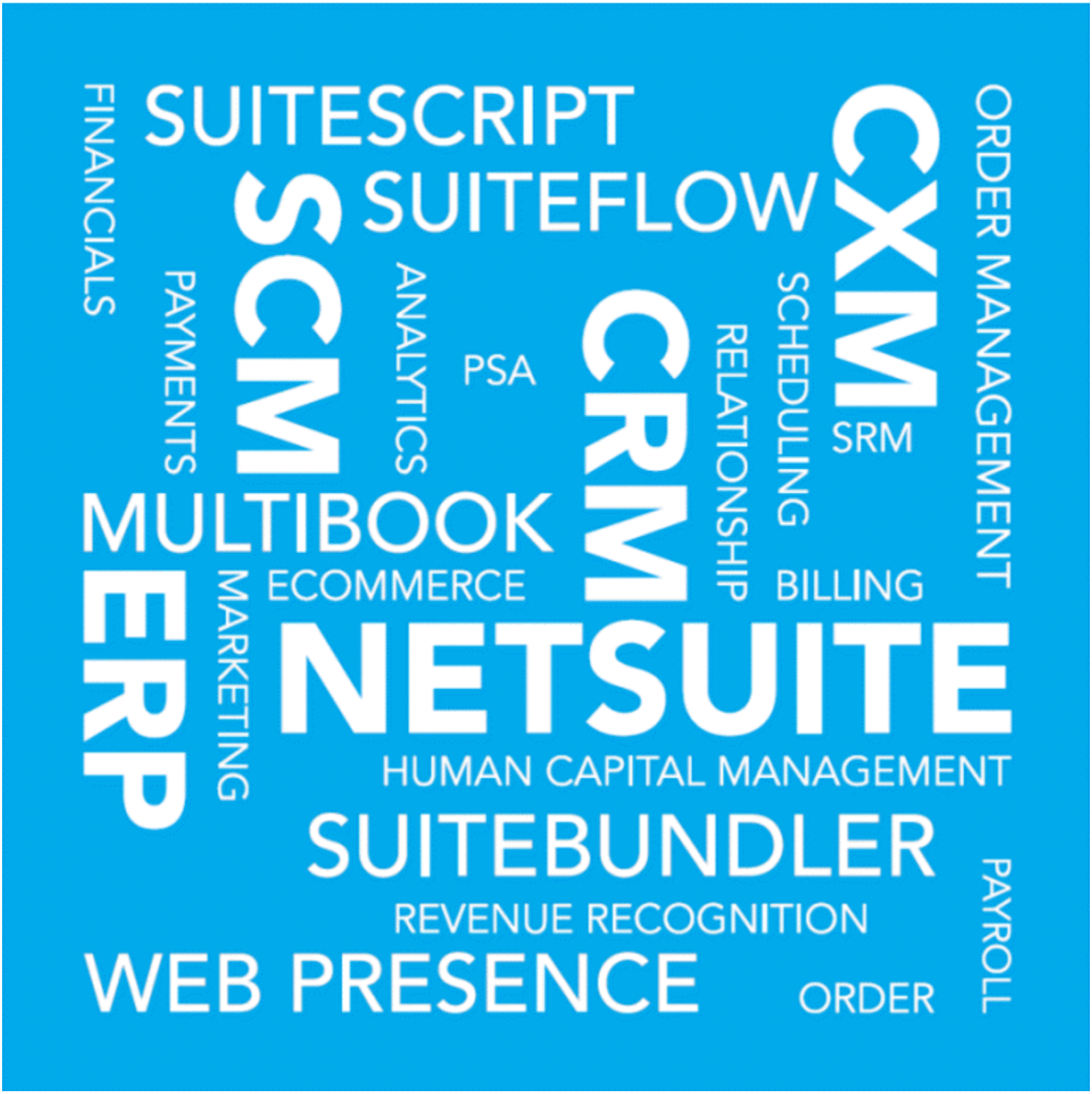
SuiteTax Plug-In for SuiteScript 2.0



Pre-General Availability

March 4, 2020

2019.2

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

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

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

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

Sample Code

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

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

No Excessive Use of the Service

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

Beta Features

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

Send Us Your Feedback

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

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

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

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

To report software issues, contact NetSuite Customer Support.

Table of Contents

SuiteTax Plug-in Overview

Use the SuiteTax plug-in to define tax engines that connect to third-party systems and calculate taxes for transactions in NetSuite. The tax engine that calculates the taxes on a transaction is defined by the tax registration of the transaction. The tax engine associated with the nexus represents an implementation of the SuiteTax plug-in. The plug-in implementation defines the process by which the tax engine calculates taxes on a transaction. You can use one plug-in implementation per nexus or use a single plug-in implementation with multiple nexuses.

To create and use a tax engine, you must first define the nexus and the associated tax types and tax codes in NetSuite. Then, a developer creates an implementation of the SuiteTax plug-in. Only Tax Engine type plug-in implementations which have been enabled in the NetSuite account can be selected as a tax engine for a subsidiary. When you edit the tax registrations of a subsidiary, you specify a tax engine for the nexus. Any transactions that correspond to the nexus use the tax engine to calculate the applicable taxes for the transaction.

You can calculate taxes on a transaction-level basis or on a line item-level basis. NetSuite passes transaction and line item data to the plug-in implementation. A developer uses this data to define the behavior of the plug-in implementation and the algorithm that defines how taxes are calculated for a transaction in the associated nexus. For example, a plug-in implementation can use the **Bill From** or **Bill To** values and the total amount of the transaction to calculate taxes based on the address and amount properties.

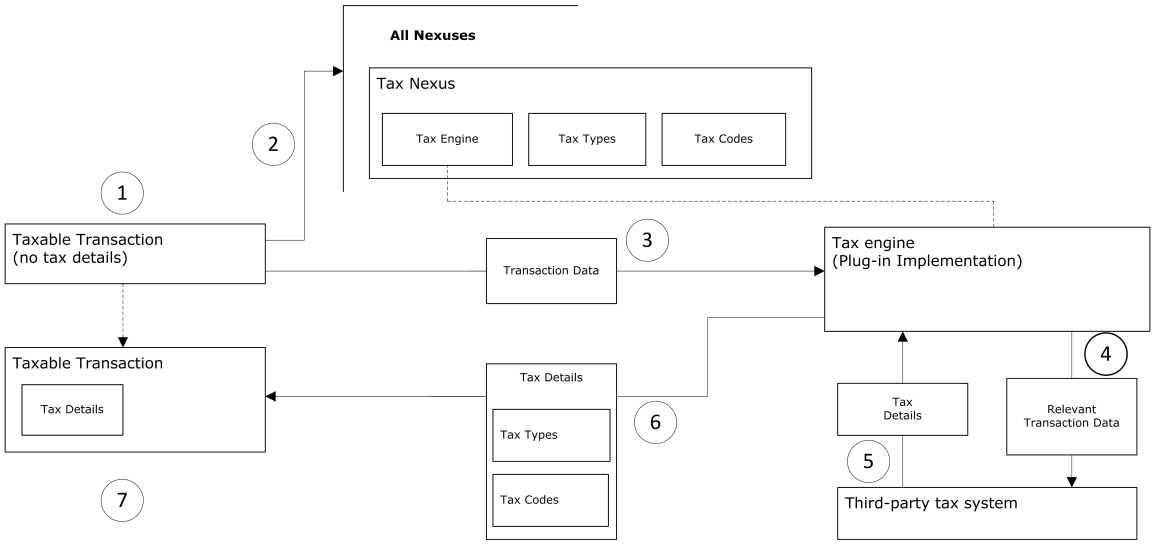
Depending on the design of the plug-in implementation, the implementation can define the transaction and line item data on which taxes are calculated and send a request to a third-party tax system to calculate the tax. The tax details are then sent back to NetSuite by the plug-in implementation and displayed or saved on the transaction record.

For information about the SuiteTax plug-in, see the following topics:

NetSuite Role	For more information, see ...
All roles	SuiteTax Plug-in Features in NetSuite
Developer	Developing a SuiteTax Plug-in Implementation
Administrator	Administering a SuiteTax Plug-in Implementation

Tax Calculation Process

The following figure describes the process for calculating taxes with a SuiteTax plug-in implementation:



1. **User requests tax calculation.** A user clicks **Preview Tax** on a taxable transaction or saves a taxable transaction. If tax has not yet been calculated, no taxes appear on the transaction before this operation.
2. **NetSuite looks up nexus and tax engine.** NetSuite looks up the appropriate nexus for the transaction and identifies the corresponding plug-in implementation specified as the tax engine for the nexus.
3. **NetSuite passes the transaction data.** NetSuite passes the transaction and line item data for the transaction to the plug-in implementation.
4. **Plug-in implementation processes the transaction data.** The logic defined for the plug-in implementation processes the transaction data and sends the appropriate data in a request to an external third-party tax calculation system.
5. **Tax system sends response.** The external third-party system calculates the tax details and passes the data back to the plug-in implementation.
6. **Plug-in implementation sends tax details back to NetSuite.** The tax details must include the tax summary for each tax type and tax code and can also include tax details for each individual line item.
7. **NetSuite displays or saves the tax details.** The transaction in NetSuite includes the tax summary information in the Summary box and can also include tax details for each individual line item on the **Tax Details** subtab. If previous tax details existed, this process overwrites the existing values. The tax details are saved to the database if the user saves the transaction.

