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

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

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

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

Sample Code

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

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

No Excessive Use of the Service

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

Beta Features

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

Custom GL Lines Plug-in Overview ............................................................................................................ 1
Custom GL Lines Plug-in Process Flow .......................................................................................................... 2
Transaction Types Supported By Custom GL Lines Plug-in ............................................................................... 4
Custom GL Lines Plug-in Implementation Development .................................................................................. 7
  - Gather Required Internal IDs .................................................................................................................. 8
  - Create a Plug-in Implementation Script File .......................................................................................... 8
  - Add the Plug-in Implementation ............................................................................................................ 11
  - Test the Plug-in Implementation .......................................................................................................... 12
  - Bundle the Plug-in Implementation ...................................................................................................... 13
Custom GL Lines Plug-in Implementation Administration .............................................................................. 13
  - Enable Features for a Custom GL Lines Plug-in Implementation .......................................................... 14
  - Install a Custom GL Lines Plug-in Bundle ............................................................................................ 14
  - Configure the Custom GL Lines Plug-in Implementation ....................................................................... 15
  - Enable the Custom GL Lines Plug-in Implementation ........................................................................... 16
  - Custom GL Lines Plug-in Implementation System Notes ...................................................................... 17
Custom GL Lines Plug-in Interface Definition .............................................................................................. 19
  - customizeGlImpact(transactionRecord, standardLines, customLines, book) ................................................. 20
    - StandardLines .................................................................................................................................. 20
    - CustomLines ................................................................................................................................... 29
    - AccountingBook .............................................................................................................................. 41
    - Record ............................................................................................................................................ 42
  - Common StandardLine and CustomLine Object Methods ........................................................................ 43
Custom GL Lines Plug-in Reference ............................................................................................................. 51
  - Custom GL Lines Plug-in Example Code ............................................................................................... 51
    - Utility Files for a Custom GL Lines Plug-In Implementation ......................................................................... 52
    - Using a Custom Record to Reference Internal NetSuite IDs Example .................................................... 54
Classifications ............................................................................................................................................... 55
  - Classification Handling on Standard Transactions ............................................................................. 55
  - Classification Handling on Journals .................................................................................................... 56
  - Classification Handling on Custom Transactions ................................................................................... 57
Multiple Currencies Examples ....................................................................................................................... 57
  - Find the Base Currency ....................................................................................................................... 58
  - Base Currency from Accounting Book Record ..................................................................................... 59
  - Transaction-Specific Exchange Rates .................................................................................................... 59
  - Convert Currencies with SuiteScript .................................................................................................... 60
Custom GL Lines Plug-in Guidelines and Best Practices ............................................................................... 61
Error Messages for Custom GL Lines Plug-in ............................................................................................... 63
Custom Lines on Reports and in Searches ..................................................................................................... 65
  - Custom GL Lines on NetSuite Reports ................................................................................................ 65
  - Searches for Custom GL Lines .............................................................................................................. 66
Custom GL Lines Audit Log ........................................................................................................................... 67
  - Accessing the Custom GL Lines Audit Log ............................................................................................ 67
  - Custom GL Lines Audit Log Page ......................................................................................................... 68
Reviewing Custom GL Lines Plug-in Executions ........................................................................................... 72
Deleting a Custom GL Lines Plug-in Implementation .................................................................................. 73
Custom GL Lines Plug-in Overview

The Custom GL Lines Plug-in modifies the general ledger impact of standard and custom transactions. Use the Custom GL Lines Plug-in to comply with a wide range of global accounting standards by applying custom transaction logic that adds lines to the NetSuite GL Impact page.

The Custom GL Lines Plug-in can set and read custom segment values from custom and standard lines, and from transaction records. The default values for column segments are sourced from the body. The Custom GL Lines Plug-in can change this value, even for segments that are applied only to the body.

Solution providers can create implementations of the plug-in to add GL lines to transactions. These plug-in implementations can be bundled and distributed as SuiteApps. NetSuite account administrators can install the SuiteApp and configure custom GL plug-in implementations to be applied to specific transaction types, subsidiaries, and accounting books. OneWorld accounts can set a plug-in implementation to run for specific subsidiaries, and accounts with the Multi-Book Accounting feature enabled can set a plug-in implementation to run for specific accounting books.

For example, a solution provider can create a plug-in implementation to capture locale-specific GL impact on standard transaction types. Plug-in implementations could also add bank, credit card, and wire charges to payments, cash sales, and refunds or create unique discount or markup rules that add an amount to an invoice or bill.

The following screenshot shows the GL Impact page for an invoice with custom lines created by the Tax Expense On Accruals plug-in implementation:

![GL Impact screenshot](image)

In the preceding screenshot, the Custom Script column links to the plug-in implementation that created the custom line. In addition, because the custom lines created by a plug-in implementation are linked to the original transaction, the custom lines can appear in reports and searches. See Custom Lines on Reports and in Searches.

For information about the Custom GL Lines plug-in, see the following topics:

<table>
<thead>
<tr>
<th>NetSuite Role</th>
<th>For more information, see ...</th>
</tr>
</thead>
</table>
| All roles     | Custom GL Lines Plug-in Process Flow  
Transaction Types Supported By Custom GL Lines Plug-in |
| Developer     | Custom GL Lines Plug-in Implementation Development |
| Administrator | Custom GL Lines Plug-in Implementation Administration |
Important: The Custom GL Lines plug-in does not currently support SuiteScript 2.0. Only SuiteScript 1.0 is supported.

Custom GL Lines Plug-in Process Flow

A Custom GL Lines plug-in implementation executes depending on how the plug-in implementation is configured. The implementation executes when you save a transaction of the type and subsidiary configured for the plug-in implementation. The subsidiary is the subsidiary to which the transaction belongs. The plug-in implementation may also execute multiple times, one time for each accounting book configured for the plug-in implementation.

For example, the following plug-in implementation is configured to run when you save an invoice for the Parent Company or Czech Subsidiary, and runs on both the primary and secondary books.

The following diagram shows the execution model of the Custom GL Lines plug-in implementation:
The plug-in implementation executes according to the following process:

1. User saves a transaction for the entity belonging to a subsidiary. Using the plug-in implementation configuration, any configured transaction type with an entity associated with a configured subsidiary triggers the plug-in implementation execution on the transaction. The plug-in can run either during transaction save or asynchronously, depending on the Asynchronous box on the plug-in configuration page. The subsidiary is the main subsidiary for the transaction, or the subsidiary to which the transaction belongs.

   NetSuite first creates the standard lines for the primary book.

   NetSuite creates any custom lines in the primary book configured for the plug-in implementation. If you do not use the Multi-Book Accounting feature, the plug-in implementation process completes. Otherwise, NetSuite continues with the mapping of standard and custom lines to the secondary books.

3. NetSuite determines which lines should be applied to any secondary books. By default, NetSuite maps standard lines to the appropriate secondary accounting books, according to any mapping rules set up for the Multi-Book Accounting feature. NetSuite performs the same mapping for all custom lines with the `setBookSpecific(bookSpecific)` method set to false.

4. NetSuite creates standard lines. NetSuite creates any of the appropriate mapped standard lines and any custom lines that are not specific to the primary book in the secondary accounting books.


   NetSuite creates the custom lines on the secondary accounting book.

6. Transaction save completes. You can view the general ledger impact of the transaction by selecting Actions > GL Impact. If an asynchronous implementation is used and the plug-in is still pending, an information message appears on the GL Impact page. Custom GL Lines from asynchronous plug-ins are shown after the plug-in is executed. When costing updates are performed (for example, the COGS update), the plug-in might run asynchronously, and the values on the GL Impact page might change after the update is completed.

   **Important:** The current version of the Custom GL Lines Plug-in is always used.

### Transaction Types Supported By Custom GL Lines Plug-in

The following table lists the transactions supported by the Custom GL Lines Plug-in.

<table>
<thead>
<tr>
<th>Transaction Name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Build</td>
<td>assemblybuild</td>
</tr>
<tr>
<td>Assembly Unbuild</td>
<td>assemblyunbuild</td>
</tr>
<tr>
<td>Bill</td>
<td>vendorbill</td>
</tr>
<tr>
<td>Bill Credit</td>
<td>vendorcredit</td>
</tr>
<tr>
<td>Bill Payment</td>
<td>vendorpayment</td>
</tr>
<tr>
<td>Blanket Purchase Order</td>
<td>blanketpurchaseorder</td>
</tr>
<tr>
<td>Cash Refund</td>
<td>cashrefund</td>
</tr>
<tr>
<td>Cash Sale</td>
<td>cashsale</td>
</tr>
<tr>
<td>Transaction Name</td>
<td>ID</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Check</td>
<td>check</td>
</tr>
<tr>
<td>Credit Memo</td>
<td>creditmemo</td>
</tr>
<tr>
<td>Customer Deposit</td>
<td>customerdeposit</td>
</tr>
<tr>
<td>Customer Payment</td>
<td>customerpayment</td>
</tr>
<tr>
<td>Customer Refund</td>
<td>customerrefund</td>
</tr>
<tr>
<td>Deposit</td>
<td>deposit</td>
</tr>
<tr>
<td>Deposit Application</td>
<td>depositapplication</td>
</tr>
<tr>
<td>Expense Report</td>
<td>expensereport</td>
</tr>
<tr>
<td>Finance Charge *</td>
<td>invoice</td>
</tr>
<tr>
<td>Inventory Adjustment</td>
<td>inventoryadjustment</td>
</tr>
<tr>
<td>Invoice *</td>
<td>invoice</td>
</tr>
<tr>
<td>Item Fulfillment</td>
<td>itemfulfillment</td>
</tr>
<tr>
<td>Item Receipt</td>
<td>itemreceipt</td>
</tr>
<tr>
<td>Journal *</td>
<td>journalentry</td>
</tr>
<tr>
<td>Opportunity</td>
<td>opportunity</td>
</tr>
<tr>
<td>Paycheck Journal</td>
<td>paycheckjournal</td>
</tr>
<tr>
<td>Purchase Contract</td>
<td>purchasecontract</td>
</tr>
<tr>
<td>Purchase Order</td>
<td>purchaseorder</td>
</tr>
<tr>
<td>Return Authorization</td>
<td>returnauthorization</td>
</tr>
<tr>
<td>Revenue Commitment *</td>
<td>revenuecommitment</td>
</tr>
<tr>
<td>Revenue Commitment Reversal</td>
<td>revenuecommitmentreversal</td>
</tr>
<tr>
<td>Sales Order *</td>
<td>salesorder</td>
</tr>
<tr>
<td>Store Pickup Fulfillment</td>
<td>storepickupfulfillment</td>
</tr>
<tr>
<td>Vendor Return Authorization</td>
<td>vendorreturnauthorization</td>
</tr>
<tr>
<td>Work Order</td>
<td>workorder</td>
</tr>
<tr>
<td>Work Order Close</td>
<td>workorderclose</td>
</tr>
<tr>
<td>Work Order Completion</td>
<td>workordercompletion</td>
</tr>
<tr>
<td>Work Order Issue</td>
<td>workorderissue</td>
</tr>
</tbody>
</table>

* See Transaction Support Exceptions.

**Transaction Support Exceptions**

The following transactions are supported with restrictions:
- **Finance Charge.** A special type of invoice, the plug-in supports this transaction type when created through an invoice form.

- **Journal.** The plug-in supports journals except in the following cases:
  - Book-specific journal entries
  - Voiding journal and reversing journal
    These journals are unsupported because they negate the original transaction and should not create new GL impact
  - Open balance journal
  - Statistical journal
  - Intercompany journal
  - Intercompany elimination journal
  - Allocation journal
  - Revaluation journals
  - Collect Tegata
  - Pay Tegata
  - Revenue recognition
  - Amortization
  - Revenue reclassification
  - Revenue arrangements
  - Time posting to journal entry
  - Bill variances journal entry
  - Recognize gift certificate income
  - Allocate paycheck to jobs
  - Payroll: Create funding transaction
  - Payroll: YTD process
  - Payroll batch journal
  - Paycheck journal

- **Revenue Commitment.** A plug-in implementation does not execute on book-specific revenue commitments.

- **Sales Order.** A plug-in implementation does not execute on book-specific sales order.

**Transaction Processing**

An implementation of the plug-in processes transactions by the following methods:

- **Voided transactions.** Voided transactions with custom lines are handled in the same way as voided transaction with standard GL impact lines. See the help topic Voiding, Deleting, or Closing Transactions.

- **Memorized transactions.** When the transaction is entered into the system, the plug-in implementation creates custom lines. The transaction is also set to Posting or Non-Posting, matching the value for the standard lines.

- **Due date.** Due date is set on custom lines when the transaction is saved. The due date is equal to the transaction date. This enables the custom line to appear on aging reports.
Custom GL Lines Plug-in Implementation Development

You can develop a Custom GL Lines plug-in implementation for each use case where you want to modify the general ledger entries for transactions. A single plug-in implementation can process multiple types of transactions or you can create a separate plug-in implementation for each transaction type.

To create a plug-in implementation, use a developer account to develop and test the plug-in implementation. Then, use SuiteBundler to bundle the plug-in objects and distribute them to other NetSuite accounts. NetSuite administrators use the bundle to install the plug-in implementation in a NetSuite account and enable the implementation. For more information about administration tasks for a Custom GL Lines Plug-in bundle, see Custom GL Lines Plug-in Implementation Administration.

**Note:** If you use the Custom GL Lines Plug-in with Custom Segments, any segments affected by the implementation must be flagged as having GL impact. When the plug-in uses a custom segment that is not marked as having a GL impact, you will see an error message.

The functionality that you include in a Custom GL Lines plug-in implementation depends on the type of accounting features used in the NetSuite account where the plug-in implementation runs and the following properties that you configure for each plug-in implementation:

- **Transaction types.** A Custom GL Lines plug-in implementation runs when you save a transaction for which the plug-in implementation is configured to run, where the entity for the transaction is associated with a specific subsidiary. You can include code in the plug-in implementation script file that applies only to specific transaction types.
- **Subsidiaries.** In a One World account, the plug-in implementation only affects the general ledger entries for the subsidiaries that you configure for the plug-in implementation.
- **Accounting books.** The plug-in implementation affects only the accounting books that you configure for the plug-in implementation. If you have the Multi-Book Accounting feature enabled, you can direct the plug-in implementation to modify existing or add custom general ledger entries in multiple accounting books. If you do not have the Multi-Book Accounting feature enabled, the plug-in implementation affects only the general ledger entries in the primary accounting book.

**Note:** You can have multiple Custom GL Lines plug-in implementations enabled in an account. If you enable a plug-in implementation that conflicts with an existing implementation, NetSuite displays a warning message that several implementations for one transaction can cause longer save times. You are allowed to save the implementation, however if you notice long transaction save times, you can resolve this by setting one or more plug-in implementations to run asynchronously.

For more information about how a Custom GL Lines plug-in implementation modifies the general ledger impact for a transaction, see Custom GL Lines Plug-in Process Flow.

