checkbox) sources the value from the
    // customer record onto the transaction.
    // On field change of customer (entity), check the sourced locked field. If it
    // is "T", then pop an alert saying
    // "This customer has had order problems previously. Be sure to verify all
    // items with the customer before
    // submitting the order."

    if (name == 'custbody_had_order_problems')
    {
// if customer is not null and this customer has had order problems,
// remind the employee to double check the order...
if ( (nlapiGetFieldValue('custbody_had_order_problems') == 'T') &&
    (nlapiGetFieldText('entity')) )
{
    alert("This customer has had order problems previously. Be sure to verify
    all items with
    the customer before submitting the order.");
    return true;
}

return true;

// sampleValidateLine(type)
function sampleValidateLine(type)
{
    // Prevents a service line item to be added without specifying the service rep
    // employee.

    // alert("service item flag: " + nlapiGetCurrentLineItemValue('item',
    // 'custcol_service_item'));
    // alert("service rep: " + nlapiGetCurrentLineItemText('item',
    // 'custcol_service_rep'));

    // if ( (nlapiGetCurrentLineItemValue('item', 'custcol_service_item') == 'T')
    // && (String(nlapiGetCurrentLineItemText('item', 'custcol_service_rep'))) ==
    // "")

    if ( (nlapiGetCurrentLineItemValue('item', 'custcol_service_item') == 'T') &&
         (!nlapiGetCurrentLineItemText('item', 'custcol_service_rep')) )
    {
        alert("You must choose a Service Rep for this service item.");
        return false;
    }

    return true;
}

// sampleRecalc(type)
function sampleRecalc(type)
{
    // For each service line item that is added, its total is added to the service
    // bookings field.

    // The custcol_service_item field sources its value from the Service Item
    // custom field checkbox on the item.
}
// initialize total
var total = 0;

// Run through each line item looking for service items.
for ( i = 1; i <= nlapiGetLineItemCount('item'); i++)
{
    // Set item_amount for the current item.
    var item_amount = parseFloat(nlapiGetLineItemValue('item', 'amount', i));

    // If the item is a service item, add its value to the total.
    if (nlapiGetLineItemValue('item', 'custcol_service_item', i) == 'T')
    {
        total += item_amount;
    }
}

// Set the service bookings custom field to the total of all service items.
nlapiSetFieldValue('custbody_service_bookings', nlapiFormatCurrency(total));

//=====================================================
// The following scripts are the three variable init functions called from the
// sampleValidateField function.
//
function calcIsPercent()
{
    // If you cannot find a percent sign in the discount field, set isPercent to
    // false.
    if( nlapiGetFieldValue('discountrate').indexOf('%') == -1 )
        return false;
    else
        return true;
}

function calcTransactionDiscountRate()
{
    // If a transaction discount is present, set tran_discount_rate,
    // otherwise initialize to zero.
    if( isNaN(parseFloat(nlapiGetFieldValue('discountrate'))) )
        return parseFloat(nlapiGetFieldValue('discountrate'));
    else
        return 0;
}
function calcCustomerDiscountRate()
{
    if( !isNaN(parseFloat(nlapiGetFieldText('custbody_cust_discount'))) )
    // Multiply the rate by -1 because all discounts must be negative or they
    // add to order total.
    return ((parseFloat(nlapiGetFieldText('custbody_cust_discount'))) * -1);
    else
    return 0;
}
//=====================================================

User Event SuiteScript Tutorial

The following use cases often depend on specific fields being available on the forms being used. If you are using custom forms, be sure that the fields that are referenced in the script are available in that form.

The following uses cases are covered in this section:

- User Event ERP Use Cases
- User Event CRM Use Cases

User Event ERP Use Cases

The following ERP use cases are outlined in this section:

- Email Notification
- Adding Contact Information to a Sales Order
- Modifying Line Items on a Transaction

Email Notification

This User Event script causes an email notification to be sent when a Sales Order is entered with a discount level higher than a given value. The maximum allowed discount level is set as a _maximumdiscountlevel parameter of the specific deployment.