Transaction Types Supported By the SuiteTax Plug-in

The following table lists the transaction types and the associated transaction type ID supported by the SuiteTax Plug-in:

Transaction Type	Transaction Type ID
Cash Refund	cashrfnd
Cash Sale	cashsale
Credit Card Charge	cardchrg
Credit Card Refund	cardrfnd

Transaction Type	Transaction Type ID
Credit Memo	custcred
Estimate	estimate
Expense Report **Note:** Upon saving an expense report with taxes, NetSuite sends the tax amounts to the tax engine for reporting purposes. The tax engine is not expected to calculate taxes or send back results for expense reports.	exprept
Invoice	custinvc
Journal Entry (book generic only) **Note:** Upon saving a journal transaction with taxes, NetSuite sends the tax amounts to the tax engine for reporting purposes. The tax engine is not expected to calculate taxes or send back results for journal transactions.	journal
Opportunity	opprtnty
Purchase Order	purchord
Return Authorization	rtnauth
Sales Order	salesord
Vendor Bill	vendbill
Vendor Credit (Bill Credit)	vendcred
Vendor Return	vendauth

SuiteTax Plug-in Features in NetSuite

To use the SuiteTax plug-in in a NetSuite account, an administrator must enable the SuiteTax feature, install a bundle that contains an implementation of the SuiteTax plug-in, and enable the implementation. A bundle can contain one or more SuiteTax plug-in implementations, where each implementation represents a different third-party tax engine.

For more information, see Developing a SuiteTax Plug-in Implementation and Administering a SuiteTax Plug-in Implementation.

Enabling the SuiteTax feature, and supporting the use of SuiteTax plug-in implementations, means that you are using a tax calculation engine that is different from the legacy NetSuite tax engine. The SuiteTax tax calculation engine enables greater flexibility to support specific country needs and legislation changes regarding tax calculation and reporting. With SuiteTax enabled, tax setup procedures vary from the tax setup procedures described in the legacy tax help topics. For details about setting up taxes with SuiteTax, see the help topic General SuiteTax Topics.

After an administrator installs a SuiteTax bundle and enables a SuiteTax plug-in implementation that represents a tax engine, you can complete the following tasks for taxable transactions:

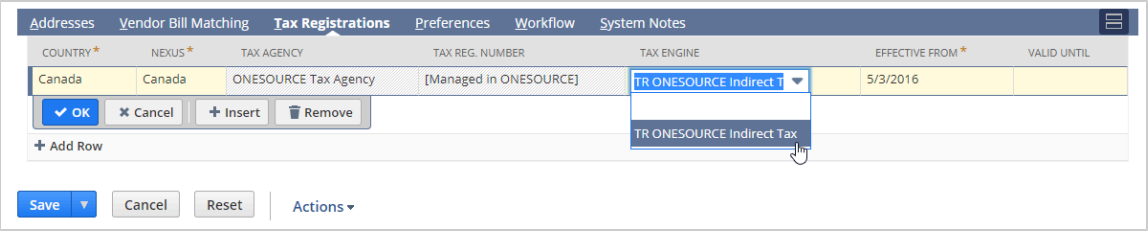
- Assign a Tax Engine to a Nexus
- Calculate Taxes on a Transaction

Assign a Tax Engine to a Nexus

NetSuite supports the use of more than one tax service provider. If the SuiteTax feature and the tax engine you want to use is enabled in your account, you can assign different tax engines to the tax registrations on each subsidiary record.

To assign a tax engine to a nexus, edit the list of tax registrations for a subsidiary and specify the tax engine to use with the appropriate nexus for the subsidiary.

The following screenshot shows the **Tax Registrations** subtab on the Subsidiary page:



In the **Tax Engine** column, select the tax engine to use for the nexus. The dropdown list shows tax engine implementations that have been added and enabled in your account.

Calculate Taxes on a Transaction

After you assign a tax engine to a nexus, you can calculate taxes for any taxable transaction associated with that nexus. You can either calculate taxes on-demand or calculate taxes automatically. You can use the **Preview Tax** button on any taxable transaction to calculate appropriate taxes whenever you make changes to the taxable items on the transaction. In addition, NetSuite calculates taxes automatically when you save the transaction.

When you click **Preview Tax**, the taxes displayed on the transaction are temporarily displayed in NetSuite. The amounts are saved to the database when you save the transaction.

You must recalculate taxes when you make the following changes to taxable items on a transaction:

- **Add items.** You add taxable items to the transaction.
- **Change existing items.** You modify fields on the transaction and these changes require a recalculation of tax.

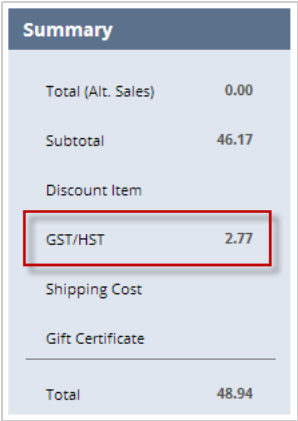
Note: As an alternative to the **Preview Tax** button, tax calculation for transactions can also be triggered through SuiteScript. See the help topic Triggering Tax Calculation through SuiteScript.

Calculated taxes appear on the Summary box and on the Tax Details subtab of a transaction form when you calculate taxes. The tax amount and gross amount also appear on the Items, Expenses, or Billables subtab of a transaction. For credit card transactions, the tax also appear in the header section. The taxes and tax types that appear on a transaction depend on the method by which the tax engine calculates taxes.

Transaction Summary Box

The total tax summary by tax type appears on the Summary box after you click **Preview Tax** on a transaction or you save the transaction. The tax summary is for all taxable items on a transaction for that tax type. All transactions that use a tax engine defined by a SuiteTax plug-in implementation must show summary totals in the Summary box.

The following screenshot show the Summary box of a transaction with the GST/HST tax totals:



Tax Details Subtab

The total tax summary by tax type appears on the Summary box after you click **Preview Tax** on a transaction or you save the transaction. The taxes by item type appear on the **Tax Details** subtab if the tax engine defined by a SuiteTax plug-in implementation returns taxes by line item. The tax details are grouped by tax type and tax code.

If an item has multiple tax types or tax codes associated with it, each tax type or tax code appears on a separate line on the **Tax Details** subtab. The tax details include the tax type, code, basis, rate, and amount. The tax type, tax code, and tax rate that are returned by the tax engine must be maintained separately in the plug-in implementation as well as in NetSuite. The Details column shows the information returned by the tax engine used to calculate the tax.

The Tax Details subtab also includes a **Tax Details Override** check box. Users can check this box to allow changes to system-generated tax detail lines. The tax engine cannot change output after the **Tax Details Override** box has been checked. You can use the TaxCalculationInput.isTaxOutputOverridden() method to return a Boolean value indicating whether this box is checked.

For more information about the Tax Details subtab, see the help topic Tax Details on Transactions in SuiteTax.

Calculate Taxes Example

You add a taxable item to a transaction and click **Preview Tax**. Depending on the tax engine functionality defined by the SuiteTax plug-in implementation, the tax details appear on the Summary box and the **Tax Details** subtab. You save the transaction. The tax details are saved to the database.

You then change the amount of the item on the **Items** subtab. NetSuite removes the tax details in the Summary box and the line-item details on the **Tax Details** subtab. Click **Preview Tax** to view the updated tax details. When you save the transaction, NetSuite recalculates the tax and saves the updated tax details to the database.

Note: Preview tax calculation is not available if the **Tax Details Override** box is checked.

Developing a SuiteTax Plug-in Implementation

NetSuite provides an interface to the SuiteTax plug-in. You can create an implementation of the interface that represents a tax engine in NetSuite. Use the tax engine to calculate taxes for taxable transactions associated with a nexus.

You can create one implementation of the plug-in interface that defines tax engines for all nexuses in a NetSuite account, or you can create multiple implementations that each define a tax engine that you can use with a single nexus. The number of implementations depends on your requirements.

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 tax engines and their associated objects in a NetSuite account and enable the tax engine. For more information about administration tasks for a SuiteTax Plug-in bundle, see Administering a SuiteTax Plug-in Implementation.

The following table describes the basic steps in developing a single plug-in implementation of the SuiteTax plug-in. Use these steps to create one or more SuiteTax plug-in implementations that define tax engines in a NetSuite account.

Step	Description
Enable features.	Enable the features required for developing the plug-in implementation, including the SuiteTax and Server SuiteScript features. For information, see Enable Features for a SuiteTax Plug-in Implementation.
Create required objects.	Create the objects on which the plug-in implementation depends. These objects include nexuses, tax types, and tax codes. You need the internal NetSuite ID of these objects to complete development. For information, see Create or Set Up Required Objects.
Create script file.	Create the script file that contains the SuiteTax plug-in implementation. For information, see Create the Plug-in Implementation Script File.
Add the plug-in implementation.	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. For information, see Add the Plug-in Implementation.
Test the plug-in implementation.	Test the functionality of the tax engine to make sure that it properly calculates taxes for the taxable transactions. You will need to activate the plug-in implementation and create a tax nexus. For information, see Test the Plug-in Implementation.
Bundle the plug-in implementation.	Bundle the plug-in implementation for distribution to other NetSuite accounts. NetSuite administrators use this bundle to install the plug-in implementation and plug-in script and utility files. For information, see Bundle the Plug-in Implementation.