The following table describes the basic steps in developing a single plug-in implementation of the Custom GL Lines plug-in:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable features</td>
<td>Enable the Custom GL Lines Plug-in and Server SuiteScript features.</td>
</tr>
<tr>
<td></td>
<td>See Enable Features for a Custom GL Lines Plug-in Implementation.</td>
</tr>
<tr>
<td>Gather required internal IDs</td>
<td>In a plug-in implementation, you can reference internal NetSuite IDs for records and accounts. Before you create the plug-in implementation script file, you must note these values.</td>
</tr>
</tbody>
</table>
### Custom GL Lines Plug-in Implementation Development

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Create script file</strong></td>
<td>Create the script file that contains the Custom GL Lines plug-in implementation. See <a href="#">Create a Plug-in Implementation Script File</a>.</td>
</tr>
<tr>
<td><strong>Add the plug-in implementation</strong></td>
<td>Add the plug-in implementation using the plug-in script file and any required utility files that you created in the previous step to the development account. See <a href="#">Add the Plug-in Implementation</a>.</td>
</tr>
<tr>
<td><strong>Test the plug-in implementation</strong></td>
<td>Verify the behavior of the plug-in implementation. See <a href="#">Test the Plug-in Implementation</a>.</td>
</tr>
<tr>
<td><strong>Bundle the plug-in implementation</strong></td>
<td>Bundle both the plug-in implementation and other objects required by the plug-in implementation for distribution to other NetSuite accounts. See <a href="#">Bundle the Plug-in Implementation</a>.</td>
</tr>
</tbody>
</table>

### Gather Required Internal IDs

When you create a Custom GL Lines plug-in implementation, you might refer to records in NetSuite by their internal NetSuite IDs.

The following table describes the required NetSuite internal IDs if you want to use the records in the plug-in implementation script file:

<table>
<thead>
<tr>
<th>Record Type</th>
<th>Location of ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounts</td>
<td>Setup &gt; Accounting &gt; Chart of Accounts</td>
</tr>
<tr>
<td>Accounting books</td>
<td>Setup &gt; Accounting &gt; Accounting Books</td>
</tr>
<tr>
<td>Departments</td>
<td>Setup &gt; Company &gt; Departments</td>
</tr>
<tr>
<td>Classes</td>
<td>Setup &gt; Company &gt; Classes</td>
</tr>
<tr>
<td>Locations</td>
<td>Setup &gt; Company &gt; Locations</td>
</tr>
<tr>
<td>Subsidiaries</td>
<td>Setup &gt; Company &gt; Subsidiaries</td>
</tr>
<tr>
<td>Tax codes</td>
<td>Setup &gt; Accounting &gt; Tax Codes</td>
</tr>
<tr>
<td>Tax types</td>
<td>Setup &gt; Accounting &gt; Tax Types</td>
</tr>
</tbody>
</table>

**Note:** You can use helper functions in a utility file to avoid adding the internal NetSuite IDs into the plug-in implementation logic. See [Utility Files for a Custom GL Lines Plug-In Implementation](#).

### Create a Plug-in Implementation Script File

You can use the SuiteCloud IDE, or another JavaScript IDE or text editor, to create a JavaScript file that includes the business logic for your Custom GL Lines plug-in implementation script file.

The following table describes the functions and objects available in a Custom GL Lines plug-in implementation script file:
<table>
<thead>
<tr>
<th>Function / Interface Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customizeGlImpact(transactionRecord, standardLines, customLines, book)</td>
<td>You must create an implementation of this function in the plug-in implementation script file. The script file can contain other helper functions. NetSuite executes this function when you save a transaction of the type and subsidiary configured for the plug-in implementation. NetSuite executes this function one time for the primary accounting book. If you use the Multi-Book Accounting feature, this function is also called for each secondary accounting book that you configure for the plug-in implementation.</td>
</tr>
<tr>
<td>StandardLines</td>
<td>Contains an array of all standard lines with GL impact in a transaction as StandardLine objects. Use the methods available to this object to access the properties of individual standard lines with GL impact on a transaction.</td>
</tr>
<tr>
<td>CustomLines</td>
<td>Contains an array of all custom lines with GL impact in a transaction as CustomLine objects. Use the methods available to this object to add and modify custom lines with GL impact on a transaction.</td>
</tr>
<tr>
<td>AccountingBook</td>
<td>Represents the accounting book passed to a Custom GL plug-in implementation when you save a transaction. Use the methods available to the book object to determine if the book is a primary or secondary book or get the internal NetSuite ID of the accounting book.</td>
</tr>
<tr>
<td>Record</td>
<td>Use this to access properties of the transaction with SuiteScript API nlobjRecord functions. You cannot modify the transaction.</td>
</tr>
</tbody>
</table>

For more information, see the Custom GL Lines Plug-in Process Flow and Custom GL Lines Plug-in Implementation Object Model.

**Custom GL Lines Plug-in Implementation Object Model**

The following diagram shows the object model for the interface input and output objects:
Rules and Guidelines

Use the following rules and guidelines when creating the plug-in implementation script file:

- The plug-in script file can have any name, as long as it contains an implementation of the interface function.
- If you want to create utility files with helper functions for the main implementation file, you can include those files when you create the plug-in implementation for the Custom GL Lines plug-in in NetSuite. For more information about utility files, see Utility Files for a Custom GL Lines Plug-In Implementation.
- You can use a custom record type to store the references to the required NetSuite standard records. See Using a Custom Record to Reference Internal NetSuite IDs Example.
- You can use SuiteScript API functions in a plug-in implementation, for example, `nlapiLoadRecord(type, id, initializeValues)` and `nlapiSearchRecord(type, id, filters, columns)`. However, using these APIs may negatively affect the performance of the plug-in implementation. Limit the use of these APIs, where possible, to improve performance of the plug-in implementation. In general, searching for records yields better performance than loading NetSuite records. Governance limits also apply to these functions. The plug-in script file allows up to 1000 usage units when used with SuiteScript. For more information, see the help topic SuiteScript 1.0 API Governance.
Add the Plug-in Implementation

When you have finished creating the plug-in implementation script file, create a new plug-in implementation. When you create the plug-in implementation, you upload the script file and other utility files as required.

To add the script files and create the plug-in implementation:

2. In the Script File field, open the script file or add a new file.
3. Click Create Plug-in Implementation.

4. On the Select Plug-in Type page, click the Custom GL Lines Plug-in link.
5. On the New Plug-in Implementation page, enter the following information:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>User-friendly name for the implementation. The plug-in implementation appears in the following locations:</td>
</tr>
<tr>
<td></td>
<td>- Manage Plug-ins page. Page used by administrators to enable/disable the plug-in implementation in their account.</td>
</tr>
<tr>
<td></td>
<td>- Bundle Builder. Select this name in the Bundle Builder to distribute the plug-in implementation to other accounts.</td>
</tr>
<tr>
<td></td>
<td>- GL Impact page. For each custom lines added by the plug-in implementation, the Custom Script column on the GL Impact page lists the plug-in implementation name.</td>
</tr>
<tr>
<td>ID</td>
<td>Internal ID for the implementation for use in scripting. If you do not provide an ID, NetSuite will provide one for you when you click Save.</td>
</tr>
<tr>
<td>Status</td>
<td>Current status for the implementation. Choose Testing to have the implementation accessible only to the owner of the implementation. Choose Released to have the implementation accessible to all users in the account.</td>
</tr>
<tr>
<td>Log Level</td>
<td>Logging level you want for the script. Select Debug, Audit, Error, or Emergency. These messages appear on the Execution Log subtab for the plug-in implementation.</td>
</tr>
<tr>
<td>Execute As Role</td>
<td>Role that the script runs as. The Execute As Role field provides role-based granularity in terms of the permissions and restrictions of the executing script. The Current Role value indicates that the script executes with the permissions of the currently logged-in NetSuite user.</td>
</tr>
</tbody>
</table>
Option | Description
--- | ---
**Note:** You can create the custom role during testing to test the plug-in implementation with the proper role. The role requires the SuiteScript permission. You can then bundle the custom role to distribute it with the plug-in implementation. See Test the Plug-in Implementation and Bundle the Plug-in Implementation.

Description | Optional description of the implementation. The description appears for the implementation on the Plug-In Implementations page.
Owner | User account that owns the implementation. Default is the name of the logged in user.
Inactive | Indicates the plug-in implementation does not run in the account. Inactivate a plug-in implementation, for example, to temporarily disable it for testing purposes.

6. On the Scripts subtab, from the **Implementation** list, select the script file that contains the implementation of the plug-in.

7. On the Scripts subtab, from the **Library Script File** list, select any utility script files required by the plug-in script file.

8. On the Unhandled Errors subtab, select which individuals will be notified if script errors occur. By default the **Notify Script Owner** box is checked.

   To enter multiple email addresses in the **Notify Emails** box, separate email addresses with a semi-colon.

9. Click **Save**. You can access the list of implementations by going to Customization > Plug-ins > Plug-in Implementations.

Before you can test the plug-in implementation, configure the plug-in implementation and then enable it. See Configure the Custom GL Lines Plug-in Implementation and Enable the Custom GL Lines Plug-in Implementation.

Test the Plug-in Implementation

To test a Custom GL Lines plug-in implementation, perform the following tasks:

- Create a custom role for the plug-in implementation. The **Execute As Role** field determines the role under which the plug-in implementation script runs. This role requires the SuiteScript permission.

  For more information about the **Execute As Role** field, see Add the Plug-in Implementation. For more information about custom roles in NetSuite, see the help topic Customizing or Creating NetSuite Roles.

- Configure the transaction types, subsidiaries, and accounting books for the plug-in implementation, depending on the accounting features. See Configure the Custom GL Lines Plug-in Implementation.

- Enable the plug-in implementation at Customization > Plug-ins > Manage Plug-ins. See Enable the Custom GL Lines Plug-in Implementation.

- Create transactions that will impact the general ledger and test the script and plug-in behavior.

- Examine the GL impact of the transaction. Make sure that the plug-in implementation performs as expected.
**Important:** Plug-in implementations modify or supplement NetSuite business logic, changing the manner in which standard business processes run. Make sure that the GL impacts on the transaction types, subsidiaries, and accounting books accurately reflect the customization in the plug-in implementation script file and the plug-in implementation configuration.

You should test your plug-in implementation on each NetSuite version and release in use by your NetSuite customers.

If you use the Custom GL Lines Plug-in with Custom Segments, the plug-in may fail for any one of the following reasons:

- Custom segment does not exist.
- Custom segment is inactive.
- The role set up to run the plug-in does not have permission to set values on the custom segment.
- The custom segment does not impact GL.
- The Custom Segment feature is not enabled.

### Bundle the Plug-in Implementation

After developing the Custom GL Lines plug-in implementation, you can distribute the implementation to a production account. SuiteBundler enables NetSuite users to package together groups of objects for distribution to other accounts. These packages are called bundles, or SuiteApps. To distribute the Custom GL Lines plug-in implementation script file and utility files, custom records and roles, and the plug-in implementation object, create a bundle with SuiteBundler. After you create the bundle, administrators can install the bundle in production accounts.

The following table lists the objects you must include in the bundle and their location on the Select Objects page in the Bundle Builder:

<table>
<thead>
<tr>
<th>Object</th>
<th>Location On Select Objects Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plug-in implementation script file</td>
<td>File Cabinet &gt; Files</td>
</tr>
<tr>
<td>Utility files</td>
<td>File Cabinet &gt; Files</td>
</tr>
<tr>
<td>Custom records</td>
<td>Custom Lists/Records &gt; Records</td>
</tr>
<tr>
<td>Custom roles</td>
<td>Roles &gt; Custom Roles</td>
</tr>
<tr>
<td>Plug-in implementation</td>
<td>Plug-ins &gt; Custom GL Lines Plug-in</td>
</tr>
</tbody>
</table>

### Custom GL Lines Plug-in Implementation Administration

After a developer creates an implementation of a Custom GL Lines plug-in and bundles it, you can install and set up the plug-in implementation bundle.

To install and set up a Custom GL Lines plug-in implementation, complete the following steps:

- Enable Features for a Custom GL Lines Plug-in Implementation
- Install a Custom GL Lines Plug-in Bundle
- Configure the Custom GL Lines Plug-in Implementation
- Enable the Custom GL Lines Plug-in Implementation

You can monitor implementations using system notes. For information, see Custom GL Lines Plug-in Implementation System Notes.

⚠️ Important: The current version of the Custom GL Lines Plug-in is always used. An older version of the plug-in might have been used to create a transaction, and might have generated a different result than the current version. If you attempt to update a transaction in a closed period, and the current version of the plug-in generates a different result, you cannot save the updates.

You can either open the period, or branch your plug-in implementation by the date of the plug-in version.

Enable Features for a Custom GL Lines Plug-in Implementation

Before you install a Custom GL Lines Plug-in bundle, make sure that the Custom GL Lines Plug-in and Server SuiteScript features are enabled in the account.

To enable features for the Custom GL Lines Plug-in:

1. Choose Setup > Company > Enable Features.
2. On the SuiteCloud subtab, enable the following features:

<table>
<thead>
<tr>
<th>Section</th>
<th>Feature Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuiteScript</td>
<td>Server SuiteScript</td>
</tr>
<tr>
<td>SuiteGL</td>
<td>Custom GL Lines</td>
</tr>
</tbody>
</table>

3. If necessary, check the box and agree to the Terms of Service.
4. Click Save.

Install a Custom GL Lines Plug-in Bundle

A developer can create an implementation of the Custom GL Lines plug-in and then bundle it for distribution to other NetSuite accounts. An administrator can then install the bundle into a target NetSuite account.

To install a Custom GL Lines plug-in implementation bundle:

1. Go to Customization > SuiteBundler > Search & Install Bundles.
2. On the Install Bundle page, choose Production Account from the Location list.
4. Click Install for the bundle.

⚠️ Important: To avoid duplicate objects during install, select “Replace Existing Object” if prompted.

After you begin the installation of a bundle, you can continue working in NetSuite as the bundle installs.
Configure the Custom GL Lines Plug-in Implementation

Configure the transaction types, subsidiaries, and accounting books for a Custom GL Lines plug-in implementation from the Plug-In Implementation page. Any configured transaction type with an entity associated with a configured subsidiary triggers the plug-in implementation execution when you save the transaction. If you use the Multi-Book Accounting feature, NetSuite executes the plug-in implementation logic on each of the accounting books you configure for the implementation.

For more information on how the transaction types, subsidiaries, and accounting books affect the running of a plug-in implementation, see Custom GL Lines Plug-in Process Flow.

**Note:** The SuiteScript permission is required to configure a Custom GL Lines plug-in implementation.

**To configure a Custom GL Lines plug-in implementation:**

1. Go to Customization > Plug-ins > Plug-in Implementations.
2. Click the Edit link next to the Custom GL Lines plug-in implementation that you want to configure.
3. Click Configure. The Custom GL Lines: Configuration page appears.
4. By default, plug-ins are run synchronously. If you have multiple plug-ins on a specific transaction type, you might notice that it takes a long time to save transactions. If a transaction type
Custom GL Lines Plug-in Implementation Administration

Custom GL Lines Plug-in takes a long time to save, you can set one or more of the plug-in implementations to run asynchronously.

Every plug-in runs independently and has access only to its own lines and the standard lines, which ensures that if one plug-in fails that the others can run. No priority is assigned to individual plug-ins, meaning that they are run in any order.

5. By default, custom GL lines plug-ins are not run in closed periods and the original GL impact of custom GL lines is preserved. To run the custom GL lines plug-in on transactions in closed periods, check the Run in Closed Periods box. If you check the box and edit a transaction, an error message appears indicating that the GL impact cannot be changed in a closed period.

| Note: | The Run in Closed Periods setting affects only transactions that have a posting period. |

6. The Effective Date and End Date fields are useful when you are updating a script file or adding a new script file for an implementation. In the Effective Date field, enter a start date. The plug-in runs on transaction dates that fall on or after this date. This field is blank by default and is not mandatory.

7. To edit or inactivate the plug-in and preserve the existing GL impact, enter an end date for the older plug-in in the End Date field. The plug-in does not run on transactions dated after this date. This field is blank by default and is not mandatory.