The script uses the parameters outlined in the following table. Values for these parameters must be defined for each deployment of the script. To define these parameters, go to the Parameters tab of the script definition and create a New Field for each parameter. These parameter fields are essentially custom fields. For detailed information on how to define custom fields, refer to Creating a Custom Field.
**Note:** To enhance readability, it is recommended that you use an underbar in the field ID definitions as shown here since each script field created is automatically prefaced with `custscript`.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Field ID</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>_maximumdiscountlevel</td>
<td>custscript_maximumdiscountlevel</td>
<td>Percent</td>
</tr>
<tr>
<td>_salesorderapproveremail</td>
<td>custscript_salesorderapproveremail</td>
<td>E-mail Address</td>
</tr>
<tr>
<td>_salesordername</td>
<td>custscript_salesordername</td>
<td>Free-Form Text</td>
</tr>
<tr>
<td>_salesorderapproveremail2</td>
<td>custscript_salesorderapproveremail2</td>
<td>E-mail Address</td>
</tr>
</tbody>
</table>

In the following script, edit the nlapiSendEmail function to reflect a valid email in your account and valid recipients. The first argument (-5 in this sample) is the author of the email. The second argument (adminsToEmail) is a comma-delineated list of emails that can correspond to the internalId of any entity in the system or any other valid email address.
Script:

/* before Save trigger: test discount rate on new orders against cutoff rate and if it exceeds it then change the status to Pending approval. Also update memo */

function beforeSaveSalesOrder(type)
{
    var newRecord = nlapiGetNewRecord();
    var cutoffRate = custscript_maximumdiscountlevel;
    var discountRate = newRecord.getFieldValue('discountrate');
    if ( type == 'Create' && discountRate != null && discountRate.length > 0 && cutoffRate != null && cutoffRate.length > 0 )
    {
        discountRate = Math.abs( parseFloat( discountRate ) );
        cutoffRate = Math.abs( parseFloat( cutoffRate ) );
        if ( discountRate > cutoffRate )
        {
            newRecord.setFieldValue('orderstatus','A' /* Pending Approval */);
            newRecord.setFieldValue('memo','Changed status to pending approval because discount exceeded ' + custscript_maximumdiscountlevel);
        }
        else
        {
            newRecord.setFieldValue('orderstatus','B' /* Pending Fulfillment */);
            newRecord.setFieldValue('memo','Changed status to pending fulfillment since it did not exceed cutoff');
        }
    }
}

/* after Save trigger: test discount rate on new orders against cutoff rate and if it exceeds it then send an email to a predefined list of addresses. */
function afterSaveSalesOrder(type)
{
    var newRecord = nlapiGetNewRecord();
    var cutoffRate = nlapigetContext().getSetting('SCRIPT', custscript_maximumdiscountlevel);
    var discountRate = newRecord.getFieldValue('discountrate');
    if ( type == 'Create' && discountRate != null && discountRate.length > 0 &&
         cutoffRate != null && cutoffRate.length > 0 )
    {
        discountRate = Math.abs( parseFloat( discountRate ) );
        cutoffRate = Math.abs( parseFloat( cutoffRate ) );
        if ( discountRate > cutoffRate )
        {
            sendDiscountWarningEmail();
        }
    }
}

/* Lookup customer name by internalId */
function queryCustomerName( customer )
{
    var filters = new Array();
    filters[0] = new nlobjSearchFilter( 'internalid', null, 'is', customer, null );
    var columns = new Array();
    columns[0] = new nlobjSearchColumn( 'entityid' );
    var searchresults = nlapiSearchRecord( 'customer', null, filters, columns );
Adding Contact Information to a Sales Order

This User Event script sets a custom field on sales orders to the fax number of the customer's primary contact.

To use this script:

1. Create a custom transaction body field. Custom Transaction Body fields can be defined at Setup > Customization > Transaction Body Fields > New. The custom field should have the following characteristics:

```javascript
var entityid = searchresults[0].getValue('entityid');
return entityid;
}

/* Lookup salesrep name by internalId */
function querySalesRepName(salesrep) {
  var filters = new Array();
  filters[0] = new nlobjSearchFilter('internalid', null, 'equalTo', salesrep, null);
  var columns = new Array();
  columns[0] = new nlobjSearchColumn('entityid');
  var searchresults = nlapiSearchRecord('contact', null, filters, columns);
  var entityid = searchresults[0].getValue('entityid');
  return entityid;
}

/* Send pre-formatted email to predefined list of recipients */
function sendDiscountWarningEmail() {
  var newRecord = nlapiGetNewRecord();
  var customerName = queryCustomerName(newRecord.getFieldValue('entity'));
  var salesrepName = querySalesRepName(newRecord.getFieldValue('salesrep'));
  var orderName = custscript_salesordername != null ? custscript_salesordername : 'Sales Order';
  var str = salesrepName + ' has entered a ' + orderName + ' for ' + customerName + ' that exceeds ' + custscript_maximumdiscountlevel + '.

  str += 'Your approval is required before it can be fulfilled.

  str += 'Please log in to your NetSuite solution to approve the discount and order.

  var adminsToEmail = custscript_salesorderapproveremail;
  if ( custscript_salesorderapproveremail2 != null )
    adminsToEmail += '; ' + custscript_salesorderapproveremail2;

  /*nlapiSendEmail( -5 /* Joe Wolfe (admin)*/, adminsToEmail, 'Discount Warning', str ); */
  /* logging a note in the execution log to indicate success */
  nlapiLogExecution('DEBUG', 'Discount Warning');
}`
• ID = _contactfax
• Type = Phone Number
• Display Subtab = Main
• Applies To = Sale
• Store Value = False (deselect)

For detailed information on creating custom fields, see Creating a Custom Field in the SuiteBuilder (Customization) Guide, available in both Help and PDF formats.

2. Set the script to execute **Before Load** on Sales Order records.
Script:

```javascript
function beforeLoadRecord(type) {
    // only run this script for existing records
    if (type.toLowerCase() == 'create')
        return;
    // Get the current record
    var record = nlapiGetNewRecord();
    // Get the entity (customer) for this order
    var customer = record.getFieldValue('entity');
    // Execute a search to retrieve the primary contact's fax
    // Filter to custom id = the customer of the sales order gleaned above
    var filters = new Array();
    filters[0] = new nlobjSearchFilter('internalid', null, 'anyof ', customer);
    // Return the fax # from the primary contact joined search
    var results = new Array();
    results[0] = new nlobjSearchColumn('fax', 'contactprimary');
    // Execute the search
    var searchresults = nlapiSearchRecord('customer', null, filters, results);
    // Retrieve the result field
    var contactfax = searchresults[0].getValue('fax', 'contactprimary');
    // Set it into the record
    record.setFieldValue('custbody_contactfax', contactfax);
}
```

Modifying Line Items on a Transaction

This **User Event** script demonstrates how to modify each line item of a transaction. In this case, the Commit field on a Sales Order is set to **Do Not Commit**.

To use this script, set the script to execute Before Submit on a Sales Order record.

Script:

```javascript
function setLineField(type) {
    var currentRecord;
    var lines;
    var i;
    // Execute this only when Sales order is created or edited.
    if( (type == 'create') || (type=='edit') )
    {
        currentRecord = nlapiGetNewRecord();
        // Get the number of line items before submit
        lines = currentRecord.getLineItemCount('item');
        for (i = 1; i <= lines; i++)
        {
            currentRecord.setLineItemValue('item', 'commitinventory', i, '3');
        }
    }
}
```

User Event CRM Use Cases

The following CRM use cases are outlined in this section:
• Creating Tasks on Opportunity Creation
• Redirecting to a New Task Record
• Notification of Closed Cases
• Defaulting Case Fields Based on Incoming Email Address
• High Open Balance Dashboard Portlet

Creating Tasks on Opportunity Creation

This User Event script causes a Task to be automatically created for the Sales Rep when a new Opportunity record is created. If no Sales Rep is set for the Opportunity at the time it is created, the task is assigned to a predefined default Sales Rep. A notification email is also sent to inform the Sales Rep of the new Opportunity.

To use this script:

1. Change the salesrep variable to reflect an internal ID of a valid employee with the Sales Rep Role for your account. The Sales Role is set on the Human Resource tab of an Employee record and the Internal ID for employees can be seen on the Employee Record when the Show Internal IDs preference is enabled.

2. Edit the author ID in the nlapSendEmail function to reflect a valid email in your account.

3. Set the script to execute After Submit on Opportunity records.

Script:

```javascript
function taskCreator(type)
{
    var currentRecord;
    var recordCreated;
    var salesrep;
    var customer;
    var tranNum;
    var emailText = 'A new task has been assigned to you';
    // create Task only when a new Opportunity is created
    if ( type == 'create' )
    {
        // Get the Current Record
        currentRecord = nlapiGetNewRecord();
        // Get the Sales Rep on opportunity.
        salesrep = currentRecord.getFieldValue('salesrep');
        // Get the Customer from the opportunity.
        customer = currentRecord.getFieldValue('entity');
        // Get the opportunity Id being created
        tranNum = currentRecord.getId();
        // Set default sales rep if none on opportunity
        if (salesrep == '')
        {
            salesrep = 302;
        }
        // Create Task
        recordCreated = nlapiCreateRecord('task');
    }
```
Redirecting to a New Task Record

This User Event script is similar to the Creating Tasks on Opportunity Creation use case, except that after creating the task record pre-populated with information from the Opportunity record, the user is actually redirected to the new task record in edit mode.

To use this script, set the script to execute after submit on Opportunity records.

Script:

```javascript
function taskRedirect(type)
{
    var currentRecord;
    var recordCreated;
    var salesrep;
    var customer;
    var tranNum;
    var taskNum;

    // create Task only when a new Opportunity is created
    if ( type == 'create' )
    {
        // Get the Current Record
        currentRecord = nlapiGetNewRecord();
        // Get the Sales Rep on opportunity.
        salesrep = currentRecord.getFieldValue('salesrep');
        // Get the Customer from the opportunity.
        customer = currentRecord.getFieldValue('entity');
        // Get the opportunity Id being created
        tranNum = currentRecord.getId();
        // Create Task
        recordCreated = nlapiCreateRecord('task');
        // Set Task Title Title
        recordCreated.setFieldValue('title', 'Opportunity Follow-Up');
        // Set the sales rep
        recordCreated.setFieldValue('assigned', salesrep);
        // Set the company
        recordCreated.setFieldValue('company', customer);
        // Set the Customer
        recordCreated.setFieldValue('transaction', tranNum);
        // Save the Task record
        taskNum = nlapiSubmitRecord(recordCreated);
        // Redirect user to the task that was created in Edit mode.
        nlapiSetRedirectURL('RECORD', 'task', taskNum, true);
    }
}
```

```javascript
// Set Title, Assigned to, Message and Company
recordCreated.setFieldValue('title', 'Opportunity Follow-Up');
recordCreated.setFieldValue('assigned', salesrep);
recordCreated.setFieldValue('message', 'Follow up with your customer');
recordCreated.setFieldValue('company', customer);
recordCreated.setFieldValue('transaction', tranNum);
nlapiSubmitRecord(recordCreated, true);
nlapiSendEmail(-5, salesrep, 'Task Creating Email Notification', emailText, null);
```
Notification of Closed Cases

This User Event script causes an email to be sent whenever the status of a case is changed to Closed. The email is sent to the all emails listed in the E-mail(s) field on the Case record — which normally defaults to the email of the currently selected company.

**Note:** Any emails selected in the Email Employees multi-select field are not emailed this notification using this script.

**To use this script:**

1. Set the script to execute after submit on Case records
2. Edit the nlapiSendEmail author ID (-5) to a valid ID from your account

**Script:**

```javascript
function sendCaseEmail()
{
    var currentRecord;
    var status;
    var emailaddress;
    var emailMessage = 'Dear Customer - Your case has been resolved and is now closed';
    var emailSubject = 'Case Status Notification';
    currentRecord= nlapiGetNewRecord();
    // Get the value of the Status
    status = currentRecord.getFieldValue('status');
    // check if status is closed
    if ( status == '5')
    {
        // Get email address from the case
        emailaddress = currentRecord.getFieldValue('email');
        if (emailaddress != '')
        {
            nlapiSendEmail( -5, emailaddress, emailSubject, emailMessage, null);
        }
    }
}
```

Defaulting Case Fields Based on Incoming Email Address

This User Event script causes a custom field, custevent_inboundmemo, on a case record to populate with a predetermined value based on the Inbound Email field when a case is submitted via email.

**To use this script:**

1. Create a custom CRM field. Custom CRM fields can be defined at Setup > Customization > CRM Fields > New. The custom field should have the following characteristics:
   - ID = _inboundmemo
- Type = Freeform Text
- Display Subtab = Main
- Applies To = Case
- Store Value = False (deselect)

For detailed information on creating custom fields, see Creating a Custom Field in the SuiteBuilder (Customization) Guide, available in both Help and PDF formats.

2. Set the script to execute **Before Submit** on Case records.

**Script**

```javascript
function beforeSubmit(type) {
    var record = nlapiGetNewRecord();
    if (type == 'create') {
        var inboundemail = record.getFieldValue('inboundemail');
        if (inboundemail != null) {
            if (inboundemail == 'info@rippleit.com') {
                record.setFieldValue('custevent_inboundmemo','INFO');
            } else if (inboundemail == 'demo@rippleit.com') {
                record.setFieldValue('custevent_inboundmemo','DEMO');
            } else {
                record.setFieldValue('custevent_inboundmemo','SUPPORT');
            }
        }
    }
}
```

**High Open Balance Dashboard Portlet**

This **Portlet** script searches for all Customers with an open balance greater than $5,000.00 for a specific sales rep. Once the script is defined, the SuiteScript Portlet preference must be enabled on the dashboard and this script selected to display on the dashboard.

**Script:**