Create or Set Up Required Objects

Before you begin to develop a SuiteTax plug-in implementation, you must set up the tax features for the development account. Any SuiteTax plug-in implementation requires the following objects in the account. In addition, the tax type and tax code details that the plug-in implementation returns as tax details must exist in NetSuite.

Note: The information provided in this section is intended for use as a guideline. For more detailed information on setting up taxes for a company that uses SuiteTax, see the help topic General SuiteTax Topics.

The following table describes the objects that must exist in the account when you begin development of the plug-in implementation:

Object Type	Description
Nexus	A nexus is a tax jurisdiction, usually defined at the country level. Each parent company or subsidiary must be associated with at least one nexus. The SuiteTax plug-in implementation is referenced by a nexus as a tax engine. You must have a nexus to which you assign the tax engine. For information about nexuses, see the following help topics: ■ Understanding Nexuses in SuiteTax ■ Setting Up Nexuses in SuiteTax ■ Assigning Tax Registrations to a Subsidiary in SuiteTax
Tax type	A tax type determines where the tax paid or collected is tracked on the balance sheet. A tax type is associated with a nexus. The SuiteTax plug-in implementation must return the tax type internal ID as part of the tax details, either on a transaction or line-item level. The internal ID returned by the implementation must exist in NetSuite. For information about tax types, see the help topic Understanding Tax Types and Tax Codes in SuiteTax.
Tax code	Tax codes contain information about tax rates and the types of transactions that the tax codes should be applied to. A tax code is associated with a tax type. A tax type can have multiple tax codes. The SuiteTax plug-in implementation must return the internal ID of the tax code as part of the tax details, either on a transaction or line-item level. The internal ID returned by the implementation must exist in NetSuite. For information about tax codes, see the help topic Understanding Tax Types and Tax Codes in SuiteTax.
Standard or custom field	You can pass additional field values to the SuiteTax plug-in implementation to use with your tax calculation algorithm. You can use standard NetSuite fields or custom fields, either on the transaction form or the item record form. If you want to pass custom field values to the implementation, you must create the custom fields before you include them in the plug-in implementation. For more information about creating custom fields, see the help topic Custom Fields.

Create the Plug-in Implementation Script File

You must implement each SuiteTax plug-in interface function in a JavaScript file (with a .js extension) to define the behavior of the plug-in implementation. You can use the SuiteCloud IDE or another JavaScript IDE or text editor to create the plug-in script file.

Interface Functions

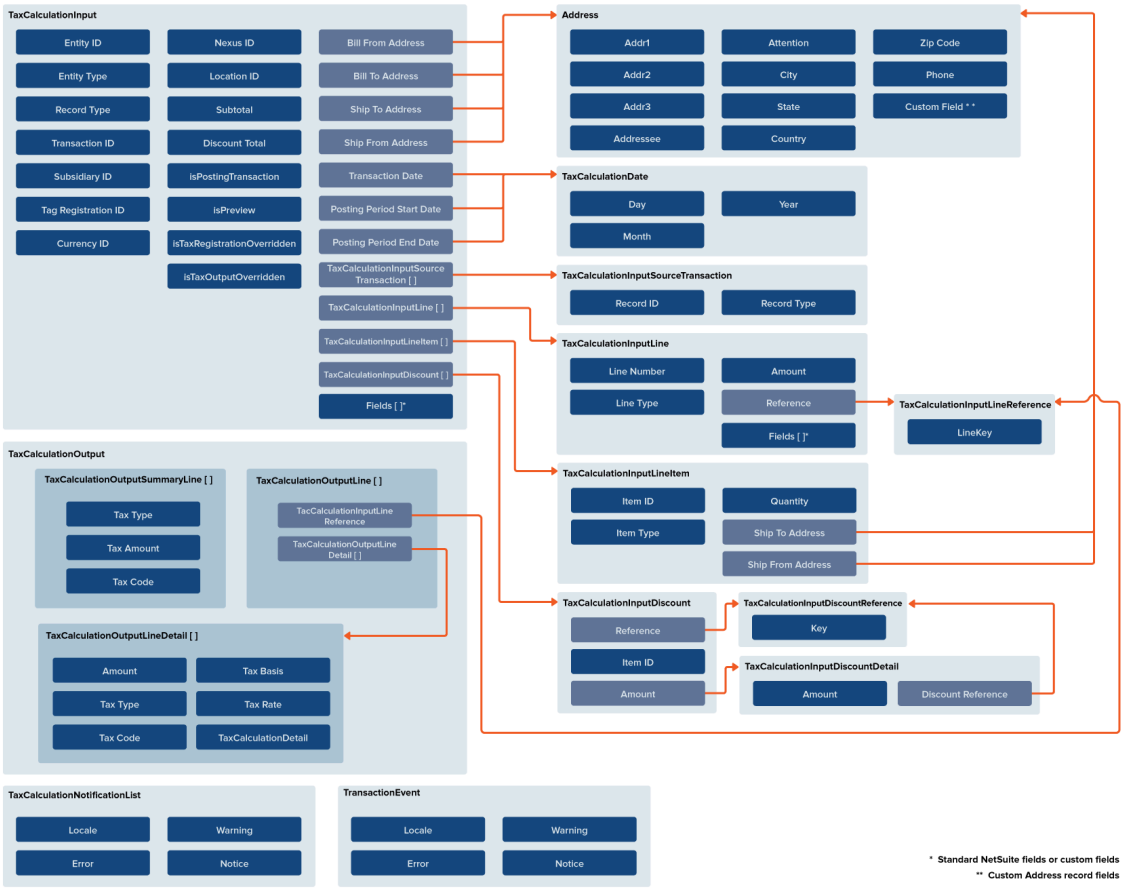
The following table describes the functions that must be implemented in the plug-in script file if using SuiteScript 2.0:

Function	Description
calculateTax(context)	Defines the tax calculation functionality for a SuiteTax plug-in implementation. Use the methods available to the TaxCalculationInput and TaxCalculationInputLine objects to get information about a taxable transaction.

Function	Description
	You can access transaction-level properties such as location, shipping and handling costs, and shipping and billing addresses and line item-level properties such as item type, quantity, and value. After you calculate the tax amounts on a transaction or line-item level, use the methods available to the TaxCalculationOutput object to pass the tax details, including tax amounts and tax types, back to NetSuite. This function is called by NetSuite when you click **Preview Tax** on a taxable transaction or when you save a taxable transaction. You can use the SuiteScript API N/http Module method to request tax details from a third-party external system. Use TaxCalculationNotificationList to display error, warning, and notice notifications to the NetSuite user during plug-in implementation execution.
defineAdditionalFields(context)	Defines the additional fields on a transaction or sublist that NetSuite passes to the plug-in implementation. The fields can be standard NetSuite fields or custom fields added to the record form for the transaction type or line values. Use this function to pass additional field values to the plug-in that are not supported by the methods available to the TaxCalculationInput or the TaxCalculationInputLine interface input objects. Use the field values to define additional plug-in functionality for tax calculation on the transaction. **Note:** If you do not want to pass additional field values to the plug-in implementation, leave this function with an empty declaration.
onTransactionEvent(context)	Defines the functionality for a SuiteTax plug-in implementation when specific events occur on the transaction record. You can define plug-in functionality for voids and deletions of a transaction. Use the methods available to the TaxCalculationInput object to get information about the transaction and use TransactionEvent to get the type of event that occurred. **Note:** This function is not called when a user clicks **Preview Taxes** on a supported taxable transaction.

SuiteTax Plug-in Object Model

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



Basic Process for Implementing Interface Functions

Use the following basic processes to define the plug-in implementation functionality for calculating tax on a transaction and line-item level. Both the transaction and line-item levels must be set in the implementation.

To calculate taxes on a transaction level:

1. Use the methods available to the TaxCalculationInput object to access transaction details.
2. Calculate the taxes on the transaction in the plug-in script using the transaction values.
3. Use setTaxSummaryLine(options) to add the summary to the TaxCalculationOutput interface output object.

   The summary total appears in the Summary box for the transaction in NetSuite. See Transaction Summary Box.

> **Important:** You must use setTaxSummaryLine(options) to set the summary for each tax type on which you calculate values.

To calculate taxes on a line item-level:

1. Use the TaxCalculationInput object and the lines property to get the array of all line items in the transaction.

> **Note:** The order of lines returned is not guaranteed.

2. Use the properties available to the TaxCalculationInputLine objects to get details about each line item in the transaction.
3. Create a TaxCalculationOutputLine object with createLine(options) to store the tax details for the line item.

   This object can hold information for different tax types and tax codes by calling addTaxDetail(options).
4. Use addLine(options) to add the `TaxCalculationOutputLine` object to the TaxCalculationOutput interface output object. The plug-in implementation passes these values back to NetSuite and displays them on the Tax Details subtab for the transaction. See Tax Details Subtab.
5. In the plug-in implementation script file, aggregate the totals for each tax type and tax code for each line item.
6. Use setTaxSummaryLine(options) to add the summary to the TaxCalculationOutput interface output object.

   The summary total appears in the Summary box for the transaction in NetSuite. See Transaction Summary Box.