If you are loading a new script for a Custom GL Plug-in implementation, on the configuration for the older version of the plug-in, enter an end date. Add the new script and create a new plug-in implementation with an effective date of the day after the end date of the old plug-in implementation.

If you update the script used by the existing plug-in implementation file, all plug-in implementations that use the file are affected. The best practice is to add a new script file and create a new plug-in implementation with a different effective date.

| Note: | If the end date of one plug-in implementation is the same as the effective date of another plug-in, both plug-ins run on that date. To ensure that plug-ins do not overlap on transaction dates, the effective date of the new plug-in should be one day after the end date of the older plug-in. |

8. Select the transaction types on which the plug-in implementation executes.

To select multiple or all transaction types, press and hold the Ctrl key on your keyboard, then continue selecting transaction types.

Check the ALL box to select all of the transactions types for this configuration.

9. In the Subsidiaries list, select one or more subsidiaries, or check ALL.

10. In the Accounting Books list, select one or more accounting books, or check ALL.

11. Click Save.

When configuring additional plug-ins that overlap with an existing implementation, an information message appears about running the plug-in asynchronously. You can save the configuration, but if you notice long transaction save times, consider making one or more plug-in implementations asynchronous to speed the save process.

Enable the Custom GL Lines Plug-in Implementation

After you have configured a Custom GL Lines plug-in implementation, you can enable it in your account.
**Custom GL Lines Plug-in Implementation Administration**

**Custom GL Lines Plug-in**

Note: You can have multiple Custom GL Lines plug-in implementations enabled in an account. If you enable a plug-in implementation that conflicts with an existing implementation, NetSuite displays a warning message that several implementations for one transaction can cause longer save times. You are allowed to save the implementation, however if you notice long transaction save times, you can resolve this by setting one or more plug-in implementations to run asynchronously.

To enable the Custom GL Lines plug-in implementation:

1. Go to Customization > Plug-ins > Manage Plug-ins.

   ![Manage Plug-In Implementations](image)

   **WARNING**

   Plug-ins are generally designed to modify or supplement NetSuite business logic. When you enable a plug-in, standard NetSuite business processes may no longer run as expected, especially if you enable a plug-in developed by a third party. Therefore, make sure that you understand the impact on your NetSuite account before enabling a plug-in and ensure that use of the plug-in meets all of your security, regulatory, and other compliance obligations, including payment card industry (PCI) Data Security Standards.

2. Under Custom GL Lines Plug-in, check the box next to the name of the plug-in implementation.
3. Click Save.

Note: The plug-in implementation executes when users save transactions configured for the implementation, according to the plug-in logic. Users can see the custom lines added or modified by the plug-in implementation when they click Actions > GL Impact on a saved transaction.

**Custom GL Lines Plug-in Implementation System Notes**

System notes are logged on custom GL lines plug-in implementations.

A System Notes subtab is available on the plug-in implementation configuration page for Custom GL Lines Plug-ins. The system notes show the changes that have been made to the plug-in implementation configuration settings.
Custom GL Lines Plug-In Interface Definition

Interface Function

The Custom GL Lines plug-in interface includes the following functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customizeGlImpact(transactionRecord, standardLines, customLines, book)</td>
<td>Contains the business logic to modify the general ledger impact for a transaction with the Custom GL Lines plug-in. For more information about how NetSuite uses this function, see Custom GL Lines Plug-in Process Flow.</td>
</tr>
</tbody>
</table>

⚠️ **Important:** You cannot change this function signature in a Custom GL Lines plug-in implementation.

Custom GL Lines Plug-In Object Model

The following figure shows the object model for the interface input objects:

![Object Model Diagram]
**customizeGlImpact(transactionRecord, standardLines, customLines, book)**

**Function Declaration**
```java
void customizeGlImpact(Record transactionRecord, StandardLines standardLines, CustomLines customLines, AccountingBook book)
```

**Type**
Interface function

**Description**
Contains the business logic to modify the general impact for a transaction with the Custom GL Lines Plug-in. Use the objects passed to this function by NetSuite to implement the following business logic in a Custom GL Lines plug-in implementation:

- Get information about general ledger impact. Use the `StandardLines` object to get information about each line of general ledger impact for a transaction.
- Create custom lines. Use the `CustomLines` object to create lines with general ledger impact in primary and secondary accounts for a transaction.

**Note:** You can also use SuiteScript API functions to access transaction record properties in a plug-in implementation. See `Record`.

**Returns**
void

**Parameters**
- `StandardLines`
- `CustomLines`
- `AccountingBook`
- `Record`

**StandardLines**

**Type**
Interface input object

**Description**
Contains an array of all standard lines with GL impact in a transaction as `StandardLine` objects. Standard lines are the general ledger impacts that appear on the GL Impact report for a transaction.

Use this object to access individual standard lines for an accounting book in a transaction. For more information, see Custom GL Lines Plug-in Process Flow.

**Methods**
- `getCount()`
- `getLine(index)`

**Child Object(s)**
- `StandardLine`

**getCount()**

**Function Declaration**
```java
int getCount()
```

**Type**
Object method
Description

Returns the number of standard lines with GL impact for a specific accounting book in a transaction. Use this method in conjunction with `getLine(index)` to read individual standard lines.

`getCount()` counts both visible and hidden lines on the GL Impact page. For an example of how to return only the number of visible lines, see Custom GL Lines Plug-in Guidelines and Best Practices.

Returns

int

Input Parameters

None.

Parent object

`StandardLines`

Example

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    for (var i = 0; i < standardLines.getCount(); i++)
    {
        var currLine = standardLines.getLine(i);
        ...
    }
}
```

g.LINE(index)

Function Declaration

`StandardLine getLine(int index)`

Type

Object method

Description

Returns a `StandardLine` object that represents a standard line with GL impact. StandardLine objects are stored in the `StandardLines` object starting at index 0.

Returns

`StandardLine`

Input Parameters

`index (int)` — Integer value for the `StandardLine` object at the specified position in the `StandardLines` object.

Parent object

`StandardLines`

Example

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    for (var i = 0; i < standardLines.getCount(); i++)
    {
        var currLine = standardLines.getLine(i);
        ...
    }
}
```

`StandardLine`

Type

Object
### Description
Contains all properties for a single standard line on the GL impact on a transaction. Use the methods available to the StandardLine object to get the values for the standard line and define plug-in implementation functionality based on the values. The StandardLines object has a StandardLine object for each standard GL impact line.

### Methods
- `getEntityId()`
- `getId()`
- `getSubsidiaryId()`
- `getTaxableAmount()`
- `getTaxAmount()`
- `getTaxItemId()`
- `getTaxType()`
- `isPosting()`
- `isTaxable()`

**Note:** The StandardLine and CustomLine objects share some common methods. For more information, see Common StandardLine and CustomLine Object Methods.

### Parent Object(s)
StandardLines

### Child Object(s)
- `n/a`

### `getEntityId()`

**Function Declaration**
```java
int getEntityId()
```

**Type**
Object method

**Description**
Returns the internal NetSuite ID of the entity for a standard line. The entity can be any customer, employee, partner, or vendor, depending on the type of transaction.

You can use this ID, for example, to define plug-in implementation functionality based on the entity ID, to load the record for the entity with `nlapiLoadRecord(type, id, initializeValues)`, or to search for the record with `nlapiSearchRecord(type, id, filters, columns)`. You can look up the type of entity based on the record type. Use `getRecordType()` to get the record object type. See Record.

**Important:** Limit the use of these APIs, where possible, to improve performance of the plug-in implementation. Also, searching for a record results in better performance than loading the record.

To implement functionality based on the entity ID, you can use helper functions to avoid adding the internal NetSuite ID into the plug-in implementation logic. See Utility Files for a Custom GL Lines Plug-In Implementation.

**Returns**
int

**Input Parameters**
- None.

**Parent object**
StandardLine

### Example 1: Loading the Record
```plaintext
function customizeGlImpact(transactionRecord, standardLines, customLines, book) {
  ```
for (var i = 0; i < standardLines.getCount(); i++)
{
    var currLine = standardLines.getLine(i);
    var recType = transactionRecord.getRecordType();
    if (recType == 'salesorder')
    {
        var custRecord = nlapiLoadRecord('customer', currLine.getEntityId());
        // get record properties here
        ...
    }
    ...
}

Example 2: Searching for the Record

// utility function
function searchNeededPropertiesOfCustomer(customerId)
{
    var searchFilters = new Array();
    searchFilters[0] = new nlobjSearchFilter('internalid', null, 'anyof', customerId);
    var savedSearchResults = nlapiSearchRecord('customer', 'customsearch_script_id', searchFilters);
    if (savedSearchResults != null && savedSearchResults.length > 0)
    {
        return savedSearchResults[0];
    }
    return null;
}

function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    for (var i = 0; i < standardLines.getCount(); i++)
    {
        var currLine = standardLines.getLine(i);
        var recType = transactionRecord.getRecordType();
        if (recType == 'salesorder')
        {
            // nlobjSearchResult
            var custMap = searchNeededPropertiesOfCustomer(currLine.getEntityId());
            // get search columns here
            ...
        }
    }
}

gid()
### Custom GL Lines Plug-in

#### Description

Returns the internal NetSuite database ID for a standard GL impact line. The summary line for the GL impact on a transaction occurs for most transaction types at ID of 0. Some transactions, like journals, do not have a summary line.

You can use the summary information to define plug-in implementation functionality based on the values in the summary line.

**Note:** The summary line for the GL impact on a transaction does not include tax information.

**Returns**

*int*

**Input Parameters**

None.

**Parent object**

`StandardLine`

**Example**

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book) {
    ...
    for (var i = 0; i < standardLines.getCount(); i++)
    {
        var currLine = standardLines.getLine(i);
        if (currLine.getId() == 0)
        {
            // it's a summary line
        }
        ...
    }
}
```

---

### `getSubsidiaryId()`

**Function Declaration**

`int getSubsidiaryId()`

**Type**

Object method

**Description**

Returns the internal NetSuite ID of the subsidiary for the entity associated with a standard GL impact line. You can view all subsidiaries and the associated internal IDs in NetSuite at Setup > Company > Subsidiaries.

You can use this ID, for example, to define plug-in implementation functionality based on the subsidiary ID, to load the record for the subsidiary with `nlapiLoadRecord(type, id, initializeValues)`, or to search for the record with `nlapiSearchRecord(type, id, filters, columns).` The type parameter for a subsidiary is `subsidiary`.

**Important:** Limit the use of these APIs, where possible, to improve performance of the plug-in implementation. Also, searching for a record results in better performance than loading the record.

You can use helper functions in a utility file to avoid adding the internal NetSuite IDs into the plug-in implementation logic. See Utility Files for a Custom GL Lines Plug-In Implementation.

**Returns**

*int*

**Input Parameters**

None.
Parent object  StandardLine

Example

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    for (var i = 0; i < standardLines.getCount(); i++)
    {
        var currLine = standardLines.getLine(i);
        var subId = currLine.getSubsidiaryId();
        ...
        var subRecord = nlapiLoadRecord('subsidiary',subId,'');
        ...
    }
    ...
}
```

getTaxableAmount()

Function Declaration  string getTaxableAmount()

Type  Object method

Description  Returns a string that represents the amount of a standard GL line that was subject to tax. Returns 0 if the line was not subject to tax. Use this method with getTaxAmount(), getTaxItemID(), and getTaxType() to get the tax information for a standard GL impact line.

You can use helper functions in a utility file to avoid adding the internal NetSuite IDs into the plug-in implementation logic. See Utility Files for a Custom GL Lines Plug-In Implementation.

Returns  string

Input Parameters  None.

Parent object  StandardLine

Example

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    for (var i = 0; i < standardLines.getCount(); i++)
    {
        var currLine = standardLines.getLine(i);
        // check tax
        if (currLine.getTaxItemID() !== -7) // taxable transaction
        {
            var taxType = currLine.getTaxType();
            if (taxType == 'US_ST')
            {
                // process state taxes
                var taxAmt = currLine.getTaxAmount();
                var taxableAmt = currLine.getTaxableAmount();
                ...
            }
        }
    }
}
getTaxAmount()

**Function Declaration**
```
string getTaxAmount()
```

**Type**
Object method

**Description**
Returns a string that represents the amount of tax charged on a standard GL line. This method only returns a value other than 0 if the standard line is a credit to a tax account. For example, if a standard line on a taxable transaction contains a credit value of 10.00 for an account of type Other Current Liability, this method returns -10.

Use this method with `getTaxableAmount()`, `getTaxItemId()`, and `getTaxType()` to get the tax information for a standard GL impact line.

**Returns**
- `string`

**Input Parameters**
None.

**Parent object**
StandardLine

**Example**
```
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    for (var i = 0; i < standardLines.getCount(); i++)
    {
        var currLine = standardLines.getLine(i);
        // check tax
        if (currLine.getTaxItemId() != -7) // taxable transaction
        {
            var taxType = currLine.getTaxType();
            if (taxType == "US_ST")
            {
                // process state taxes
                var taxAmt = currLine.getTaxAmount();
                var taxableAmt = currLine.getTaxableAmount();
                ...
            }
        }
    }
    ...
}
```

getTaxItemIds()

**Function Declaration**
```
int getTaxItemIds()
```

**Type**
Object method
**customizeGlImpact(transactionRecord, standardLines, customLines, book)**

**Description**
Returns the internal NetSuite ID of the tax code for a standard GL line. Returns null if the line was not subject to tax or if the line does not credit a tax account with a tax liability.

You can use this ID to define plug-in implementation functionality based on the tax code. For example, a tax code of -7 indicates the line was not subject to tax. You can view all tax codes in NetSuite at Setup > Accounting > Tax Codes.

You can use helper functions in a utility file to avoid adding the internal NetSuite IDs into the plug-in implementation logic. See Utility Files for a Custom GL Lines Plug-In Implementation.

Use this method with `getTaxableAmount()`, `getTaxAmount()`, and `getTaxType()` to get the tax information for a standard GL impact line.

**Returns**
int

**Input Parameters**
None.

**Parent object**
StandardLine

**Example**

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book) {
    ...
    for (var i = 0; i < standardLines.getCount(); i++) {
        var currLine = standardLines.getLine(i);
        // check tax
        if (currLine.getTaxItemId() != -7) // taxable transaction
            {
                var taxType = currLine.getTaxType();
                if (taxType == "US_ST")
                    {
                    // process state taxes
                    var taxAmt = currLine.getTaxAmount();
                    var taxableAmt = currLine.getTaxableAmount();
                    ...
                    }
            }
    }
}
```

**getTaxType()**

**Function Declaration**
string getTaxType()

**Type**
Object method

**Description**
Returns the tax type for a standard GL line that was subject to tax. Returns null if the standard line was not subject to tax.

You can use this ID to define plug-in implementation functionality based on the tax type. You can view all tax types in NetSuite at Setup > Accounting > Tax Types.

Use this method with `getTaxableAmount()`, `getTaxAmount()`, and `getTaxItemId()` to get the tax information for a standard GL impact line.

**Returns**
string
Input Parameters: None.

Parent object: StandardLine

Example:

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    for (var i = 0; i < standardLines.getCount(); i++)
    {
        var currLine = standardLines.getLine(i);
        // check tax
        if (currLine.getTaxItemId() != -7) // taxable transaction
        {
            var taxType = currLine.getTaxType();
            if (taxType == "US_ST")
            {
                // process state taxes
                var taxAmt = currLine.getTaxAmount();
                var taxableAmt = currLine.getTaxableAmount();
                ...
            }
        }
    }
    ...
}
```

isPosting()

Function Declaration: boolean isPosting()

Type: Object method

Description: Returns true if the transaction is a posting transaction and the associated standard GL impact line posts to the general ledger. Returns false if the transaction is a non-posting transaction.