```javascript
function highOpenBalance(portlet) {
    portlet.setTitle( 'Customers with High Open Balances' );
    portlet.addLine( 'Customers with Open Balances greater than <b><i>$5,000.0</i></b>',null,0);
    var filters = new Array();
```
filters[0] = new nlobjSearchFilter( 'salesrep', null, 'anyOf', 164, null );
filters[1] = new nlobjSearchFilter( 'balance', null, 'greaterThan', 5000, null );

// -- return opportunity sales rep, customer custom field, and customer ID
var columns = new Array();
columns[0] = new nlobjSearchColumn( 'balance' );
columns[1] = new nlobjSearchColumn( 'entityid' );
columns[2] = new nlobjSearchColumn( 'email' );
columns[3] = new nlobjSearchColumn( 'phone' );
var searchresults = nlapiSearchRecord( 'customer', 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 entityid = searchresult.getValue( 'entityid' );
  var balance = searchresult.getValue( 'balance' );
  var email = searchresult.getValue( 'email' );
  var phone = searchresult.getValue( 'phone' );
  portal.addLine( entityid , nlapiResolveURL('RECORD', 'customer', record, null), 0 );
  portal.addLine( '<font style="font-size:8pt">Balance: ' + balance +
    ' E-mail: ' + email + ' Phone: ' + phone + '</font>' , null, 2 );
}
Part 13  SuiteScript
Best Practices
Chapter 88 General Development Guidelines

This section provides general guidelines for working with SuiteScript.

- Always thoroughly test your code before using on your live NetSuite data.
- All record, field, sublist, tab, and subtab IDs must be in lowercase in your SuiteScript code.
- It is recommended that you prefix all custom script IDs and deployment IDs with an underscore (_).
- If the same code is being used across multiple forms, ensure that you test any changes in the code for each form that the code is associated with.
- Wherever data may be inconsistent, not available, or invalid for certain functions, ensure that you include proper error handling sequences in your script. For example, if your script requires a field value in order 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.
- Use the built in Library functions whenever possible for reading/writing Date/Currency fields and for querying XML documents.
- During script development, it may be useful for testing purposes to componentize your scripts, load them individually and then test them one by one -- inactivating all but the one you are testing when multiple components are tied to a single user event.
- 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 formAPageInit. If your code is already established, you can simply wrap it with a top-level function that has the appropriate naming convention.
• You must use the nllobContext.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 nllobjContext.

• As with any script, ensure that you thoroughly comment your code. This will not only help with debugging and development but will assist NetSuite support in locating problems if necessary.

### Related Topics
- Suitelets and UI Object Best Practices
- User Event Best Practices
- Scheduled Script Best Practices
- Client Script Best Practices
- Script Optimization
Chapter 89 Suitelets and UI Object Best Practices

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.

Related Topics
- Suitelets
- Suitelet Script Execution
- UI Objects Overview
Chapter 90 User Event Best Practices

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 in-line.
- 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.

Related Topics

- User Event Scripts
- SuiteScript Execution Diagram
- User Event Script Execution Types
- SuiteScript API Overview
Chapter 91 Scheduled Script Best Practices

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 bottleneck 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 Admin = Yes
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.

**Related Topics**
- Scheduled Scripts
- Understanding Scheduled Script Execution
- Deploying Scheduled Scripts
Chapter 92 Client Script Best Practices

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 xxxValue functions because you are setting an internal ID, which is ultimately a more precise definition. The xxxText functions perform searches that may return duplicates or no results, which effectively disables your script.

Related Topics
- Client Scripts
- Client Event Functions
- Form-level and Record-level Client Scripts
- SuiteScript API Overview
Chapter 93 Security Considerations

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).

<table>
<thead>
<tr>
<th>Field Internal ID</th>
<th>Field UI Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>Password</td>
</tr>
<tr>
<td>password2</td>
<td></td>
</tr>
<tr>
<td>ccnumber</td>
<td>Credit Card Number</td>
</tr>
<tr>
<td>ccsecuritycode</td>
<td>CSC Code</td>
</tr>
</tbody>
</table>

Related Topics
- User Event Scripts
- SuiteScript API Overview
Chapter 94 Script Optimization

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 as Admin.)