> **Important:** You must use setTaxSummaryLine(options) to set the summary for each tax type on which you calculate values, even if you calculate taxes on a line-item level.

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 each of the interface functions.
- 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 SuiteTax plug-in in NetSuite. See Add the Plug-in Implementation.
- You can use SuiteScript API functions in a plug-in implementation. Governance limits apply to these functions. The plug–in script file allows up to 1000 usage units when used with SuiteScript.

Add the Plug-in Implementation

When you have finished creating the plug-in implementation script file, create a new SuiteTax plug-in implementation in the development account. When you create the plug-in implementation, you upload the script file and other utility files as required. You can later bundle this implementation to distribute it to other NetSuite accounts.

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

1. In NetSuite, go to Documents > Files > File Cabinet.
2. On the Folder Contents page, click the name of the folder where you want to add the plug-in script file. Optionally, click **New Folder** to create a new folder.
3. Click **Add File**. Select the plug-in script file and click **Open**.

4. Optionally, repeat step 3 to upload any utility script files required by the plug-in implementation script file.
5. Go to Customization > Plug-ins > Plug-in Implementations > New.
6. In the **Script File** field, select the script file that contains the implementation of the plug-in.
7. Click **Create Plug-in Implementation**.
8. On the Select Plug-in Type page, select the **Tax Engine** plug-in type.
9. On the Plug-in Implementation page, enter the following information:

Option	Description
Name	User-friendly name for the implementation. The plug-in implementation appears as a tax engine in the following locations: ■ Manage Plug-ins page. Page used by administrators to enable/disable the plug-in implementation in their account. ■ Subsidiary page. The plug-in implementation appears as a tax engine. ■ Bundle Builder. Select this name in the Bundle Builder to distribute the plug-in implementation to other accounts.
ID	Internal ID for the implementation for use in scripting. If you do not provide an ID, NetSuite will provide one for you after you click **Save**.
Status	Current status for the implementation. Choose **Testing** to have the implementation accessible to the owner of the implementation. Choose **Released** to have the implementation accessible in a production environment.
Log Level	Logging level you want for the script. Select **Debug**, **Audit**, **Error**, or **Emergency**.
Execute As Role	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 **Tax Engine** role is added to your account when you enable the SuiteTax feature. This role is set up with all the required permissions to use SuiteTax, and can be used as template for creating customized tax roles. For more information, see the help topic Roles and Permissions in SuiteTax. You should set up your SuiteTax plug-in implementations to run under the **Tax Engine** role. **Note:** When using a customized version of the Tax Engine role, you should consult your tax engine provider regarding your customizations.
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 value is the name of the logged in user.

10. On the **Scripts** subtab, in the **Implementation** field, the script file that you selected in step 6 is automatically selected. You can change the script file that contains the implementation of the plug-in, if required.
11. On the **Scripts** subtab, in the Library Script File list, select any utility script files (uploaded in step 4) that are required by the plug-in script file
12. 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** field, separate email addresses with a semi-colon.
13. Click **Save**. You can access the list of implementations by going to Customization > Plug-ins > Plug-in Implementations.

Test the Plug-in Implementation

To test a SuiteTax plug-in implementation, perform the following tasks:

- Set up the plug-in implementation to run under the Tax Engine role. When you create the plug-in implementation, you can set the role that the script runs as in the **Execute As Role** field. For more information, see Add the Plug-in Implementation.
- Enable the implementation. Use the Manage Plug-ins page to enable the plug-in implementation in the development account for testing. See Enable the SuiteTax Plug-in Implementation.
- Configure the tax engine for nexuses. Configure the plug-in implementation as a tax engine for the appropriate nexuses through the Subsidiary page. For more information, see Configure the Tax Engine for Nexuses.
- Create transactions that use the tax nexus you created to test the plug-in implementation functionality.

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

Bundle the Plug-in Implementation

After developing the SuiteTax plug-in implementation, you must distribute the implementation to a production account. SuiteBundler allows NetSuite users to package together groups of objects for distribution to other accounts. These packages are called bundles, or SuiteApps. To distribute the required files and objects, 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:

Object	Location On Select Objects Page
Plug-in script file Utility files	File Cabinet > Files
Custom roles	Roles > Custom Roles
Plug-in implementation	Plug-ins > Tax Engine

For more information about creating a bundle, see the help topic Creating a Bundle with the Bundle Builder in the SuiteBundler documentation.

Important: The specific SuiteTax implementation also depends on account specific objects that cannot be added to a bundle, such as tax nexuses, tax types, and tax codes. The process to set up these objects in the production account may vary depending on the implementation. See Install a SuiteTax Plug-in Bundle.

Administering a SuiteTax Plug-in Implementation

After a developer creates an implementation of the Suitetax plug-in and bundles it as a SuiteApp, you can install and set up the plug-in implementation bundle.

To install and set up an SuiteTax plug-in implementation, complete the following steps:

Enable Features for a SuiteTax Plug-in Implementation

Before you install a SuiteTax plug-in implementation, you must enable the Server SuiteScript and SuiteTax features.

To enable features for the SuiteTax plug-in:

1. Choose Setup > Company > Enable Features.
2. On the **SuiteCloud** subtab, make sure that Server SuiteScript is checked. If necessary, check the box and agree to the Terms of Service.
3. On the **Tax** subtab, check the box for **SuiteTax**.
4. Click Save.

Install a SuiteTax Plug-in Bundle

A developer can create an implementation of the SuiteTax 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 the SuiteTax plug-in bundle:

1. Go to Customization > SuiteBundler > Search & Install Bundles.
2. On the Install Bundle page, choose Production Account in the Location dropdown.
3. Search for the plug-in bundle.
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.

To check on the progress of the installation, go to the list of installed bundles at Customization > SuiteBundler > Search & Install Bundles > List. If installation is not complete, the Status column displays the percentage of installation progress. Click Refresh to update the status. When installation is complete, the Status column displays a green check.

> **Note:** If no Install button is available, this SuiteApp may not have been shared with your account. To get access to the SuiteApp, contact NetSuite Customer Support.

After You Install the Bundle

SuiteTax plug-in implementations will depend on other objects that cannot be included in a bundle, such as tax nexuses, tax types, and tax codes. You can set up a tax nexus after you install a bundle, but the other objects may need to be created in the production account after the bundle is installed.

In general, you can use the following methods to create the dependent objects:

- Run a bundle installation script. A bundle can include a specialized server SuiteScript that is automatically run when the bundle is installed, updated, or uninstalled. A bundle installation script can contain triggers to be executed before install, after install, before update, after update, and/or before uninstall. See the help topic Bundle Installation Scripts.
- Import the objects. Use a CSV file and the Import Assistant to import objects into a NetSuite account. See the help topic CSV Imports Overview.
- Create the objects manually. For more information about working with taxes in NetSuite, see the help topic Tax Accounting Overview.

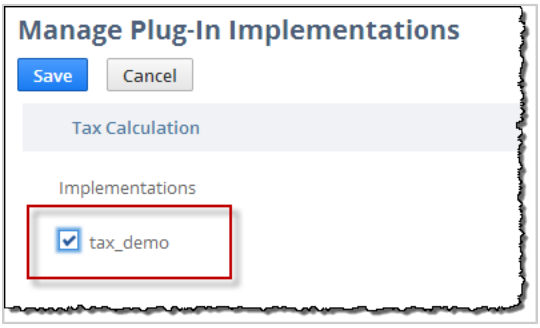
Contact the bundle author for more information.

Enable the SuiteTax Plug-in Implementation

After you install a bundle with a SuiteTax plug-in implementation, you must enable the plug-in implementation for users to select it as a tax engine for nexuses.

To activate the plug-in implementation:

1. In NetSuite, go to Customization > Plug-ins > Manage Plug-ins.
2. Under Tax Calculation, check the box next to the name of the plug-in implementation.



3. Click **Save**.

Configure the Tax Engine for Nexuses

To assign the plug-in implementation as a third-party tax engine to a nexus:

1. If you are using a OneWorld account, go to Setup > Company > Subsidiaries and click the **Edit** link of the subsidiary.

   -or-

   If you are using a non-OneWorld account, go to Setup > Company > Company Information.
2. Click the **Tax Registrations** subtab.
3. In the **Tax Engine** column, select the tax engine to use for this nexus. The dropdown list shows tax engine implementations that have been added and activated in your account.
4. Click **Save**.