NetSuite transactions can be posting, non-posting, or posting with approval. For example, journal entries do not post without approval. For more information, see the help topic General Ledger Impact of Transactions.

Returns: boolean

Input Parameters: None.

Parent object: StandardLine

Example:

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    // find all posting lines
    for (var i = 0; i < standardLines.getCount(); i++)
    {
        var currLine = standardLines.getLine(i);
```
if (currLine.isPosting())
{
    // process standard line here
}

isTaxable()

Function Declaration: boolean isTaxable()
Type: Object method
Description: Returns true if a standard GL impact line is a credit to a tax account. For example, if a standard line on a taxable transaction contains a credit value of 10.00 for an account of type Other Current Liability, this method returns true.
Returns: boolean
Input Parameters: None.
Parent object: StandardLine

Example

function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    // find all taxable lines
    for (var i = 0; i < standardLines.getCount(); i++)
    {
        var currLine = standardLines.getLine(i);
        if (currLine.isTaxable())
        {
            // process tax here
        }
    }
    ...
}

CustomLines

Type: Interface input object
Description: Contains an array of all custom lines with GL impact in a transaction as CustomLine objects. Use this object to add and modify custom lines with GL impact on a transaction. Create a new CustomLine object with addNewLine().
For more information, see Custom GL Lines Plug-in Process Flow.
Methods:
- addNewLine()
- getCount()
- getLine(index)
**addNewLine()**

**Function Declaration** CustomLine addNewLine()

**Type** Object method

**Description**

Adds a CustomLine object to the parent CustomLines object in a Custom GL Lines plug-in implementation and returns the new object. Use this method to add a custom line with GL impact to a transaction.

After you create the custom line, use the methods available to the CustomLine object to set the properties of the custom line, including the general ledger account ID and the amount of the custom line.

**Note:** If you use the Multi-Book Accounting feature and want this line to be copied to the secondary accounting books, use setBookSpecific(bookSpecific).

**Returns** CustomLine

**Input Parameters** None.

**Parent object** CustomLines

**Example**

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
  ...
  var newLine = customLines.addNewLine();
  newLine.setCreditAmount(standardLines.getLine(0).getCreditAmount());
  newLine.setAccountId(standardLines.getLine(0).getAccountId());
  newLine.setMemo("Payment catches both revenue and cash.");
  ...
  var newLine = customLines.addNewLine();
  newLine.setDebitAmount(standardLines.getLine(1).getDebitAmount());
  newLine.setAccountId(standardLines.getLine(1).getAccountId());
  newLine.setMemo("Payment catches both revenue and cash.");
  ...
}
```

**getCount()**

**Function Declaration** int getCount()

**Type** Object method

**Description**

Returns the number of custom lines with GL impact for a specific accounting book in a transaction. These lines are added with addNewLine(). Use this method in conjunction with getLine(index) to get individual custom lines after you add them in the plug-in implementation.

**Returns** int

**Input Parameters** None.
Custom GL Lines Plug-in

Parent object: CustomLines

Example

```
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    for (var i = 0; i < customLines.getCount(); i++)
    {
        // get the line
        var currLine = customLines.getLine(i);
        ...
    }
}
```

gLine(index)

Function Declaration: CustomLine getLine(int index)

Type: Object method

Description: Returns a CustomLine object that represents a custom line with GL impact. CustomLine objects are stored in the CustomLines object starting at index 0.

Returns: CustomLine

Input Parameters: index (int) — Integer value for the CustomLine object at the specified position in the CustomLines object.

Parent object: CustomLines

Example

```
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    for (var i = 0; i < customLines.getCount(); i++)
    {
        // get the line
        var currLine = customLines.getLine(i);
        ...
    }
}
```

CustomLine

Type: Object

Description: Contains all properties for a single custom line for the GL impact on a transaction. Use the methods available to the CustomLine object to set the values for the custom line and define plug-in implementation functionality based on the values. The CustomLines object contains a reference to each custom GL impact line.

Create a new CustomLine object with addNewLine().

Methods:
- isBookSpecific()
setBookSpecific(bookSpecific)
setAccountid(accountId)
setClassId(classId)
setCreditAmount(credit)
setDebitAmount(debit)
setDepartmentId(departmentId)
setEntityId(entityId)
setLocationId(locationId)
setMemo(memo)
setSegmentValueld(segmentId, segmentValueId)

Note: The StandardLine and CustomLine objects share some common methods.
For more information, see Common StandardLine and CustomLine Object Methods.

Parent Object(s) CustomLines
Child Object(s) n/a

isBookSpecific()

Function Declaration boolean isBookSpecific()
Type Object method
Description Returns true if the custom line is specific to the primary book and should not be copied to the secondary accounting books. Returns false if the custom line is not specific to the primary accounting book.

Note: This method only applies if you use the Multi-Book Accounting feature and the method is being called for the primary accounting book. For more information about how the plug-in implementation processes book-specific lines, see Custom GL Lines Plug-in Process Flow.

Returns boolean
Input Parameters None.
Parent object CustomLine

Example

function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
  ...
  if(book.isPrimary())
  {
    // set all custom lines to apply to all books
    for (var i = 0; i < customLines.getCount(); i++)
    {
      var currLine = customLines.getLine(i);
      if (currLine.isBookSpecific())
      {
        currLine.setBookSpecific(false);
      }
    }
  }
}
setBookSpecific(bookSpecific)

Function Declaration
void setBookSpecific(boolean bookSpecific)

Type
Object method

Description
Sets a custom GL impact line to affect only the primary book in a Custom GL plug-in implementation. If you use this method and set the value to false, the plug-in implementation copies the custom line to secondary books and the custom line is also subject to any mapping set up for the Multi-Book Accounting feature.

If you do not use this method or set the value to true, the custom GL impact line only applies to the primary accounting book and is not copied to secondary accounting books.

Returns
void

Input Parameters
bookSpecific (boolean) — Use true to make the custom line specific to the primary accounting book. Use false to direct NetSuite to copy the custom line to secondary accounting books.

Parent object
CustomLine

Example

function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
  ... 
  if (book.isPrimary())
  {
    // set all custom lines to be specific to the primary book
    for (var i = 0; i < customLines.getCount(); i++)
    {
      var currLine = customLines.getLine(i);
      if (!currLine.isBookSpecific())
      {
        currLine.setBookSpecific(true);
        ...
      }
    }
  }
  ...
}

setAccountId(accountId)

Function Declaration
void setAccountId(int accountId)

Type
Object method
customizeGlImpact(transactionRecord, standardLines, customLines, book)

**Description**
Sets the account ID property for a CustomLine object in a primary or secondary book. This value is the internal NetSuite ID for a general ledger account. You must use this method to set the account to be debited or credited for a custom line.

You can view the internal NetSuite ID for all accounts on the Chart of Accounts page at Setup > Accounting > Chart of Accounts.

You can use helper functions in a utility file to customize the internal NetSuite account IDs in the plug-in implementation logic. See Utility Files for a Custom GL Lines Plug-In Implementation.

**Returns**
void

**Input Parameters**
accountId (int) — Internal NetSuite ID for an account.

**Parent object**
CustomLine

**Example**
```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book) {
    ...
    var newLine = customLines.addNewLine();
    newLine.setCreditAmount(standardLines.getLine(0).getCreditAmount());
    newLine.setAccountId(standardLines.getLine(0).getAccountId());
    newLine.setMemo("Payment catches both revenue and cash.");
    ...
    var newLine = customLines.addNewLine();
    newLine.setDebitAmount(standardLines.getLine(1).getDebitAmount());
    newLine.setAccountId(standardLines.getLine(1).getAccountId());
    newLine.setMemo("Payment catches both revenue and cash.");
    ...
}
```

**setClassId(classId)**

**Function Declaration**
void setClassId(int classId)

**Type**
Object method

**Description**
Sets the class ID property for a CustomLine object in a primary or secondary book. This value is the internal NetSuite ID for a class.

Classes are categories that you can create to track records such as financials, transactions, and employees. You can view all classes defined in NetSuite at Setup > Company > Classes. See also getClassId().

You can use helper functions in a utility file to avoid adding the internal NetSuite IDs into the plug-in implementation logic. See Utility Files for a Custom GL Lines Plug-In Implementation.

**Returns**
void

**Input Parameters**
classId (int) — Internal NetSuite ID for a class.

**Parent object**
CustomLine

**Example**
```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book) {
    ...
    var newLine = customLines.addNewLine();
    newLine.setCreditAmount(standardLines.getLine(0).getCreditAmount());
    newLine.setAccountId(standardLines.getLine(0).getAccountId());
    newLine.setMemo("Payment catches both revenue and cash.");
    ...
    var newLine = customLines.addNewLine();
    newLine.setDebitAmount(standardLines.getLine(1).getDebitAmount());
    newLine.setAccountId(standardLines.getLine(1).getAccountId());
    newLine.setMemo("Payment catches both revenue and cash.");
    ...
}
```
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    var context = nlapiGetContext();
    var makeClassesMandatory = context.getPreference('classmandatory');
    var allowPerLineClasses = context.getPreference('classesperline');
    ...
    var newLine = customLines.addNewLine();
    ...
    if (makeClassesMandatory || allowPerLineClasses)
    {
        var classId = getClassForItem(itemId);
        newLine.setClassId(classId);
    }
    ...
    
    // Utility methods
    function getClassForItem(itemId)
    {
        ...
        return classId;
    }
}

setCreditAmount(credit)

**Function Declaration**

void setCreditAmount(credit)

**Type**

Object method

**Description**

Sets the credit amount of a *CustomLine* object in a primary or secondary book. The value is rounded to currency precision.

To set a debit amount, use *setDebitAmount(debit)*.

**Note:** The currency for the amount of the custom line depends on the subsidiary and accounting book to which the GL impact posts. For example, if the accounting book being processed by the plug-in implementation uses USD, the currency for the amount set by this method is USD. For more information, see *Multiple Currencies Examples*.

**Returns**

void

**Input Parameters**

credit (string) — String value of a credit on a general ledger account. Requires a positive value.

**Parent object**

*CustomLine*

**Example**

function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    var newLine = customLines.addNewLine();
    newLine.setCreditAmount(standardLines.getLine(0).getCreditAmount());
    newLine.setAccountId(standardLines.getLine(0).getAccountId());
    newLine.setMemo("Payment catches both revenue and cash.");
    ...
}
setDebitAmount(debit)

Function Declaration
void setDebitAmount(debit)

Type
Object method

Description
Sets the debit amount of a CustomLine object in a primary or secondary book. The value is rounded to currency precision.

To set a credit amount, use setCreditAmount(credit).

Note: The currency for the amount on the custom line depends on the accounting book to which the GL impact posts. For example, if the accounting book being processed by the plug-in implementation uses USD, the currency for the amount set by this method is USD. For more information, see Multiple Currencies Examples.

Returns
void

Input Parameters
debit (string) — String value of a debit on a general ledger account. Requires a positive value.

Parent object
CustomLine

Example
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
...
  var newLine = customLines.addNewLine();
  newLine.setDebitAmount(standardLines.getLine(1).getDebitAmount());
  newLine.setAccountId(standardLines.getLine(1).getAccountId());
  newLine.setMemo("Payment catches both revenue and cash.");
  ...
}

setDepartmentId(departmentId)

Function Declaration
void setDepartmentId(int departmentId)

Type
Object method

Description
Sets the department ID for a CustomLine object in a primary or secondary book. This value is the internal NetSuite ID for a department.
 Departments are listed first on transactions, and are useful when designating transactions and employees as part of an internal team. You can view all departments defined in NetSuite at Setup > Company > Departments. See also `getDepartmentId()`.

You can use helper functions in a utility file to avoid adding the internal NetSuite IDs into the plug-in implementation logic. You can also access the `nlobjContext` object with `nlapiGetContext()` to get the preferences for locations at Setup > Accounting > Accounting Preferences.

**Returns**

`void`

**Input Parameters**

- departmentId (int) — Internal NetSuite ID for a department.

**Parent object**

`CustomLine`

**Example**

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book) {
    var context = nlapiGetContext();
    var makeDepartmentsMandatory = context.getPreference('deptmandatory');
    var allowPerLineDepartments = context.getPreference('deptsperline');
    ...
    var newRow = customLines.addNewLine();
    ...
    if (makeDepartmentsMandatory || allowPerLineDepartments) {
        var departmentId = getDepartmentForItem(itemId);
        newRow.setDepartmentId(departmentId);
    }
    ...
}

// Utility methods
function getDepartmentForItem(itemId) {
    ...
    return departmentId;
}
```

---

**setEntityId(entityId)**

<table>
<thead>
<tr>
<th>Function Declaration</th>
<th><code>void setEntityId(int entityId)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Object method</td>
</tr>
<tr>
<td>Description</td>
<td>Sets the entity ID property for a <code>CustomLine</code> object in a primary or secondary book to the internal NetSuite ID</td>
</tr>
</tbody>
</table>

Supported entity types include:

- Customers
- Jobs
- Leads
- Prospects
- Vendors
- Employees
Other Names

The following restrictions apply to setting of entities on custom GL lines:

- Setting of a vendor is not supported on lines with an Accounts Payable account type.
- Setting of a customer is not supported on lines with an Accounts Receivable account type.
- Creating book-specific custom GL lines with an AP or AR account and entity combination. Only generic custom GL lines can be created with this combination.
- Only entities that have the same subsidiary as a transaction can be set on lines for that transaction.
- Only entities that have the same currency as a transaction can be set on lines for that transaction.
- Entities must be active

Note: This method does not set the sales representative for the line, and it does not change the date of the first sale for the customer.

<table>
<thead>
<tr>
<th>Returns</th>
<th>void</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Parameters</td>
<td>entityId (int) — internal NetSuite ID for an entity</td>
</tr>
<tr>
<td>Parent object</td>
<td>CustomLine</td>
</tr>
</tbody>
</table>

Example

```javascript
function customizeGlImpact(record, standardLines, customLines, book)
{
    var line = customLines.addNewLine()
    line.setAccountId(6)
    line.setDebitAmount(100)
    line.setEntityId(99)

    line = customLines.addNewLine()
    line.setAccountId(7)
    line.setCreditAmount(100)
}
```

Reports that include GL information and are grouped by entity include custom GL information, if the entity is supported by the new method.

setLocationId(locationId)

Function Declaration  void setLocationId(int locationId)
customizeGlImpact(transactionRecord, standardLines, customLines, book)

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Sets the location ID for a CustomLine object in a primary or secondary book. This value is the internal NetSuite ID for a location. Use locations to track information about employees and transactions for multiple offices or warehouses. You can view all locations defined in NetSuite at Setup &gt; Company &gt; Locations. See also getLocationId(). You can use helper functions in a utility file to avoid adding the internal NetSuite IDs into the plug-in implementation logic. You can also access the nlobjContext object with nlapiGetContext() to get the preferences for locations at Setup &gt; Accounting &gt; Accounting Preferences.</td>
</tr>
<tr>
<td>Returns</td>
<td>void</td>
</tr>
<tr>
<td>Input Parameters</td>
<td>locationId {int} — Internal NetSuite ID for a location.</td>
</tr>
<tr>
<td>Parent object</td>
<td>CustomLine</td>
</tr>
</tbody>
</table>

**Example**

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book) {
    var context = nlapiGetContext();
    var makeLocationsMandatory = context.getPreference('locmandatory');
    var allowPerLineLocations = context.getPreference('locsperline');
    ...
    var newLine = customLines.addNewLine();
    ...
    if (makeLocationsMandatory || allowPerLineLocations) {
        var locationId = getLocationForItem(itemId);
        newLine.setLocationId(locationId);
    }
    ...
}

// Utility methods
function getLocationForItem(itemId) {
    ...
    return locationId;
}
```