SuiteTax Plug-In Interface Definition

Interface Functions

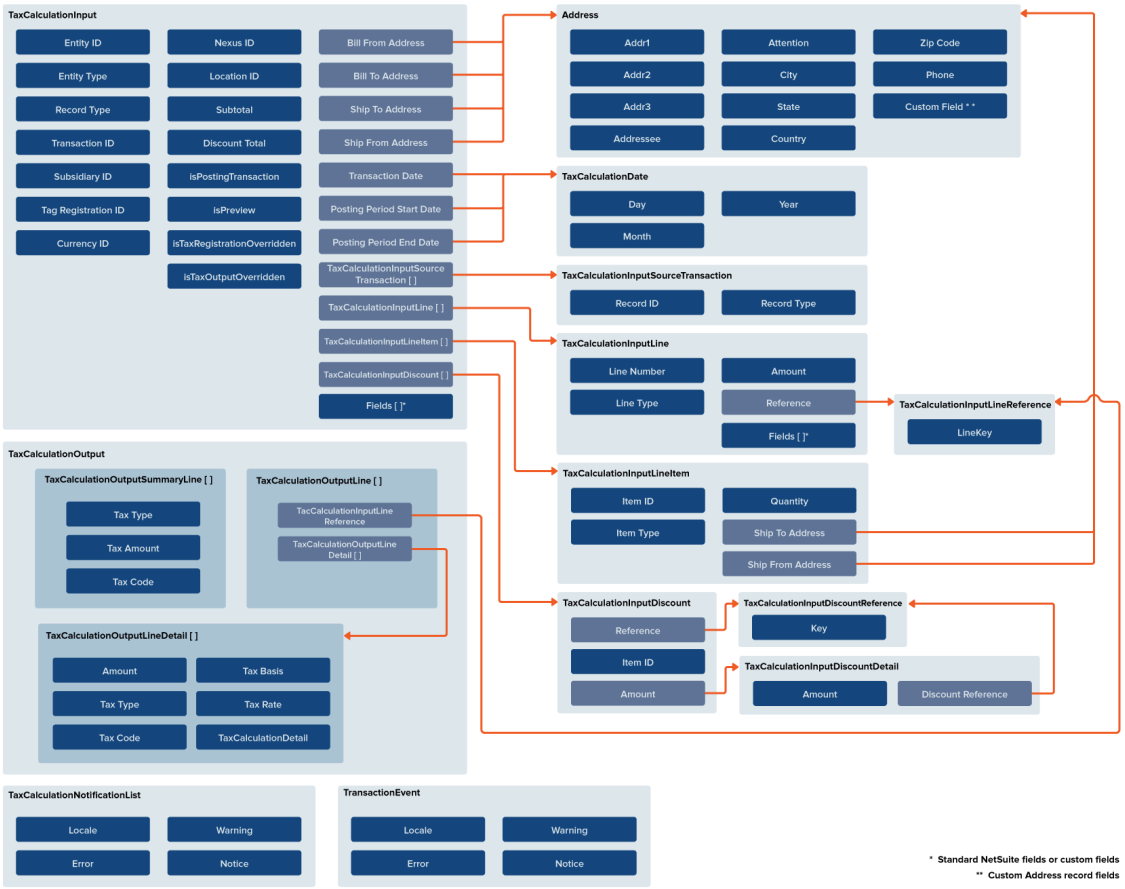
The SuiteTax plug-in interface includes the following functions:

Function	Description
calculateTax(context)	Defines the tax calculation functionality for a SuiteTax plug-in implementation. Use the methods available to the TaxCalculationInput and TaxCalculationInputLine objects to get information about a taxable transaction. You can access transaction-level properties such as location, shipping and handling costs, and shipping and billing addresses and line item-level properties such as item type, quantity, and value. After you calculate the tax amounts on a transaction or line-item level, use the methods available to the TaxCalculationOutput object to pass the tax details, including tax amounts and tax types, back to NetSuite. This function is called by NetSuite when you click **Preview Tax** on a taxable transaction or when you save a taxable transaction. You can use the SuiteScript API N/http Module method to request tax details from a third-party external system. Use TaxCalculationNotificationList to display error, warning, and notice notifications to the NetSuite user during plug-in implementation execution.
defineAdditionalFields(context)	Defines the additional fields on a transaction or sublist that NetSuite passes to the plug-in implementation. The fields can be standard NetSuite fields or custom fields added to the record form for the transaction type or line values. Use this function to pass additional field values to the plug-in that are not supported by the methods available to the TaxCalculationInput or the TaxCalculationInputLine interface input objects. Use the field values to define additional plug-in functionality for tax calculation on the transaction. **Note:** If you do not want to pass additional field values to the plug-in implementation, leave this function with an empty declaration.
onTransactionEvent(context)	Defines the functionality for a SuiteTax plug-in implementation when specific events occur on the transaction record. You can define plug-in functionality for voids and deletions of a transaction. Use the methods available to the TaxCalculationInput object to get information about the transaction and use TransactionEvent to get the type of event that occurred. **Note:** This function is not called when a user clicks **Preview Taxes** on a supported taxable transaction.

Important: You cannot change these function signatures in a SuiteTax plug-in implementation.

SuiteTax Plug-in Object Model

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



calculateTax(context)

Function Declaration

```
void calculateTax(context)
```

Type

Interface function

Description

Defines the tax calculation functionality for a SuiteTax plug-in implementation. Use the methods available to the TaxCalculationInput and TaxCalculationInputLine objects to get information about a taxable transaction.

Use JavaScript and SuiteScript to define tax calculation on a transaction. You can access transaction-level properties such as shipping and handling costs and shipping and billing addresses and line item-level properties such as item type, quantity, value, and when the Multiple Shipping Routes feature is enabled, shipping addresses. Transaction-level and line item-level discounts also are available.

Within this function, you can use SuiteScript and SOAP web services functionality to connect to external systems, pass input information to the external system, and use the response in tax calculations.

After you calculate the tax amounts on a transaction or line item level, use the methods available to the TaxCalculationOutput objects to pass the tax details, including tax amounts, tax types, and tax codes, back to NetSuite.

The tax amounts appear in the summary panel for the transaction and on the Tax Details subtab for each individual line item.

This function is called by NetSuite when you click **Preview Tax** on a taxable transaction or when you save a taxable transaction.

Returns void

Parameters

- TaxCalculationInput
- TaxCalculationOutput
- TaxCalculationNotificationList

TaxCalculationInput

Type Interface input object

Description Contains input values to use in tax calculation.

Method

- getAdditionalFieldValue(options)

Properties

- billFromAddress
- billToAddress
- currency
- discounts
- discountTotal
- entity
- entityType
- lines
- location
- nexus
- postingPeriodEndDate
- postingPeriodStartDate
- recordType
- shipFromAddress
- shipToAddress
- subsidiary
- subtotal
- taxRegistration
- transactionDate
- transactionId
- transformationSourceTransactions
- postingTransaction
- preview
- taxOutputOverridden
- taxRegistrationOverridden

Child Objects

- Address
- TaxCalculationInputLine
- TaxCalculationInputDiscount

getAdditionalFieldValue(options)

Method Declaration	`string getAdditionalFieldValue(options)`
Type	Transaction-level object method
Description	Returns the value for the field on the transaction passed as an input parameter. The field can be a standard NetSuite field or a custom field added to the transaction form. Use this method if you want to request that NetSuite sends additional fields to the plug-in implementation that are not available in the methods for TaxCalculationInput. To use this method, complete the following steps: ■ If you want to use a custom field, add the custom field to the transaction form. ■ Specify the names of the fields that you want NetSuite to pass to the plug-in using defineAdditionalFields(context).
Returns	string
Input Parameters	`options.fieldId {string}`— field name of a standard or custom field on the transaction form.
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {

   function calculateTax(context){
          ...
          // Get value of 'memo' custom field

          var memoField = context.input.getAdditionalFieldValue('memo');


          // add memo field to description of tax calculation
          description = description + ': ' + memoField;
          ...


    }
    function defineAdditionalFields(context){
        context.addField({
          fieldId: 'memo'
        });
    }

    return {
        calculateTax: calculateTax,
        defineAdditionalFields: defineAdditionalFields
    }
}
```

billFromAddress

Property Declaration	`Address billFromAddress`
Type	Transaction-level object property
Description	Returns an Address object that represents the bill from address of the current transaction being processed by the plug-in implementation. See Values on an Address Object for more information about from where the possible values for this address are sourced, depending on the transaction type. This property contains null if there is no billing address specified. Use this object, for example, to calculate tax amounts based on the billing address for entity fulfilling the transaction, if you charge taxes based on who sold an item. You can calculate tax amounts based on the values of the billing address, including city, state, zip code, or country. **Note:** You can also use location to load the location record for the Customer or Vendor.
Returns	Address | NULL
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {

  function calculateTax(context){
        ...

        var address = context.input.billFromAddress;


        if (address.country == 'US') {
            // Calculate domestic tax amount
            ...
        } else {
            // Calculate international tax amount
            ...
        }
            ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

billToAddress

Property Declaration	`Address billToAddress`
Type	Transaction-level object property
Description	Returns an Address object that represents the bill to address of the current transaction being processed by the plug-in implementation. See Values on an Address Object for more information about from where the possible values for this address are sourced, depending on the transaction type. This property contains null if there is no billing address specified. Use this object, for example, to calculate tax amounts based on the billing address of the entity initiating the transaction. You can calculate tax amounts based on the values of the billing address, including city, state, zip code, or country.
Returns	Address | NULL
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {

  function calculateTax(context){
        ...

        var address = context.input.billToAddress;


        if (address.country == 'US') {
            // Calculate domestic tax amount
            ...
        } else {
            // Calculate international tax amount
            ...
        }
            ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

currency

Property Declaration	`Number currency`

Type	Transaction-level object property
Description	Returns the internal NetSuite ID for the `currency` object with the transaction. For example, for a sales order, this property returns the ID of the currency for the transaction. You can use this value to load the currency record with record.load(options) and the type value of 'currency'. Use this object, for example, to calculate tax amounts based on the currency for the transaction.
Returns	number
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {

        function calculateTax(context){
        ...
        var currencyID = context.input.currency;
        var currencyObj = record.load({
          type: record.Type.CURRENCY,
          id: currencyID
        });
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

discounts

Property Declaration	`TaxCalculationInputDiscount[] discounts`
Type	Transaction-level object property
Description	Returns an array of TaxCalculationInputDiscount objects. You can use the discount items to calculate taxes based on the discounted amount.
Returns	TaxCalculationInputDiscount[]
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
```

```
define([], function() {

    function calculateTax(context){
        ...
        var input = context.input;
        var currencySymbol = getCurrencySymbol
          (input.currency));

        var discounts = input.discounts;
        if (discounts)
        {
              Context.notifications.addNotice(
                'This transaction contains '
                  + discounts.length
                  + ' different discounts.'
              );
        } else {
            Context.notifications.addNotice(
              'There are no discounts on this transaction.'
            );
        }
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

discountTotal

Property Declaration	`String discountTotal`
Type	Transaction-level object property
Description	Returns the total of all discount items for a transaction as a string. This total includes the amounts for line-level discounts. If there are no discount items for a transaction, this property returns 0.
Returns	string
Input Parameters	None
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {

    function calculateTax(context){
        ...
```

```
      var discountTotal = context.input.discountTotal;
      context.notifications.addNotice(
        'The total discount amount for this transaction is
          ' + discountTotal
          + '.'
      );
        ...
    }

    return {
        calculateTax: calculateTax
    }
});
```

entity

Property Declaration	`Number entity`
Type	Transaction-level object property
Description	Returns the entity internal ID for the current transaction. The type of the entity depends on the transaction type. For example, for a sales order transaction type, this property returns the entity ID associated with the customer specified in the sales order Customer property. Use this property to define plug-in functionality based on the transaction entity. For example, you can use this ID with the SuiteScript API record.load(options) to access the entity record for the transaction. For more information, see the following topics: ■ record.load(options) ■ SuiteScript Supported Records
Returns	number
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
        var input = context.input;
        var entityType = input.entityType;
        var entityID = input.entity;

        if (entityType == 'customer') {
            var customerRecord = record.load({
              type: entityType,
              id: entityID
            });
      }
```

```
    return {
        calculateTax: calculateTax
    }
}
```

entityType

Property Declaration	`String entityType`
Type	Object property
Description	Returns the entity type for the current transaction. The type of the entity depends on the transaction type. For example, for a sales order transaction type, this property returns the *customer* entity type. Use this property to define plug-in functionality based on the transaction entity. For example, you can use this entity type with the SuiteScript API record.load(options) to access the entity record for the transaction.
Returns	string
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
        var input = context.input;
        var entityType = input.entityType;
        var entityID = input.entity;
        if (entityType == 'customer') {
            var customerRecord = record.load({
              type: entityType,
              id: entityID
            });
      }

    return {
        calculateTax: calculateTax
    }
}
```

lines

Property Declaration	`TaxCalculationInputLine[] lines`
Type	Transaction-level object property
Description	Returns an array of TaxCalculationInputLine objects. Each `TaxCalculationInputLine` object represents a single line item on the transaction being processed by the plug-in

implementation. Use this property to get all the line items and iterate through them to calculate tax for each line item.

Note: The order of lines returned by this property is not guaranteed.

After you get the array of `TaxCalculationInputLine` objects, for each line item, you can:

- Use createLine(options) to create the TaxCalculationOutputLine object.
- Calculate the tax and add the tax detail for each taxation type for the line item. Use addTaxDetail(options) to add the tax detail for the line item.
- Add the tax details for the line item to the TaxCalculationOutput object with addLine(options).

Returns TaxCalculationInputLine[]

Parent object TaxCalculationInput

Example

```
/**
   * @NApiVersion 2.0
   * @NScriptType taxCalculationPlugin
   */
define([], function() {
    function calculateTax(context){
        var inputLines = context.input.lines;

        for (
          var inputLineIndex = 0;
          inputLineIndex < inputLines.length;
          inputLineIndex++
        )
        {
            // Create the tax line
            var line = inputLines[inputLineIndex];
            var lineReference = line.reference;
            var taxesForLine = context.output.createLine({
              lineReference: lineReference
            });
            // get the line amount
            amount = line.amount;
            // Calculate the tax
            ...
            // add the tax detail to the taxes for the line item
            taxesForLine.addTaxDetail({
                taxCode: taxCode,
                taxationType: taxType,
                taxRate: taxRate,
                taxAmount: taxAmount,
                taxBasis: taxBasis,
                taxCalculationDetail: description
            });

            // add the TaxCalculationOutputLine tax detail to the output object
            output.addLine(taxesForLine);
            ...
```

```
    }

    return {
        calculateTax: calculateTax
    }
}
```

location

Property Declaration	`Number location`
Type	Transaction-level object property
Description	Returns the internal NetSuite ID for the Location property associated with the Customer property of a transaction. You can use this value to load the location record with record.load(options) and a type value of 'location'. Depending on your requirements, you can also use shipFromAddress or shipToAddress. Use `location` to load the record with SuiteScript, use the other properties if you want to use the string value of the address.
Returns	number
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
        function calculateTax(context){
       var input = context.input;
       var location = input.location;
       var locationObj = record.load({
         type: record.Type.LOCATION,
         id: location
       });
    }

    return {
        calculateTax: calculateTax
    }
}
```

nexus

Property Declaration	`Number nexus`
Type	Transaction-level object property
Description	Returns the internal NetSuite ID for the nexus on a transaction. When you calculate tax on a transaction, NetSuite uses the transaction properties to look up the associated nexus for the

transaction. You can use this value to load the nexus record with record.load(options) and a type value of 'nexus'.

You can assign a single plug-in implementation, as a tax engine, to multiple nexuses. You can then use this property to determine which nexus is associated with the transaction and design plug-in implementation functionality based on the nexus.

Returns number

Parent object TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
      var input = context.input;
      var currNexus = input.nexus
      var nexusObj = record.load({
        type: record.Type.NEXUS,
        id: currNexus
      });
    }

    return {
        calculateTax: calculateTax
    }
}
```

postingPeriodEndDate

Property Declaration `TaxCalculationDate postingPeriodEndDate`

Type Transaction-level object property

Description Returns a TaxCalculationDate object that represents the end date for the accounting period to which the transaction posted.

Use this object to define plug-in implementation functionality based on the accounting period end date. You can define plug-in implementation functionality based on the values of the day, month, or year.

Note: This method will be deprecated after Accounting Period records are made available to SuiteScript.

Returns TaxCalculationDate

Parent object TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
```

```
  */
define([], function() {
    function calculateTax(context){
      var input = context.input;
      ...
      var periodEnd = input.postingPeriodEndDate;
      var periodStart = input.postingPeriodStartDate;

      var totalTax = calculateTaxByPeriod(
        periodStart,
        periodEnd
      );
      ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

postingPeriodStartDate

Property Declaration	`TaxCalculationDate postingPeriodStartDate`
Type	Transaction-level object property
Description	Returns a TaxCalculationDate object that represents the start date for the accounting period to which the transaction posted. Use this object to define plug-in implementation functionality based on the accounting period start date. You can define plug-in implementation functionality based on the values of the day, month, or year. **Note:** This method will be deprecated after Accounting Period records are made available to SuiteScript.
Returns	TaxCalculationDate
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context){
      var input = context.input;
      ...
      var periodEnd = input.postingPeriodEndDate;
      var periodStart = input.postingPeriodStartDate;

      var totalTax = calculateTaxByPeriod(
        periodStart,
```

```
        periodEnd
      );
      ...