**setMemo(memo)**

<table>
<thead>
<tr>
<th>Function Declaration</th>
<th>void setMemo(string memo)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Object method</td>
</tr>
<tr>
<td>Description</td>
<td>Sets the Memo field on a CustomLine object. See also getMemo().</td>
</tr>
<tr>
<td>Returns</td>
<td>void</td>
</tr>
<tr>
<td>Input Parameters</td>
<td>memo (string) — String text for the Memo field.</td>
</tr>
<tr>
<td>Parent object</td>
<td>StandardLine, CustomLine</td>
</tr>
</tbody>
</table>
Example

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    var newLine = customLines.addNewLine();
    newLine.setCreditAmount(standardLines.getLine(0).getCreditAmount());
    newLine.setAccountId(standardLines.getLine(0).getAccountId());
    newLine.setMemo('Payment catches both revenue and cash.');
    ...
    var newLine = customLines.addNewLine();
    newLine.setDebitAmount(standardLines.getLine(1).getDebitAmount());
    newLine.setAccountId(standardLines.getLine(1).getAccountId());
    newLine.setMemo('Payment catches both revenue and cash.');
    ...
}
```

**setSegmentValueId(segmentId, segmentValueId)**

**Function Declaration**  
void setSegmentValueId(segmentId, segmentValueId)

**Type**  
Object method

**Description**  
Sets custom segment values on a CustomLine object

NetSuite assigns custom segment values as follows:

- If the custom segment has a body and lines, the value is copied from the body to all lines. The plug-in can be used to change the value of the lines.
- If the custom segment has only a body, the value is copied from the body to all lines. The plug-in can be used to change the value.
- If the custom segment has only a line value, no values are copied.

**Returns**  
void

**Input Parameters**  
segmentId (string) — String value of the custom segment ID  
segmentValue (integer) — Integer value that the custom line should be set to

**Parent object**  
CustomLines

**Example**

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    var newLine = customLines.addNewLine();
    ...
    // custom function for getting value for cost center segment
    newLine.setSegmentValueId('csegcostcenter', getCostCenterValueForItem(itemId));
    // using constant for region segment
    newLine.setSegmentValueId('csegregion', 3);
    // copying segment value from line 0 for work center segment
    var workcenter = 'csegworkcenter'
    newLine.setSegmentValueId(workcenter, standardLines.getLine(0).getSegmentValueId(workcenter));
    ...
}
```
AccountingBook

**Type**

Interface input object

**Description**

Represents the accounting book passed to a Custom GL Lines plug-in implementation when you save a transaction. Use the methods available to the book object to determine if the book is a primary or secondary book or get the internal NetSuite ID of the accounting book.

If you use the Multi-Book Accounting feature, the AccountingBook object represents a different accounting book each time the plug-in implementation executes. For more information, see Custom GL Lines Plug-in Process Flow.

**Methods**

- `getId()`
- `isPrimary()`

**Child Object(s)**

n/a

### getId()

**Function Declaration**

```javascript
int getId()
```

**Type**

Object method

**Description**

Returns the internal NetSuite ID for the accounting book to be passed to a Custom GL Lines plug-in implementation. You can use this ID, for example, to load the accounting book record with the `nlapiLoadRecord(type, id, initializeValues)` SuiteScript API.

You can use this ID, for example, to define plug-in implementation functionality based on the accounting book ID, to load the record for the book with `nlapiLoadRecord(type, id, initializeValues)`, or to search for the record with `nlapiSearchRecord(type, id, filters, columns)`. The type for an accounting book record is `accountingbook`.

**Important:** Limit the use of these APIs, where possible, to improve performance of the plug-in implementation. Also, searching for a record results in better performance than loading the record.

You can view the list of internal NetSuite IDs for accounting books at Setup > Accounting > Accounting Books.

You can also use helper functions in a utility file to avoid adding the internal NetSuite IDs into the plug-in implementation logic. See Utility Files for a Custom GL Lines Plug-In Implementation.

**Returns**

`int`

**Input Parameters**

None.

**Parent object**

AccountingBook

**Example**

```javascript
// Utility methods
function getCostCenterValueForItem(itemId)
{
  ...
  return valueId;
}
```
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
  ...
  var bookId = book.getId();
  if (!book.isPrimary())
  {
    var bookRec = nlapiLoadRecord('accountingbook', bookId);
    ...
  }
  ...
}

isPrimary()

Function Declaration  boolean isPrimary()
Type  Object method
Description  Returns true if the book object is the primary accounting book for the NetSuite account or returns false if the accounting book is a secondary accounting book.

Returns  boolean
Input Parameters  None.
Parent object  AccountingBook

Example

function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
  ...
  var bookId = book.getId();
  if (!book.isPrimary())
  {
    var bookRec = nlapiLoadRecord('accountingbook', bookId);
    ...
  }
  ...
}

Record

Type  Interface input object
Description  Represents the current transaction record object passed to a Custom GL Lines plug-in implementation. Use the SuiteScript API functions available to the Record object to access field values on the transaction.

Note: You can only use functions to read values from the transaction record. You cannot write data to the transaction record.

Methods  See the following table for details on the available methods and the help topic nlobjRecord in the Help Center.
Parent Object(s)  n/a
Child Object(s)  n/a

Record Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAllFields()</td>
<td>Returns a normal keyed array of all the fields on a record.</td>
</tr>
<tr>
<td>getAllLineItemFields(group)</td>
<td>Returns an array of all the field names of a sublist on the record.</td>
</tr>
<tr>
<td>getFieldText(name)</td>
<td>Returns the UI display value for a select field.</td>
</tr>
<tr>
<td>getFieldTexts(name)</td>
<td>Returns the UI display values for a multi-select field.</td>
</tr>
<tr>
<td>fieldValue(name)</td>
<td>Returns the value (internal ID) of a field.</td>
</tr>
<tr>
<td>fieldValueValues(name)</td>
<td>Returns the value (field ID) or values (array of field IDs) of a multi-select field.</td>
</tr>
<tr>
<td>getId()</td>
<td>Use this method to get the internal ID of a record when editing an existing transaction.</td>
</tr>
<tr>
<td>getItemCount(group)</td>
<td>Returns the number of lines on a sublist.</td>
</tr>
<tr>
<td>getItemText(group, fldnam, linenum)</td>
<td>Returns the display name of a select field (based on its current selection) in a sublist.</td>
</tr>
<tr>
<td>getItemValue(group, name, linenum)</td>
<td>Returns the value of a sublist line item field.</td>
</tr>
<tr>
<td>getItemValueValues(type, fldnam, linenum)</td>
<td>Returns the values of a multiselect sublist field on a selected line.</td>
</tr>
<tr>
<td>getRecordType()</td>
<td>Returns the record type. For example, returns Unbuild for the Assembly Unbuild record type or returns SalesOrd for the Sales Order record type.</td>
</tr>
<tr>
<td>viewItemSubrecord(sublist, fldname, linenum)</td>
<td>Returns a nlobjSubrecord object. Use this API to view a subrecord from a sublist field on the parent record.</td>
</tr>
<tr>
<td>viewSubrecord(fldname)</td>
<td>Returns a nlobjSubrecord object. Use this API to view a subrecord from a body field on the parent record. NetSuite returns CANNOT_EDIT_SUBRECORD if you attempt to edit the subrecord returned by this function.</td>
</tr>
</tbody>
</table>

Common StandardLine and CustomLine Object Methods

The following object methods are referenced by both the StandardLine and CustomLine objects:

- getAccountId()
- getClassId()
- getCreditAmount()
- getDebitAmount()
- getDepartmentId()
Common StandardLine and CustomLine Object Methods

- getEntityId()
- getLocationId()
- getMemo()
- getSegmentValueId(segmentId)

**getAccountId()**

**Function Declaration**

```java
int getAccountId()
```

**Type**

Object method

**Description**

Returns the internal NetSuite account ID for a StandardLine or CustomLine object. This account ID represents the NetSuite general ledger account to which a credit or debit is applied on the GL impact for a transaction. Use this ID to define plug-in implementation functionality based on the credited or debited account. See also `setAccountId(accountId)`.

You can view the internal NetSuite ID for all accounts on the Chart of Accounts page at Setup > Accounting > Chart of Accounts.

You can use helper functions in a utility file to avoid adding the internal NetSuite IDs into the plug-in implementation logic. See Utility Files for a Custom GL Lines Plug-In Implementation.

**Returns**

int

**Input Parameters**

None.

**Parent object**

StandardLine, CustomLine

**Example**

```javascript
function customizeGlImpact(record, standardLines, customLines, book)
{
    ...
    var newLine = customLines.addNewLine();
    newLine.setCreditAmount(standardLines.getLine(0).getCreditAmount());
    newLine.setAccountId(standardLines.getLine(0).getAccountId());
    newLine.setMemo("Payment catches both revenue and cash.");
    ...
    var newLine = customLines.addNewLine();
    newLine.setDebitAmount(standardLines.getLine(1).getDebitAmount());
    newLine.setAccountId(standardLines.getLine(1).getAccountId());
    newLine.setMemo("Payment catches both revenue and cash.");
    ...
}
```

**getClassId()**

**Function Declaration**

```java
int getClassId()
```

**Type**

Object method

**Description**

Returns the internal NetSuite ID for the class on a StandardLine or CustomLine object. Use this method to define plug-in implementation functionality based on the class for a GL impact line. See also `setClassId(classId)`.

**Returns**

int

**Input Parameters**

None.

**Parent object**

StandardLine, CustomLine
Classes are categories that you can create to track records such as financials, transactions, and employees. You can view all classes defined in NetSuite at Setup > Company > Classes.

You can use helper functions in a utility file to avoid adding the internal NetSuite IDs into the plug-in implementation logic. See Utility Files for a Custom GL Lines Plug-In Implementation.

Returns
int

Input Parameters
None.

Parent object
StandardLine, CustomLine

Example

```javascript
function customizeGlImpact(record, standardLines, customLines, book)
{
    ...
    var context = nlapiGetContext();
    var makeClassesMandatory = context.getPreference('classmandatory');
    var allowPerLineClasses = context.getPreference('classesperline');
    ...
    var newLine = customLines.addNewLine();

    if (makeClassesMandatory || allowPerLineClasses)
    {
        // can also optionally set class
        newLine.setClassId(standardLines.getLine(0).getClassId());
    }
    ...
}
```

getCodeAmount()

Function Declaration
string getCreditAmount()

Type
Object method

Description
Returns the credit amount for a StandardLine or CustomLine object. Use this method to define plug-in implementation functionality based on the credited amount for a GL impact line. See also setCreditAmount(credit).

Returns
string

Input Parameters
None.

Parent object
StandardLine, CustomLine

Example

```javascript
function customizeGlImpact(record, standardLines, customLines, book)
{
    ...
    var newLine = customLines.addNewLine();
    newLine.setCreditAmount(standardLines.getLine(0).getCreditAmount());
    newLine.setAccountId(standardLines.getLine(0).getAccountId());
}
```
getDebitAmount()

**Function Declaration**  
`string getDebitAmount()`

**Type**  
Object method

**Description**  
Returns the debit amount for a **StandardLine** or **CustomLine** object. Use this method to define plug-in implementation functionality based on the debited amount for a GL impact line. See also `setDebitAmount(debit)`.

**Returns**  
`string`

**Input Parameters**  
None.

**Parent object**  
**StandardLine**, **CustomLine**

**Example**

```javascript
function customizeGlImpact(record, standardLines, customLines, book)
{
    ...
    var newLine = customLines.addNewLine();
    newLine.setCreditAmount(standardLines.getLine(0).getCreditAmount());
    newLine.setAccountId(standardLines.getLine(0).getAccountId());
    newLine.setMemo("Payment catches both revenue and cash.");
    ...
    var newLine = customLines.addNewLine();
    newLine.setDebitAmount(standardLines.getLine(1).getDebitAmount());
    newLine.setAccountId(standardLines.getLine(1).getAccountId());
    newLine.setMemo("Payment catches both revenue and cash.");
    ...
}
```

getDepartmentId()

**Function Declaration**  
`int getDepartmentId()`

**Type**  
Object method

**Description**  
Returns the internal NetSuite ID for the department on a **StandardLine** or **CustomLine** object. Use this method to define plug-in implementation functionality based on the department for a GL impact line. See also `setDepartmentId(departmentId)`.

Departments are listed first on transactions, and are useful when designating transactions and employees as part of an internal team. You can view all departments defined in NetSuite at Setup > Company > Departments.
You can use helper functions in a utility file to avoid adding the internal NetSuite IDs into the plug-in implementation logic. See Utility Files for a Custom GL Lines Plug-In Implementation.

Returns  
int

Input Parameters  
None.

Parent object  
StandardLine, CustomLine

Example

```javascript
function customizeGlImpact(record, standardLines, customLines, book) {
    ...
    var context = nlapiGetContext();
    var makeDepartmentsMandatory = context.getPreference('deptmandatory');
    var allowPerLineDepartments = context.getPreference('deptsperline');
    ...
    var newLine = customLines.addNewLine();

    if (makeDepartmentsMandatory || allowPerLineDepartments) {
        // can also optionally set department
        newLine.setDepartmentId(standardLines.getLine(0).getDepartmentId());
    }
    ...
}
```

gentityId()

Function Declaration  
int getEntityId()

Type  
Object method

Description  
Returns the internal NetSuite ID for the entity on a StandardLine or CustomLine object

Returns  
int

Input Parameters  
None

Parent object  
StandardLine, CustomLine

Example

```javascript
function customizeGlImpact(record, standardLines, customLines, book) {
    var entityId = standardLines.getLine(1).getEntityId();
    var line = customLines.addNewLine()
    line.setAccountId(6)
    line.setDebitAmount(100)
    line.setEntityId(entityId)

    line = customLines.addNewLine()
    line.setAccountId(7)
    line.setCreditAmount(100)
```
getLocationId()

**Function Declaration**

```
int getLocationId()
```

**Type**

Object method

**Description**

Returns the internal NetSuite ID for the location on a **StandardLine** or **CustomLine** object. Use this method to define plug-in implementation functionality based on the location for a GL impact line. See also `setLocationId(locationId)`.

Use locations to track information about employees and transactions for multiple offices or warehouses. You can view all locations defined in NetSuite at Setup > Company > Locations.

You can use helper functions in a utility file to avoid adding the internal NetSuite IDs into the plug-in implementation logic. See **Utility Files for a Custom GL Lines Plug-In Implementation**.

**Returns**

- `int`

**Input Parameters**

None

**Parent object**

**StandardLine**, **CustomLine**

**Example**

```javascript
function customizeGlImpact(record, standardLines, customLines, book) {
    ... 
    var context = nlapiGetContext();
    var makeLocationsMandatory = context.getPreference('locmandatory');
    var allowPerLineLocations = context.getPreference('locsperline');
    ... 
    var newLine = customLines.addNewLine();
    ... 
    if (makeLocationsMandatory || allowPerLineLocations) {
        // can also optionally set location
        newLine.setLocationId(standardLines.getLine(0).getLocationId());
    }
    ... 
}
```

getMemo()

**Function Declaration**

```
string getMemo()
```

**Type**

Object method

**Description**

Returns the **Memo** field on a **StandardLine** or **CustomLine** object. The **Memo** field is populated from fields on the transaction, for example, the Description field for a line item on an Invoice transaction.