    }

    return {
        calculateTax: calculateTax
    }
}
```

recordType

Parameter Declaration	`String recordType`
Type	Transaction-level object property
Description	Returns the record type for the current transaction. Use this value to define plug-in implementation functionality based on the type of record passed to the plug-in implementation. For example, you can calculate taxes differently for sales order transactions or purchase order transactions. **Note:** For more information about the transaction types supported by the plug-in, see Transaction Types Supported By the SuiteTax Plug-in.
Returns	string
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
      var input = context.input;
      ...
      var entityType = input.entityType;
      var entityID = input.entity;

      if(input.recordType == 'SalesOrd') {
          var customerRecord = record.load({
            type: entityType,
            id: entityID
          });
      ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

shipFromAddress

Property Declaration	`Address shipFromAddress`
Type	Transaction-level object property
Description	Returns an Address object that represents the ship from address of the current transaction being processed by the plug-in implementation. See Values on an Address Object for more information about from where the possible values for this address are sourced, depending on the transaction type. This property returns null if there is no shipping address specified. Use this object, for example, to calculate tax amounts based on the shipping address for where the transaction originates. You can calculate tax amounts based on the values of the address, including city, state, zip code, or country. **Note:** You can also use location to load the location record for the Customer or Vendor.
Returns	Address | NULL
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {

  function calculateTax(context){
        ...
        var input = context.input;
        var address = input.shipFromAddress;
        var amount = input.subtotal;

        var totalTax = calculateTaxByLocation(
          address.country,
          amount
        );
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

shipToAddress

Property Declaration	`Address shipToAddress`

Type	Transaction-level object property
Description	Returns an Address object that represents the ship to address of the current transaction being processed by the plug-in implementation. See Values on an Address Object for more information about from where the possible values for this address are sourced, depending on the transaction type. This property returns null if there is no shipping address specified. Use this object, for example, to calculate tax amounts based on the shipping address for the destination. You can calculate tax amounts based on the values of the address, including city, state, zip code, or country.
Returns	Address | NULL
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {

  function calculateTax(context){
        ...
        var input = context.input;
        var address = input.shipToAddress;
        var amount = input.subtotal;

        var totalTax = calculateTaxByLocation(
          address.country,
          amount
        );
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

subsidiary

Property Declaration	`Number subsidiary`
Type	Transaction-level object property
Description	Returns the internal NetSuite ID of the subsidiary record associated with the current transaction. The subsidiary is located under Classification on a sales order main page. Use this property to define plug-in implementation functionality based on the subsidiary. For example, you can use this ID with the SuiteScript API record.load(options) and **subsidiary** record type to access the subsidiary record associated with the transaction. For more information, see the following topics:

- record.load(options)
- SuiteScript Supported Records

Note: For non-One World accounts, this method returns '1'.

Returns	number
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
        var input = context.input;
        ...
        var subsidiaryID = input.subsidiary;
        var subsidiary = record.load({
          type: record.Type.SUBSIDIARY,
          id: subsidiaryID
        });
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

subtotal

Property Declaration	`String subtotal`
Type	Transaction-level object property
Description	Returns the subtotal for a transaction as a string. The value of the subtotal is calculated by adding the Amount property of all line items in a transaction. This is the total before any discounts, shipping cost, or handling cost is added to the transaction. Use this method to define plug-in implementation functionality based on the total value of all items in a transaction.
Returns	string
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
```

```
 */
define([], function() {
    function calculateTax(context){
        var input = context.input;
        ...
        var totalAmount = input.subtotal + input.discountTotal;
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

taxRegistration

Property Declaration	`Number taxRegistration`
Type	Transaction-level object property
Description	Returns the internal NetSuite ID for the tax registration number associated with the nexus for a taxable transaction. Use this property, for example, to check the country of origin for purchased items on a sales order or invoice. **Note:** As of Version 2015 Release 2, the plug-in developer must maintain the mapping between tax registration IDs and their corresponding values.
Returns	number
Parent object	TaxCalculationInput

Example

```
/**
 * @NApiVersion 2.0
 * @NScriptType taxCalculationPlugin
 */
define([], function() {
    function calculateTax(context){
        ...
        var taxRegMap = {
          123: 'UK1092108',
          136: 'DE02839892'
        };
        var taxRegNumber = taxRegMap[context.input.taxRegistration];
        if (taxRegNumber.substring(0, 2) == 'DE')
        {
            ...
        }
        ...
    }

    return {
        calculateTax: calculateTax
```

```
    }
}
```

transactionDate

Property Declaration	`TaxCalculationDate transactionDate`
Type	Transaction-level object property
Description	Returns a TaxCalculationDate object that contains the date value from the transaction record. Use this object to define plug-in implementation functionality based on the transaction date. You can define plug-in implementation functionality based on the values of the day, month, or year.
Returns	TaxCalculationDate
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context){
        var input = context.input;
        ...
        var transDate = input.transactionDate;
        var address = input.shipToAddress;
        var amount = input.subtotal;

        var totalTax = calculateTaxByLocation(
          address.country,
          amount,
          transDate
        );
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

transactionId

Property Declaration	`Number transactionId`
Type	Transaction-level object property
Description	Returns the NetSuite internal ID for the current transaction.

Use this property to define plug-in functionality based on the transaction record ID. For example, you can use this ID with the SuiteScript API record.load(options) to access the record for the transaction.

This property returns null if the transaction ID does not yet exist, for example, before you save a transaction for the first time.

Returns number

Parent object TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
      var input = context.input;
      ...
      var transID = input.transactionId;
      var transType = input.recordType;

      var transRec = record.load({
        type: transType,
        id: transID
      });
      ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

transformationSourceTransactions

Property Declaration `TaxCalculationInputSourceTransaction[] transformationSourceTransactions`

Type Transaction-level object property

Description Returns an array of TaxCalculationInputSourceTransaction objects.

Use this property to define plug-in functionality based on the record or records from which the current transaction originated. For example, if you created a sales order from an opportunity, use this property to access the record ID and record type of the opportunity record from the sales order transaction. You can then use record.load(options) to access the source record for the current transaction record.

If a transaction has more than one originating transaction, this method returns an array of TaxCalculationInputSourceTransaction objects, where each object represents one of the source transactions. The list of source transactions is built using the following sources, in this order:

1. Created From field – This field specifies a single transaction which is used as the source of all values for the newly created transaction.

2. PO Doc Number field – This field is specific to vendor bills which are created from a single purchase order.
3. Purchase Orders sublist – This sublist is specific to vendor bills which are created from one or more purchase orders. In this case, it is possible for the method to return multiple source transactions.

For example, you create an item fulfillment from a sales order, create an invoice from the item fulfillment, and then calculate taxes on the invoice. This method returns an array that contains only one `TaxCalculationInputSourceTransaction` object which references the sales order.

If there are no source transactions for the current transaction, this method returns an empty array.

For information about related transactions in NetSuite, see the help topic Using Transaction Links.

Returns	TaxCalculationInputSourceTransaction[]
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context){
        var input = context.input;
        ...
        var sourceTrans = input.transformationSourceTransactions;

        if (sourceTrans.length > 0) {
            for (
              var sourceTransIndex = 0;
              sourceTransIndex < sourceTrans.length;
              sourceTransIndex++
            )
            {
                currTrans = sourceTrans[sourceTransIndex];
                ...
          }
         }
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

postingTransaction

Property Declaration	`Boolean postingTransaction`
Type	Transaction-level object property

Description	Returns true if the current transaction is a posting transaction. Otherwise, this method returns false. For more information about posting transactions versus non-posting transactions in NetSuite, see the help topic General Ledger Impact of Transactions.
Returns	boolean
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context){
        var input = context.input;
        ...
        if(input.postingTransaction) {
            ...
            var periodEnd = input.postingPeriodEndDate;
            var periodStart = input.postingPeriodStartDate;

            var totalTax = calculateTaxByPeriod(
              periodStart,
              periodEnd
            );
            ...
        }
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

preview

Property Declaration	`Boolean preview`
Type	Transaction-level object property
Description	Returns true if the user clicks the **Preview Tax** button to trigger the tax calculation. Otherwise this property returns false, indicating that the user saved the transaction. Use this property to optimize the tax calculation for a preview of tax details.
Returns	boolean
Parent object	TaxCalculationInput

Example

```
/**
```

```
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context){
      var input = context.input;
      ...
      if (input.preview)
      {
          // Calculate taxes for preview
          ...
      }
      else
      {
          // Calculate taxes for final calculation
          ...
      }
      ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

taxOutputOverridden

Property Declaration	`Boolean taxOutputOverridden`
Type	Transaction-level object property
Description	Returns true if the **Tax Details Override** field is checked on the Tax Details subtab of a transaction.
Returns	boolean
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context){
        var input = context.input;
        ...
        if (context.input.taxOutputOverridden)
        {
            // check that the overridden values are valid, add an error message if not
            verifyOutputOverrideLines(
              context.output.lines,
              context.notifications
```

```
            );
            return;
        }
        ...
        // Calculate taxes, add them to the output object
       ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

taxRegistrationOverridden

Property Declaration	`Boolean taxRegistrationOverridden`
Type	Transaction-level object property
Description	Returns true if the user overrides the **Tax Registration Number** field on the Tax Details subtab of a transaction and selects a value other than what was determined by NetSuite for the nexus.
Returns	boolean
Input Parameters	None
Parent object	TaxCalculationInput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context){
      var input = context.input;
      ...
      if (input.taxRegistrationOverridden)
      {
          // Calculate taxes based for the overridden registration
          ...
      }
      else
      {
          // Use a different method to calculate taxes
          ...
      }
      ...
    }

    return {
        calculateTax: calculateTax
    }
```

```
}
```

TaxCalculationInputLine

Type	Object
Description	Object that contains the properties of a single line or line item on a taxable transaction. Each transaction passed to the plug-in implementation has an `TaxCalculationInputLine` object associated with it. Use this object to get the details for a specific line item and calculate tax based on the item properties. Use the `TaxCalculationInputLine` object methods and properties to retrieve the line item properties.
Method	■ getAdditionalFieldValue(options)
Properties	For Lines: ■ amount ■ amountIncludeTax ■ discounts ■ discountsTotal ■ lineType ■ reference For lines of type "Item" ■ itemId ■ itemRecordType ■ quantity ■ shipFromAddress ■ shipToAddress **Important:** Methods for objects with lineType = "Item" support the itemId, itemRecordType, quantity, shipFromAddress, and shipToAddress properties. These methods are not supported for other types of lines such as expenses. You can use the `TaxCalculationInputLine`.lineType property to get the line type of the current line item.
Parent Object	TaxCalculationInput

amount

Property Declaration	`String amount`
Type	Line-level object property
Description	Returns the total amount for a transaction line item, based on the item amount and item quantity. The value may either be a Net Amount or a Gross Amount, see amountIncludeTax for details.
Returns	string
Parent object	TaxCalculationInputLine

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context){
        var input = context.input;
        ...
        var totalAmount = 0;
        var inputLines = input.lines;
        for (
          var inputLineIndex = 0;
          inputLineIndex < inputLines.length;
          inputLineIndex++
        )
        {
            totalAmount += inputLines[inputLineIndex].amount;
            ...
        }
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

amountIncludeTax

Property Declaration	`Boolean amountIncludeTax`
Type	Line-level object property
Description	Returns true when the amount property represents a Gross Amount, otherwise returns false when it represents a Net Amount.
Returns	Boolean
Parent object	TaxCalculationInputLine

Example