See also `setMemo(memo)`.
Returns string
Input Parameters None.
Parent object StandardLine, CustomLine

Example

function customizeGlImpact(record, standardLines, customLines, book)
{
    ...
    var newLine = customLines.addNewLine();
    newLine.setCreditAmount(standardLines.getLine(0).getCreditAmount());
    newLine.setAccountId(standardLines.getLine(0).getAccountId());
    // append old memo to new memo text
    newLine.setMemo("Payment catches both revenue and cash. " + standardLines.getLine(0).getMemo());
    ...
    var newLine = customLines.addNewLine();
    newLine.setDebitAmount(standardLines.getLine(1).getDebitAmount());
    newLine.setAccountId(standardLines.getLine(1).getAccountId());
    // append old memo to new memo text
    newLine.setMemo("Payment catches both revenue and cash. " + standardLines.getLine(1).getMemo());
    ...
}

getSegmentValueId(segmentId)

Function Declaration int getSegmentValueId(segmentId)
Type Object method
Description Returns the internal NetSuite ID for the custom segment value set on the line on a StandardLine or CustomLine object. Use this method to define plug-in implementation functionality based on the value for a GL impact line.
See also setSegmentValueId(segmentId, segmentValueId).
Returns int
Input Parameters segmentId (string) — String value of the custom segment ID
Parent object StandardLine, CustomLine
Example

function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    for (var i = 0; i < standardLines.getCount(); i++)
    {
        var line = standardLines.getLine(i);

        var costCenterId = line.getSegmentValueId('csegcostcenter');
        var workCenterId = line.getSegmentValueId('csegworkcenter');
...  
// do something with the values
}

var newLine = customLines.addNewLine();
// reading segment value from newly created line (value is null or copied from head)
var costCenterId = newLine.getSegmentValueId('csegcostcenter');
...

var newLine2 = customLines.addNewLine();
...
newLine2.setSegmentValueId('csegcostcenter', getCostCenterValueForItem(itemId));
if (newLine2.getSegmentValueId('csegcostcenter') == 1)
{
  ...
}
...

// Utility methods
function getCostCenterValueForItem(itemId)
{
  ...
  return valueId;
}
Custom GL Lines Plug-in Reference

Use the information in this section as reference when you develop, run, and test a Custom GL Lines plug-in implementation:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View example code</td>
<td>View example code for using utility files in a plug-in implementation. Use the utility files to create helper functions, reference internal NetSuite IDs, search records, and use with a custom record to store internal NetSuite record IDs.</td>
</tr>
<tr>
<td>Classification handling</td>
<td>You can set classifications, including department, class, and location, on custom lines for standard transactions, journals, and custom transactions. NetSuite may also set the classifications depending on the accounting preferences.</td>
</tr>
<tr>
<td></td>
<td>See Classifications.</td>
</tr>
<tr>
<td>Guidelines and best practices</td>
<td>Get more information about creating the plug-in implementation script file and plug-in implementation behavior.</td>
</tr>
<tr>
<td></td>
<td>See Custom GL Lines Plug-in Guidelines and Best Practices.</td>
</tr>
<tr>
<td>Get help on error messages</td>
<td>Get more information about error messages that can occur when NetSuite validates the plug-in implementation logic or during the plug-in implementation script execution.</td>
</tr>
<tr>
<td></td>
<td>See Error Messages for Custom GL Lines Plug-in.</td>
</tr>
<tr>
<td>Searching and reporting on custom lines</td>
<td>Use the Custom GL flag on reports and searches.</td>
</tr>
<tr>
<td></td>
<td>See Custom Lines on Reports and in Searches.</td>
</tr>
<tr>
<td>Use the audit log</td>
<td>Use the Custom GL Lines audit log to get more information about custom lines created by a plug-in implementation.</td>
</tr>
<tr>
<td></td>
<td>See Custom GL Lines Audit Log.</td>
</tr>
<tr>
<td>Use the plug-in system notes</td>
<td>Use the plug-in system notes to get more information about changes made on a plug-in implementation record.</td>
</tr>
<tr>
<td></td>
<td>See the help topic Viewing Plug-in Implementation System Notes.</td>
</tr>
<tr>
<td>Delete a plug-in implementation</td>
<td>Delete a Custom GL Lines plug-in from a NetSuite account, or uninstall a bundle that contains a plug-in implementation.</td>
</tr>
<tr>
<td></td>
<td>See Deleting a Custom GL Lines Plug-in Implementation.</td>
</tr>
</tbody>
</table>

Custom GL Lines Plug-in Example Code

Use the code examples in this section as a reference when you create a Custom GL Lines plug-in implementation.

The following table describes the examples:

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using utility files</td>
<td>Use utility JavaScript files to contain helper functions for the main plug-in implementation script file or store internal NetSuite IDs for NetSuite records.</td>
</tr>
<tr>
<td></td>
<td>See Utility Files for a Custom GL Lines Plug-In Implementation.</td>
</tr>
</tbody>
</table>
Utility Files for a Custom GL Lines Plug-In Implementation

You can create utility JavaScript files for use with a Custom GL Lines plug-in implementation. These utility files can contain the following types of logic:

- **Helper functions for use with the main plug-in implementation script file.** See Helper Functions Example.
- **Internal NetSuite IDs for NetSuite records.** See Internal NetSuite ID Example.

Depending on your plug-in implementation requirements, you may want to not include internal NetSuite IDs in the implementation and utility script files. In this case, you can create a custom record type to store references to the internal IDs for NetSuite records, and then retrieve the IDs via SuiteScript. For an example, see Using a Custom Record to Reference Internal NetSuite IDs Example.

**Helper Functions Example**

You can use helper functions similar to the examples below to avoid using NetSuite internal IDs in the plug-in implementation logic.

**Utility Javascript File**

```javascript
// search for entity's email
function findEntityEmail(entityId)
{
  var searchFilters = new Array();
  searchFilters[0] = new nlobjSearchFilter('internalid', null, 'anyof', entityId);
  var searchColumns = new Array();
  searchColumns[0] = new nlobjSearchColumn('email');

  var searchResults = nlapiSearchRecord('entity', null, searchFilters, searchColumns);
  var email = '';
  if (searchResults != null && searchResults.length > 0)
  {
    email = searchResults[0].getValue('email');
  }
  return email;
}

// search for entity's phone with saved search
function findEntityPhone(entityId)
{
  var searchFilters = new Array();
  searchFilters[0] = new nlobjSearchFilter('internalid', null, 'anyof', entityId);

  var savedSearchResults = nlapiSearchRecord('entity', 'customsearch_script_id', searchFilters);
  var phone = '';
  if (savedSearchResults != null && savedSearchResults.length > 0)
  {
    phone = savedSearchResults[0].getValue('phone');
  }
  return phone;
}
```
Custom GL Lines Plug-in Example Code

Plug-in Implementation Script File

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    var entityId = transactionRecord.getFieldValue('entity');
    var email = findEntityEmail(entityId);
    var phone = findEntityPhone(entityId);
    ...
}
```

Internal NetSuite ID Example

The interface for the Custom GL plug-in includes methods that you can use to retrieve the internal NetSuite IDs for NetSuite records related to the transaction. You can use these IDs to define plug-in implementation functionality or with SuiteScript APIs like `nlapiLoadRecord(type, id, initializeValues)` and `nlapiSearchRecord(type, id, filters, columns)`. However, these IDs are specific to a particular NetSuite account. You may want to include the account-specific IDs in a separate utility file so that they may be later customized if the plug-in implementation is installed in a different account.

The following example shows a helper function named `getAccountIDforCustomLine(accountId)` that returns the internal NetSuite ID for an account record. The account ID to return is included in the `getAccountIDforCustomLine(accountId)` function. The function is referenced in the main plug-in implementation script file.

```
function getAccountIDforCustomLine(accountId)
{
    ...
    if (accountID == 30) // main transaction account is 4000 - Sales (id: 30)
    {
        return 40; // custom line account is 1000 - VAT Expenses (id:40)
    } else {
        // return something else
    }
}
```

Note: This code is for demonstration purposes only. The `customizeGlImpact(transactionRecord, standardLines, customLines, book)` function does not add balanced custom lines and the transaction will not save.

Utility Javascript File

```
// Utility methods
function getAccountIDforCustomLine(accountId)
{
    ...
    if (accountID == 30) // main transaction account is 4000 - Sales (id: 30)
    {
        return 40; // custom line account is 1000 - VAT Expenses (id:40)
    } else {
        // return something else
    }
}
```

Plug-in Implementation Script File

```
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    var newLine = customLines.addNewLine();
    newLine.setDebitAmount(amount);
    newLine.setAccountId(getAccountIDforCustomLine(standardLines.getLine(0).getAccountId()));
    newLine.setMemo("Payment catches both revenue and cash.");
}
```
Using a Custom Record to Reference Internal NetSuite IDs Example

You can use a custom record type to store the references to the required NetSuite standard records and retrieve them via SuiteScript. For more information about creating custom record types, see the help topic Creating Custom Record Types in the SuiteBuilder documentation.

**Note:** The following custom record example and plug-in implementation code is for demonstration purposes only. Your implementation may differ depending on your requirements. You can use a similar solution to retrieve other IDs, like subsidiaries or entities. Also, you use a similar method to create a solution that is not language-dependent.

To store and retrieve references to internal NetSuite IDs, complete the following steps:

1. Create a custom record type with an ID of `customrecord_tutorial_account_config` and add a field that references a standard NetSuite record. For example, create a custom record type named **Account Configuration** to store references to the accounts required by the plug-in implementation. Then, create a field for the custom record type with the following configuration:

<table>
<thead>
<tr>
<th>Field Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Account</td>
</tr>
<tr>
<td>ID</td>
<td>_account</td>
</tr>
<tr>
<td>Type</td>
<td>List/Record</td>
</tr>
<tr>
<td>List/Record</td>
<td>Account</td>
</tr>
</tbody>
</table>

2. Create a new instance of the custom record. For this example, you create a custom record with a name of **Cash Basis Income Account** that references a NetSuite account that you want to use with a custom line in the plug-in implementation.

3. Create the plug-in implementation and utility script files. The helper function `findConfiguredAccount(name)` searches for the account name and returns the internal NetSuite account ID.

**Utility Javascript File**

```javascript
// Returns internal ID of account from a custom record with the name specified in parameter
function findConfiguredAccount(name)
{
    var searchFilters = new Array();
    searchFilters[0] = new nlobjSearchFilter('name', null, 'is', name);
    searchFilters[1] = new nlobjSearchFilter('isinactive', null, 'is', 'F');

    var searchColumns = new Array();
    searchColumns[0] = new nlobjSearchColumn('custrecord_account');

    var searchResults = nlapiSearchRecord('customrecord_tutorial_account_config', null, searchFilters, searchColumns);

    if (searchResults!=null && searchResults.length > 0)
    {
        return searchResults[0].getValue('custrecord_account');
    } else {
        return null;
    }
}
```
return searchResults[0].getValue('custrecord_account');
}
throw "Account " + name + " not found";

Plug-in Implementation Script File

function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
...
var accountId = findConfiguredAccount('Cash Basis Income Account');
line.setAccountId(accountId);
...

Classifications

You can use the CustomLine methods in a Custom GL Lines plug-in implementation to set department, class, and location classifications on custom lines. If you do not set classifications on custom lines, NetSuite may set the classifications, depending on the accounting preferences or the transaction form. Refer to the following sections for information about how NetSuite handles classifications for the following types of transactions:

- **Standard transactions.** Classifications are set on custom lines depending on the plug-in implementation behavior and the accounting preferences. See Classification Handling on Standard Transactions.
- **Journals.** In addition to the classification handling on standard transactions, NetSuite also respects the journal-specific accounting preferences when setting classifications on custom lines. See Classification Handling on Journals and Classification Handling on Standard Transactions.
- **Custom transactions.** NetSuite uses the preferences set on the custom transaction definition when setting classifications on custom lines. See Classification Handling on Custom Transactions.

Classification Handling on Standard Transactions

Setting department, class, and location classifications on custom lines for standard transactions depends on the plug-in implementation behavior and the accounting preferences.

For example, if classifications are mandatory, all custom lines inherit header classifications. Otherwise, if classifications can be set at the line level, the plug-in implementation can set the classifications. If the plug-in implementation does not set classifications, the classifications from the transaction header are applied to any custom line.

**Important:** These rules only apply for standard transactions. Custom transactions use the preferences set on the custom transaction definition. See Classification Handling on Custom Transactions.

For standard transactions, setting classifications on custom lines by NetSuite and the plug-in implementation depends on the following accounting preferences:

- Mandatory classifications and per-line classification preferences. You can set preferences that make the department, class, and location classification mandatory on transactions or set preferences that allow classifications to be set at the line-level. See the help topics General Accounting Preferences and Using Per-Line Classifications.
Level at which classifications display of transaction forms. If you have enabled preferences to use per-line classes, departments, or locations, you can also customize transaction forms to identify and show the classification at both the header and line level at the same time. See the help topic **Customizing Forms for Per-Line and Header Classifications**.

The following table describes how the department, class, and location classification handling depends on the accounting preferences:

<table>
<thead>
<tr>
<th>Show Classification Level</th>
<th>Allow Classifications at Line-Level</th>
<th>Plug-in Implementation Sets Classification</th>
<th>Result on Custom Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>No</td>
<td>Yes</td>
<td>Use classification from transaction header.*</td>
</tr>
<tr>
<td>Header</td>
<td>No</td>
<td>No</td>
<td>Use classification from transaction header.</td>
</tr>
<tr>
<td>Header</td>
<td>Yes</td>
<td>Yes</td>
<td>Use classification set by plug-in implementation.</td>
</tr>
<tr>
<td>Header</td>
<td>Yes</td>
<td>No</td>
<td>Use classification from transaction header.</td>
</tr>
<tr>
<td>Line</td>
<td>Yes</td>
<td>Yes</td>
<td>Use classification set by plug-in implementation.</td>
</tr>
<tr>
<td>Line</td>
<td>Yes</td>
<td>No</td>
<td>No classification.</td>
</tr>
<tr>
<td>None</td>
<td>No</td>
<td>Yes</td>
<td>No classification.</td>
</tr>
<tr>
<td>None</td>
<td>No</td>
<td>No</td>
<td>No classification.</td>
</tr>
<tr>
<td>None</td>
<td>Yes</td>
<td>Yes</td>
<td>Use classification set by plug-in implementation.</td>
</tr>
<tr>
<td>None</td>
<td>Yes</td>
<td>No</td>
<td>No classification.</td>
</tr>
</tbody>
</table>

* NetSuite overrides the plug-in implementation classification setting.

### Guidelines for Classifications Handling

Use the following guidelines when working with classifications for standard transactions and a Custom GL Lines plug-in implementation:

- The above classification handling only applies to custom lines. Standard lines use the value as determined by the accounting preferences and the values entered on the transaction form.
- If you enable the preference to make a classification mandatory and use a form customized to display a classification at both the header and line level, the classification is mandatory at both the header and line level.

### Classification Handling on Journals

The following table lists the journal-specific accounting preferences and classification handling on custom lines.

<table>
<thead>
<tr>
<th>Preference</th>
<th>Classification Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always Allow Per-line Classifications on Journals</td>
<td>Set line-item classes, departments, and locations on custom lines for journal entries.</td>
</tr>
</tbody>
</table>
### Classification Handling on Custom Transactions

The classifications that you can set on custom lines for a custom transaction depend on the configuration of the custom transaction type.

You can set the following preferences for departments, classes, and locations on a custom transaction:

<table>
<thead>
<tr>
<th>Preference</th>
<th>Custom Lines Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>No value is required for the department, class, or location on instances of the custom transaction.</td>
</tr>
<tr>
<td>Header</td>
<td>Specify the classification at the header level for instances of the custom transaction. All custom lines inherit this value. If you use a <code>CustomLine</code> method to set the classification value, NetSuite ignores the value and uses the header value on all custom lines.</td>
</tr>
<tr>
<td>Line</td>
<td>Specify the classification at the line-item level for all lines on the custom transaction. You must use a <code>CustomLine</code> method to set the value of the classification on each custom line created by the plug-in implementation. Otherwise, you cannot save the transaction.</td>
</tr>
</tbody>
</table>

The classification settings on a custom transaction override any general accounting preferences you specify at Setup > Accounting > Accounting Preferences. For example, you make the department classification required for transactions in the general accounting preferences, but set the `Department` option on a custom transaction type to `None`. Consequently, the `Department` classification is not required for instances of the custom transaction, overriding the general accounting preferences.

### Multiple Currencies Examples

When you are working with custom lines in an account with multiple accounting books or multiple currencies, NetSuite automatically performs currency conversion in the following situations:

- The transaction currency is different from the accounting book currency to which the GL impact posts. NetSuite performs a conversion to make standard lines appear in base currency. Custom lines do not require conversion, NetSuite creates the custom lines using the base currency.

  For example, a transaction uses EURO for the items on the transaction, but the accounting book uses GBP. NetSuite converts from EURO to GBP when calculating the GL impact, depending on the exchange rate in NetSuite or the exchange rate specified on the transaction.
For more information, see the help topic Currency Management.


For example, a primary accounting book uses USD and a secondary accounting book uses GBP, and you use setCreditAmount(credit) to set the amount of a custom line on a primary book with setBookSpecific(bookSpecific) set to false.

NetSuite converts the amount from USD to GBP when adding the custom line impact to the secondary book. For more information about the mapping of lines between primary and secondary accounting books by a Custom GL Lines plug-in implementation, see Custom GL Lines Plug-in Process Flow.

The currency to which NetSuite converts an amount depends on both the subsidiary and the accounting book currency. For example, a US company has two subsidiaries:

<table>
<thead>
<tr>
<th>Subsidiary Name</th>
<th>Secondary Book 1 Currency</th>
<th>Secondary Book 2 Currency</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td>USD</td>
<td>GBP</td>
</tr>
<tr>
<td>Canada</td>
<td>CAD</td>
<td>Euro</td>
</tr>
</tbody>
</table>

Any GL impact posts to the secondary accounting books based on the currency for the subsidiary accounting book. The GL impact displays in the currency for the accounting book, and not the currency that appears on the transaction for the transaction line items.

Converting Currency in a Custom GL Lines Plug-in Implementation

In general, you do not need to perform manual conversion of currencies if you are using the amounts of standard lines posted to a transaction to calculate the values of custom lines. However, if you want to use hard-coded currency values or retrieve an amount to use in a custom line from a source other than a standard GL impact line, you need to manually convert the currency amounts.

To perform the conversion, complete the following steps:

1. Get the base currency. You can search for the base currency or get the base currency from the accounting book record. See Find the Base Currency and Base Currency from Accounting Book Record.

2. Perform the conversion. Get the transaction-specific conversion rate from the accounting book record or convert using the nlapiExchangeRate API. See Transaction-Specific Exchange Rates or Convert Currencies with SuiteScript.

Find the Base Currency

Use the following function to find the base currency for the current accounting book using Search APIs:

```javascript
/* Returns internal id of base currency for specified subsidiary and accounting book combination */
function getBaseCurrency(subsidiaryId, bookId)
{
    var searchFilters = new Array();
    searchFilters[0] = new nlobjSearchFilter('internalid', null, 'is', subsidiaryId);
    searchFilters[1] = new nlobjSearchFilter('accountingbook', null, 'is', bookId);
    // Additional filters for subsidiary and accounting book currency
    return nlobjSearch.execute('nlobj_Currency', searchFilters);
}
```
Multiple Currencies Examples

var searchColumns = new Array();
searchColumns[0] = new nlobjSearchColumn('accountingbookcurrency');

var searchResults = nlapiSearchRecord('subsidiary', null, searchFilters, searchColumns);

if (searchResults!=null && searchResults.length > 0)
{
    return searchResults[0].getValue('accountingbookcurrency');
}

throw "Subsidiary " + subsidiaryId + " or accounting book " + bookId + " not found";

Base Currency from Accounting Book Record

Depending on the transaction type, you can get the base currency from the transaction record. For example, a Cash Sale record contains information about the accounting book and the associated base currency for the transaction:

Use the following example code to find the base currency for the current accounting book using the Record object:

function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    var bookId = book.getId();
    var baseCurrency;
    for (var line = 1; line <= transactionRecord.getLineItemCount('accountingbookdetail'); line++)
    {
        if (bookId == transactionRecord.getLineItemValue('accountingbookdetail','bookid',line))
        {
            baseCurrency = transactionRecord.getLineItemValue('accountingbookdetail','currency',line);
            break;
        }
    }
    ...
}

Transaction-Specific Exchange Rates

If a user can define transaction-specific exchange rates for the accounting book on a transaction, you can use the conversion rate for the accounting book to perform the conversion.
The following screenshot shows the Exchange Rate on the Accounting Books subtab of a Cash Sale transaction:

![Exchange Rate on Accounting Books](image)

Use the following example code to find the exchange rate by accessing the exchange through the Record object:

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    var amountToAdd = 20; // Amount in Primary book's currency

    if (!book.isPrimary()) // Amount is in Primary book's currency, convert only for non-primary books
    {
        var bookId = book.getId();

        for (var line = 1; line <= transactionRecord.getLineItemCount('accountingbookdetail'); line++)
        {
            if (bookId == transactionRecord.getLineItemValue('accountingbookdetail', 'bookid', line))
            {
                var rate = transactionRecord.getLineItemValue('accountingbookdetail', 'exchangerate', line);
                amountToAdd = amountToAdd * rate;
                break;
            }
        }
    }

    var newLine = customLines.addNewLine();
    newLine.setDebitAmount(amountToAdd);
    ...
}
```

Convert Currencies with SuiteScript

The Exchange Rate column of the Currency Exchange Rates record displays the company-wide exchange rates. To access the Currency Exchange Rates record, go to Lists > Accounting > Currencies.

You can access these values with the `nlapiExchangeRate(sourceCurrency, targetCurrency, effectiveDate)` API.

Use the following example code to perform a conversion using the company-wide exchange rates:

```javascript
function customizeGlImpact(transactionRecord, standardLines, customLines, book)
{
    ...
    var amountToAdd = 20; // USD
    var amountCurrency = 1; // internal id of USD currency in their account
```
Custom GL Lines Plug-in Guidelines and Best Practices

The following table describes guidelines and best practices for working with the Custom GL Lines plug-in:

<table>
<thead>
<tr>
<th>Guidelines</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verify environment</td>
<td>Perform the following checks to verify the environment and define plug-in implementation behavior based on the results:</td>
</tr>
<tr>
<td></td>
<td>- Transaction type. Check the transaction type. Use <code>getRecordType()</code>. For an example, see the example for <code>getEntityId()</code>.</td>
</tr>
</tbody>
</table>
|                       |   - Multi-Book Accounting feature and primary book. Check if the Multi-Book Accounting feature is enabled and if the current book processed by the plug-in implementation is the primary accounting book. See `isPrimary()`.
|                       |   - Classification settings. Check the accounting preferences for department, class, and location in the current account. Use `nlapiGetContext()`, `getDepartmentId()`, `getClassId()`, or `getLocationId()`. |
|                       | You can use this information to define the plug-in behavior for the types of transactions configured for the implementation and how the custom lines should be created. |
| Account and amount    | At minimum, each custom line created by the plug-in implementation must contain an account ID and an amount for the custom line. See `setAccountId(accountId)` and one of the following methods: `setCreditAmount(credit)` or `setDebitAmount(debit)`. |
| Utility files         | Use utility files to include helper functions that assist in defining plug-in implementation behavior. See Utility Files for a Custom GL Lines Plug-In Implementation. |
| Validations           | NetSuite performs the following validations on the logic in the plug-in implementation script file when the script runs:                                          |
|                       |   - Input to methods that set the custom line amount is not non-numerical.                                                                                     |
|                       |   - Script file does not access non-existent standard lines.                                                                                                |
|                       |   - Each custom line contains an amount and an account ID.                                                                                                     |
### Custom GL Lines Plug-in Guidelines and Best Practices

<table>
<thead>
<tr>
<th>Guidelines</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom GL Lines Plug-in</td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Statistical account IDs are not used for account IDs.</td>
<td></td>
</tr>
<tr>
<td>IDs for accounts, departments, classes, and locations exist in the current NetSuite account.</td>
<td></td>
</tr>
<tr>
<td>Debits and credits on the GL impact lines balance.</td>
<td></td>
</tr>
<tr>
<td>Currency type for the account matches the base currency for the subsidiary entity or matches the currency for the transaction.</td>
<td></td>
</tr>
<tr>
<td><strong>Transaction must balance</strong></td>
<td>The total amount of debits and credits for the transaction must balance.</td>
</tr>
<tr>
<td>Each plug-in implementation only runs on the subsidiary for the current entity. You cannot distribute credits and debits across subsidiaries.</td>
<td></td>
</tr>
<tr>
<td><strong>Using the main line</strong></td>
<td>The first line in the <code>StandardLines</code> object, Line 0, is a summary line that contains the total amount of credits and debits for the transaction.</td>
</tr>
<tr>
<td>The account ID for the main line depends on the transaction properties, line items, and accounting book. You can use this account ID when setting the account ID for a new custom line.</td>
<td></td>
</tr>
<tr>
<td><strong>Creating new versions</strong></td>
<td>To update an existing, installed version of a plug-in implementation, you should create a new implementation of the plug-in with a different name, instead of pushing an update to the existing plug-in implementation.</td>
</tr>
<tr>
<td><strong>Searching instead of loading records</strong></td>
<td>You can use SuiteScript API functions in a plug-in implementation, for example, <code>nlapiLoadRecord(type, id, initializeValues)</code> and <code>nlapiSearchRecord(type, id, filters, columns)</code>.</td>
</tr>
<tr>
<td>However, using these APIs may negatively affect the performance of the plug-in implementation. Limit the use of these APIs, where possible, to improve performance of the plug-in implementation. In general, searching for records yields better performance than loading NetSuite records.</td>
<td></td>
</tr>
<tr>
<td><strong>Creating records or performing data processing</strong></td>
<td>You should limit the logic of a plug-in implementation to the features required to create custom lines on the GL impact.</td>
</tr>
<tr>
<td>Additional functionality, for example, creating records or performing other complex data processing operations, should not be added to the plug-in implementation script file.</td>
<td></td>
</tr>
<tr>
<td><strong>Using the <code>getCount()</code> method to return the number of visible lines</strong></td>
<td>By default, the <code>getCount()</code> method returns the number of visible and hidden lines on the GL Impact page.</td>
</tr>
<tr>
<td>If you want to find only the number of visible lines, use the following condition.</td>
<td></td>
</tr>
</tbody>
</table>
| ```javascript
if((debitAmount !=0 || creditAmount !=0) && accountId != null) {  
  // normal visible lines
}
```                                                                                                                                                                                                                                                                                                                                 |
| Or use the following shorter version.                                                                                                                                                                                                                                                                                                        |
| ```javascript
if((debitAmount || creditAmount) && accountId) {  
  // normal visible lines
}
```                                                                                                                                                                                                                                                                                                                                 |
| **Governance Limit** | The plug-in script file allows up to 1000 usage units. For more information, see the help topic Script Usage Unit Limits.                                                                                                                                                                                                                     |
Error Messages for Custom GL Lines Plug-in

The following errors can occur when NetSuite validates the plug-in implementation logic before the script executes or during the plug-in implementation script execution.

Custom GL Lines Plug-in error due to an invalid script operation. Message: Cannot Parse Value: \{amount\}

**Cause:**
You used `setDebitAmount(debit)` to add an amount to a custom line. However, the amount is NULL or not a number.

**Action:**
Verify that the parameter value for the method is a non-null number in the plug-in implementation script file or utility file.

Custom GL Lines Plug-in error due to an invalid script operation. Message: Cannot use \{amount\} as input to `setDebitAmount(debit)`. Amount to debit must be positive

**Cause:**
You used `setDebitAmount(debit)` to add a debit amount to a custom line. However, the amount is negative.

**Action:**
Use a positive value for `setDebitAmount(debit)`.

Custom GL Lines Plug-in error due to an invalid script operation. Message: Line with index \{index\} does not exist. Total line count is: \{lineCount\}

**Cause:**
You used `getLine(index)` to get a standard or custom line. However, the value of the index parameter is equal or greater than the total number of lines in the `StandardLine` or `CustomLine` object.

**Action:**
Make sure the index parameter for `getLine(index)` references a valid element. You can use `getCount()` to get the total number of lines.

Custom GL Lines Plug-in error due to a failed validation of script output: Account cannot be empty

**Cause:**
You created a new `CustomLine` object with `addNewLine()`, but did not set an account for the line with `setAccountId(accountId)`.

All custom lines must debit or credit an account.

**Action:**
Make sure you set an account for all `CustomLine` objects.

Custom GL Lines Plug-in error due to a failed validation of script output: Debit/Credit cannot be empty

**Cause:**
You created a new `CustomLine` object with `addNewLine()`, but did not set an amount for the line with `setCreditAmount(credit)` or `setDebitAmount(debit)`.

All custom lines must include a debit or credit amount.

**Action:**
Make sure you set an amount for all `CustomLine` objects.

Custom GL Lines Plug-in error due to a failed validation of script output: Statistical account \{id\} was used in financial journal entry. Statistical accounts are not allowed in financial journal entries.

**Cause:**
The transaction is a Journal and you created a new `CustomLine` object with `addNewLine()`. However, you used a statistical account id for `setAccountid(accountId)`.
Action: Make sure the id for the account is not for a statistical account. Statistical accounts appear on the Chart of Accounts page at Setup > Accounting > Chart of Accounts with a type of Statistical.

Custom GL Lines Plug-in error due to a failed validation of script output: Account does not exist, id: {list of accounts not found}

Cause: You used setAccountId(accountId) to set an account for a custom line. However, the account id does not exist in the current account. For convenience, the error message lists all account IDs used by the plug-in implementation that do not exist.

Action: Make sure the account id used for setAccount(accountId) exists. You can view valid account IDs in the Internal ID column on the Chart of Accounts page at Setup > Accounting > Chart of Accounts.

Custom GL Lines Plug-in error due to a failed validation of script output: Class does not exist, id: {list of classes not found}

Cause: You used setClassId(classId) to set a class classification for a custom line. However, the class id does not exist in the current account. For convenience, the error message lists all class IDs used by the plug-in implementation that do not exist.

Action: Make sure the class id used for setClassID(classID) exists. You can view valid class IDs in the Internal ID column on the Classes page at Setup > Company > Classes.

Custom GL Lines Plug-in error due to a failed validation of script output: Department does not exist, id: {list of departments not found}

Cause: You used setDepartmentId(departmentId) to set a department classification for a custom line. However, the department id does not exist in the current account. For convenience, the error message lists all department IDs used by the plug-in implementation that do not exist.

Action: Make sure the department id used for setDepartmentID(departmentID) exists. You can view valid department IDs in the Internal ID column on the Departments page at Setup > Company > Departments.

Custom GL Lines Plug-in error due to a failed validation of script output: Location does not exist, id: {list of locations not found}

Cause: You used setLocationId(locationId) to set a location classification for a custom line. However, the location ID does not exist in the current account. For convenience, the error message lists all location IDs used by the plug-in implementation that do not exist.

Action: Make sure the location ID used for setLocationID(locationID) exists. You can view valid location IDs in the Internal ID column on the Locations page at Setup > Company > Locations.

Custom GL Lines Plug-in error due to a failed validation of script output: Transaction was not in balance. Total = {amount}

Cause: A Custom GL Lines plug-in implementation creates custom lines for a transaction, but the total number of debits and credits for the custom lines in the transaction do not balance.
Custom GL Lines Plug-in error due to a failed validation of script output: Transaction and foreign currency account use different currencies. Account internal ID: {id}

**Cause:**
You used `setCreditAmount(credit)` or `setDebitAmount(debit)` to add an amount to a custom line.

However, the currency type for the account does not match the base currency for the subsidiary entity or match the currency for the transaction.

**Action:**
Choose an account with a currency that matches the base currency for the subsidiary entity or the currency for the transaction.

---

Custom Lines on Reports and in Searches

You can use the **Custom GL** flag in the Report Builder and in searches to view custom lines created by a Custom GL Lines plug-in implementation. See one of the following topics:

- Custom GL Lines on NetSuite Reports
- Searches for Custom GL Lines

**Note:** You can still view custom lines on reports and search for custom lines even if you delete the plug-in implementation that created the lines. See Deleting a Custom GL Lines Plug-In Implementation.

---

Custom GL Lines on NetSuite Reports

You can view general ledger impact for custom lines created by a Custom GL Lines plug-in implementation on NetSuite reports. In the Report Builder, you can add **Custom GL** as a column or as a filter to view report data for custom lines. For more information about customizing reports in the Report Builder, see the help topic Report Customization.

---

Custom GL Report Filter

The following screenshot shows the Custom GL filter in the Report Builder for a General Ledger report:
Custom GL Report Column

The following screenshot shows the Custom GL report column in the Report Builder for a General Ledger report:

Searches for Custom GL Lines

You can use the Custom GL filter on searches to search for transactions that do or do not have custom lines created by a Custom GL Lines plug-in implementation. You can use this filter on searches of type Transaction. You can also use this filter on searches of type Multi-Book Accounting Transaction, if you have the Multi-Book Accounting feature enabled.
The following screenshot shows the configuration of the Custom GL filter on a search:

![Custom GL filter configuration](image)

For more information about using search in NetSuite, see the help topic Running Searches.

**Custom GL Lines Audit Log**

The Custom GL Lines Audit Log shows custom lines created by Custom GL Lines plug-in implementations in a NetSuite account. Use the audit log to view any custom lines created for a specific transaction, all transactions, or for a specific plug-in implementation. To view the Custom GL Lines audit log page, you must have either administrator or Custom GL Lines Plug-in Audit Log permissions. To view the Custom GL Lines segments audit log page, you must have either administrator or Custom GL Lines Plug-in Audit Log (Segments) permissions.

See the following topics for more information:
- Accessing the Custom GL Lines Audit Log
- Custom GL Lines Audit Log Page

**Note:** Custom lines remain in the audit log even if you delete the plug-in implementation that created the lines. See Deleting a Custom GL Lines Plug-in Implementation.

**Accessing the Custom GL Lines Audit Log**

You can access the Custom GL Lines Audit Log using the following methods:

- GL Impact page. On a transaction in View mode, click Actions > GL Impact. For any custom lines on the transaction, click the name of the plug-in implementation in the **Custom Script** column:

  ![GL Impact page](image)

- Use a search. Create a search with the **Custom GL Lines Plug-in Audit Log** search type:
Important: You must have administrator or Custom GL Lines Plug-in Audit Log permissions to view or search the Custom GL Lines audit log.

You can also view system notes for plug-in implementations. For more information, see the help topic Viewing Plug-in Implementation System Notes.

Custom GL Lines Audit Log Page

The Custom GL Lines Audit Log page contains details about the custom lines that the plug-in created and attempted to create for a transaction. If failed validations occur, they are also logged to help you determine why the script failed. Use the filters and the links available in the audit log columns to get more information about the custom lines.

The following table describes the available filters for working with the audit log:

<table>
<thead>
<tr>
<th>Filter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Transaction number to display.</td>
</tr>
<tr>
<td>Plug-in Implementation</td>
<td>Plug-in implementations and other custom script files available in the account. By default, this list includes all custom script files in the NetSuite account with script IDs that begin with customscript.</td>
</tr>
<tr>
<td>Current</td>
<td>Includes only custom lines that currently affect transactions in the NetSuite account. Default is Yes. Select No or All to include only custom lines for deleted transaction or custom lines that the plug-in implementation failed to create due to an error.</td>
</tr>
<tr>
<td>Style</td>
<td>View the custom lines as a report or in a grid.</td>
</tr>
</tbody>
</table>

The following table describes the columns available on the default view of the Custom GL Lines audit log:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Entry No.</td>
<td>Internal ID of the custom lines in the audit log.</td>
</tr>
<tr>
<td>Transaction Type</td>
<td>Type of transaction on which the custom lines were created.</td>
</tr>
<tr>
<td></td>
<td>This column can equal Nonexistent if one of the following conditions are met:</td>
</tr>
</tbody>
</table>
## Custom GL Lines Audit Log

The following table describes the columns available on the default view of the Custom GL Lines Audit Log Transaction Details page:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Record number for the transaction in NetSuite. Click the link to open the transaction record.</td>
</tr>
<tr>
<td>Lines Added</td>
<td>Number of custom lines that the plug-in implementation created for the transaction. Click the link to open the Custom GL Lines Audit Log Transaction Details Page.</td>
</tr>
<tr>
<td>Revision</td>
<td>Revision of the plug-in implementation that created the custom lines. Click the link to open the Custom GL Lines Plug-in Revision Page.</td>
</tr>
<tr>
<td>Error</td>
<td>Any errors that occur when NetSuite validates the plug-in implementation logic before the script executes or during the plug-in implementation script execution.</td>
</tr>
</tbody>
</table>

**Note:** If an error appears in this column, the Transaction Type for the custom line will be **Nonexistent**. If there is no error, the plug-in implementation successfully created the custom lines.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed Time (ms)</td>
<td>Amount of time, in milliseconds, that the plug-in implementation spent to create all custom lines for the transaction.</td>
</tr>
<tr>
<td>Date</td>
<td>Date and time the custom lines were created.</td>
</tr>
</tbody>
</table>

The following screenshot shows the Custom GL Lines Audit Log page:

The Custom GL Lines Audit Log Segments page shows the segments that were set by a specific transaction. For information, see Custom GL Lines Audit Log Segments Page.

You can also view system notes for plug-in implementations. For information, see the help topic Viewing Plug-in Implementation System Notes.

### Custom GL Lines Audit Log Transaction Details Page

The Custom GL Lines Audit Log Transaction Details page shows the balancing custom lines added to a specific transaction. Click the link in the Lines Added column on the Custom GL Lines Audit Log Page to access the Custom GL Lines Audit Log Transaction Details page.

The following table describes the columns available on the default view of the Custom GL Lines Audit Log Transaction Details page:
<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Entry No.</td>
<td>Internal ID of the custom lines in the audit log. This is the same ID as Log Entry No. on the Custom GL Lines Audit Log Page. Click the link to navigate to the audit log page for the custom line.</td>
</tr>
<tr>
<td>Account ID</td>
<td>Internal ID of the account that the custom line credited or debited. This ID appears on the Chart of Accounts page at Setup &gt; Accounting &gt; Chart of Accounts.</td>
</tr>
<tr>
<td>Credit *</td>
<td>Amount of credit for the custom line. This value is set either by the plug-in implementation itself when it maps custom lines to secondary books, if applicable, or with the setCreditAmount(credit) method.</td>
</tr>
<tr>
<td>Debit *</td>
<td>Amount of debit for the custom line. This value is set either by the plug-in implementation itself when it maps custom lines to secondary books, if applicable, or with the setDebitAmount(debit) method.*</td>
</tr>
<tr>
<td>Memo</td>
<td>Text added to the Memo field on the custom line. This value is set with the setMemo(memo) method by the plug-in implementation.</td>
</tr>
<tr>
<td>Class ID</td>
<td>Class ID for the custom line set with the setClassId(classId) method by the plug-in implementation.</td>
</tr>
<tr>
<td>Department ID</td>
<td>Department ID for the custom line set with the setDepartmentId(departmentId) method by the plug-in implementation.</td>
</tr>
<tr>
<td>Location ID</td>
<td>Location ID for the custom line set with the setLocationId(locationId) method by the plug-in implementation.</td>
</tr>
<tr>
<td>Total Segments Set</td>
<td>Number of custom segments that were set by the plug-in implementation.</td>
</tr>
<tr>
<td>Entity ID</td>
<td>Entity ID for the custom line set with the setEntityId(entityId) method by the plug-in implementation.</td>
</tr>
</tbody>
</table>

* The currency for the amount depends on the transaction currency and the base currency for the accounting book and subsidiary. See Multiple Currencies Examples.

The following screenshot shows the Custom GL Lines Audit Log Transaction Details page:
Custom GL Lines Audit Log Segments Page

The Custom GL Lines Audit Log Segments page shows the segments that were set by a specific transaction. Click the link in the Total Segments Set column of the Custom GL Lines Audit Log Transaction Details page to access the Custom GL Lines Audit Log Segments page. Administrator or Custom GL Lines Plug-in Audit Log (Segments) permissions are required to view the Custom GL Lines Audit Log Segments page.

The following table describes the columns available on the default view of the Custom GL Lines Audit Log Segments page:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Entry No.</td>
<td>Internal ID of the custom line in the audit log. This is the same ID as the Log Entry No. column on the Custom GL Lines Audit Log Page. Click the link to navigate to the audit log page for the custom line.</td>
</tr>
<tr>
<td>Line</td>
<td>Line number of the custom GL line.</td>
</tr>
<tr>
<td>Segment ID</td>
<td>ID for the custom segment definition.</td>
</tr>
<tr>
<td>Segment</td>
<td>Description of the custom segment definition.</td>
</tr>
<tr>
<td>Segment Value ID</td>
<td>ID of the segment value for the custom line set with the setSegmentValueId(segmentId, segmentValueId) method by the plug-in implementation. For more information, see setSegmentValueId(segmentId, segmentValueId).</td>
</tr>
<tr>
<td>Segment Value</td>
<td>Custom segment value.</td>
</tr>
</tbody>
</table>

The following screenshot shows the Custom GL Lines Audit Log Segments page:

Custom GL Lines Plug-in Revision Page

The Custom GL Lines Plug-in Revision page shows the specific version of the plug-in implementation and associated library files that created a custom line. Use this page to determine the versions of the plug-in implementation script file and library file used to create the custom lines.

Click the link in the Revision column on the Custom GL Lines Audit Log Page to access this page.

The following table describes the columns available on the default view of the Custom GL Lines Plug-in Revision page:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plug-in Implementation</td>
<td>Name of the plug-in implementation that created the custom lines.</td>
</tr>
</tbody>
</table>
### Custom GL Lines Audit Log

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
<td>File name of the plug-in implementation script file. For more information about this file, see Create a Plug-in Implementation Script File.</td>
</tr>
<tr>
<td>File Contents</td>
<td>Snapshot of the plug-in implementation script file contents. Note: This information is extracted from the NetSuite database and is for reference only. Consequently, you cannot click (more...) to see the rest of the file contents. To view the complete file content, open the Script record for the plug-in implementation script file.</td>
</tr>
<tr>
<td>Modified</td>
<td>Date and time the plug-in implementation script file was last modified.</td>
</tr>
<tr>
<td>Modified By</td>
<td>User account that last modified the plug-in implementation script file.</td>
</tr>
<tr>
<td>Libraries</td>
<td>Associated library file names for the plug-in implementation that were specified during creation of the plug-in implementation. All library files are listed. For more information, see Add the Plug-in Implementation.</td>
</tr>
<tr>
<td>Libraries Modified</td>
<td>Latest date and time when any one of the libraries for the plug-in implementation was last modified.</td>
</tr>
</tbody>
</table>

The following screenshot shows the Custom GL Lines Plug-in Revision page:

![Custom GL Lines Plug-in Revisions](image)

**Reviewing Custom GL Lines Plug-in Executions**

Custom GL line plug-ins might be set to run asynchronously in the following instances:

- If you have multiple custom GL line plug-in implementations that cause long transaction save times, some might be configured to run asynchronously.
- If you use both the Custom GL Lines Plug-in and Inventory features, when the cost of goods sold (COGS) changes, cost amounts are updated. The Custom GL Lines Plug-in runs asynchronously to update the GL impact in the transactions.

When a transaction is updated by the COGS engine, the GL impact is updated. Then, the Custom GL Lines Plug-in implementations on the transaction are triggered. After a change causes the COGS update, the Custom GL Lines Plug-in process runs to update the GL impact a second time. When you view the GL Impact page, you might see a message at the top of the page to indicate that the GL impact might change. This message appears if an update process is scheduled or in progress.

To address any exceptions that occurred when a custom GL line plug-in was run asynchronously, the Review Custom GL Plug-in Executions page displays Incomplete and Failed executions. You must have the administrator or the Review Custom GL Plug-ins Executions permission to access the Review...

![Image of Review Custom GL Plug-in Executions page]

The Review Custom GL Plug-in Executions page displays incomplete and failed Custom GL Plug-in executions from subsidiaries and accounting books to which you have access. Transactions appear on this page only if the last run of the custom GL plug-in was incomplete or failed.

By default, failed or incomplete transactions for all accounting periods are displayed. Select the accounting period as required. To include closed periods in the Accounting Period list, check the Include Closed Periods box.

Each task listed displays the transaction ID, transaction type, accounting period, and execution date. An incomplete task has at least one pending or running task for the custom GL plug-in in the selected period.

**Note:** If you update a transaction in a closed period that triggers the Custom GL Lines Plug-in, an error appears. You can either ignore the error, or open the period to run the plug-in again.

For any failed executions, you can select the item and then click Execute Selected to run the plug-in again. The page is refreshed, the status of the line becomes incomplete, and the line is displayed on the Incomplete subtab.

You can ignore an error, but if you decide to run the associated custom GL lines plug-in in the future, you have to edit the transactions manually. To ignore an error, check the line and click Ignore Selected. A confirmation message appears. Click Yes to confirm and remove the transaction from the list.

To view information about custom GL line plug-in executions that completed successfully, see the Custom GL Lines Audit Log Page.

### Deleting a Custom GL Lines Plug-in Implementation

If you delete a Custom GL Lines plug-in implementation or uninstall a bundle that contains a plug-in implementation, the removal has the following effects in the NetSuite account:

- Custom lines created by the plug-in implementation remain in the NetSuite database. Consequently, the custom lines remain linked to the original transaction and still appear in the GL impact for the transactions on which they were created and remain on any reports which lists the GL impact of the transaction. You can also still search for the custom lines. See Custom Lines on Reports and in Searches.
- The custom lines remain in the Custom GL Lines audit log. See Custom GL Lines Audit Log.
To delete a Custom GL Lines plug-in implementation:

1. Go to Customization > Plug-ins > Manage Plug-ins.
2. On the Manage Plug-ins page, disable the implementation that you want to delete and click Save.
4. On the Plug-in Implementations page, click the Edit link next to the plug-in implementation.
5. On the Plug-in Implementation page, click Actions > Delete. Click OK to confirm the deletion.

Note: If you uninstall a bundle that contains a plug-in implementation, the script file and any library files are also removed.