```
/**
* @NApiVersion 2.0
* @NScriptType taxCalculationPlugin
*/
define(['N/record'], function(record) {
    function calculateTax(context) {
        ...
        var lineTaxAmount;
        var taxRate = getTaxRate();

        if (inputLine.amountIncludeTax === true)
```

```
        {
                        var lineGrossAmount = inputLine.amount;
                        lineTaxAmount = lineGrossAmount - (lineGrossAmount / (1 + taxRate));
        }
        else
        {
                        var lineNetAmount = inputLine.amount;
                        lineTaxAmount = lineNetAmount * taxRate;
        }
        ...
    }

    function getTaxRate() {
            return 0.15;
    }

    return {
            calculateTax: calculateTax
    }
});
```

getAdditionalFieldValue(options)

Function Declaration	`String getAdditionalFieldValue(options)`
Type	Line-level object method
Description	Returns the value for the field on the line item passed as an input parameter. The field can be a standard NetSuite field or a custom field added to the item form. Use this method if you want to request additional field values sent to the plug-in implementation that are not available in the methods for TaxCalculationInputLine. To use this as a parameter, you must complete the following steps: ■ If you want to use a custom field, add the custom field to the item form. ■ Specify the names of the fields that you want NetSuite to pass to the plug-in implementation using defineAdditionalFields(context).
Returns	string
Input Parameters	`options.fieldId {string}` — Field ID of a standard or custom field on the item form.
Parent object	TaxCalculationInputLine

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context){
        var input = context.input;
        ...
        var totalAmount = 0;
```

```
        var inputLines = input.lines;
        for (
          var inputLineIndex = 0;
          inputLineIndex < inputLines.length;
          inputLineIndex++
        )
        {
            totalAmount += inputLines[inputLineIndex].amount;
            fieldValue = inputLines[inputLineIndex].getAdditionalFieldValue('memo');
            ...
        }
        ...
    }

        function defineAdditionalFields(context){
        context.addField({
          fieldId: 'memo'
        });
    }

    return {
        calculateTax: calculateTax,
        defineAdditionalFields: defineAdditionalFields
    }
}
```

discounts

Property Declaration	`TaxCalculationInputDiscountDetail[] discounts`
Type	Line-level object property
Description	Returns an array of TaxCalculationInputDiscountDetail objects.
Returns	TaxCalculationInputDiscountDetail[]
Parent object	TaxCalculationInputLine

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context){
        var input = context.input;
        ...
        var lines = input.lines;
        for (
          var i = 0;
          i < lines.length;
          i++
        )
        {
```

```
            var line = lines[i];
            var discountDetails = line.discounts;
            context.notifications.addNotice('
              Line #'
              + i +
              ' has '
              + discountDetails.length +
              ' discounts for a total of '
              + line.discountsTotal + '
              ' + currencySymbol
            );
        }
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

discountsTotal

Property Declaration	`String discountsTotal`
Type	Line-level object property
Description	Returns the total of all discounts applied to a transaction line as a string. If there are no discounts applied to a line item, this property returns 0.
Returns	string
Parent object	TaxCalculationInputLine

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context){
        var input = context.input;
        ...
        var lines = input.lines;
        for (
          var i = 0;
          i < lines.length;
          i++)
        {
            var line = lines[i];
            var discountDetails = line.discounts;
            context.notifications.addNotice(
              'Line #'
              + i +
```

```
                ' has '
                + discountDetails.length +
                ' discounts for a total of '
                + line.discountsTotal + '
                ' + currencySymbol);
        }
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

lineNumber

Property Declaration	`Number lineNumber`
Type	Line-level object property
Description	Returns the line number associated with a line item on a transaction. Each item in a transaction has an TaxCalculationInputLine object associated with the item. Each `TaxCalculationInputLine` object represents a single line item on the transaction. This method returns the line number for that item. The line number appears on the **Tax Detail** subtab of a transaction after you save it for the first time. Use this property to define plug-in implementation functionality based on the line number associated with a line item or use the line number for testing or debugging purposes.
Returns	number
Parent object	TaxCalculationInputLine

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context){
        var input = context.input;
        ...
        var totalAmount = 0;
        var inputLines = input.lines;
        for (
          var inputLineIndex = 0;
          inputLineIndex < inputLines.length;
          inputLineIndex++
        )
        {
            totalAmount += inputLines[inputLineIndex].amount;
            var itemLine = inputLines[inputLineIndex].lineNumber;
            ...
        }
```

```
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

lineType

Property Declaration	`TaxCalculationInputLineType lineType`
Type	Line-level object property
Description	Returns the line type of the current line item on a transaction as a `TaxCalculationInputLineType enum` value. Use this property to determine if the current line item is a NetSuite item type, a shipping or handling line type, or an expense line type and then calculate tax using the line type. For more information, see TaxCalculationInputLineType.
Returns	string
Parent object	TaxCalculationInputLine

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {

    function calculateTax(context){
        var inputLines = context.input.lines;
        ...
        for (var inputLineKey in inputLines)
        {
            var inputLine = inputLines[inputLineKey];

            if (inputLine.lineType == 'item')
            {
                // Calculate tax for item
               ...
            }
            else if (
              (inputLine.lineType == 'shipping') ||
              (inputLine.lineType == 'handling')
            )
            {
                // Calculate tax for shipping / handling
                ...
            }
            ...
            }
```

```
    }

    return {
        calculateTax: calculateTax
    }
});
```

reference

Property Declaration	`TaxCalculationInputLineReference reference`
Type	Line-level object property
Description	Returns a reference to the line item as a TaxCalculationInputLineReference object. Use the `TaxCalculationInputLineReference` object to create an associated tax output line with createLine(options). You can use the `TaxCalculationInputLineReference` object to uniquely refer to the line in a transaction.
Returns	TaxCalculationInputLineReference
Parent object	TaxCalculationInputLine

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
        function calculateTax(context) {
        ...
        var inputLines = context.input.lines;
        for (var inputLineKey in inputLines) {
            var inputLine = inputLines[inputLineKey];
            var outputLine = output.createLine({
              lineReference: inputLine.reference
            });
            ...
        }
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

itemId

Property Declaration	`Number itemId`
Type	Line-level object property

Description	Returns the internal NetSuite ID for a transaction line item. The type of NetSuite record represented by the ID depends on the TaxCalculationInputLineType for the transaction line item, returned by lineType. For a line type of ITEM, this property returns the internal NetSuite ID for an Item record. For a line type of SHIPPING or HANDLING, the ID represents the shipping method record for the line item. You can use this property, for example, with the SuiteScript API record.load(options) to access the item or shipping method record for the line item. For more information about item types in NetSuite, see the help topics Using Item Records and Item Types.
Returns	number

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {

    function calculateTax(context){
        ...
        if (inputLine.lineType == 'item')
        {
            var itemId = inputLineItem.itemId;
            ...
        }
        else if (
          (inputLine.lineType == 'shipping') ||
          (inputLine.lineType == 'handling')
        )
        {
            var shippingId = inputLineItem.itemId;
            ...
        }
    }

    return {
        calculateTax: calculateTax
    }
});
```

itemRecordType

Property Declaration	`String itemRecordType`
Type	Line-level object property
Description	Use this property to define plug-in implementation functionality based on the item type associated with a line item. For example, different item types may require different types

of tax calculation methods. Additionally, you can use this property with the item ID and the SuiteScript API record.load(options) to access the item record for the line item.

For more information about item types in NetSuite, see the help topics Using Item Records and Item Types.

Important: This item type is not the same type as returned by lineType. In addition, this method returns a NULL value if you use it on an object with a type other than ITEM.

Returns string

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context) {
        var totalAmount = 0;
        var totalTax = 0;
        var inputLines = context.input.lines;
        for (
          var inputLineIndex = 0;
          inputLineIndex < inputLines.length;
          inputLineIndex++
        ) {
            var line = inputLines[inputLineIndex];
            totalAmount += line.amount;

            var itemType = line.itemRecordType;
            if (itemType == 'Service') {
                // Calculate tax for services
                totalTax += calculateServiceTax(inputLine);
            } else {
                ...
            }
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

quantity

Property Declaration	`String quantity`
Type	Line-level object property
Description	Use this property to define plug-in implementation functionality based on the quantity of an item associated with a line item.

Important: This method returns a NULL value if used on an object with a type other than ITEM.

Returns string

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context){
        var totalAmount = 0;
        var inputLines = var context.input.lines;
        for (
          var inputLineIndex = 0;
          inputLineIndex < inputLines.length;
          inputLineIndex++
        ) {
            var line = inputLines[inputLineIndex];
            var quantity = line.quantity;
            totalAmount += line.amount;
            ...
        }
    }

    return {
        calculateTax: calculateTax
    }
}
```

shipFromAddress

Property Declaration `Address shipFromAddress`

Type Line-level object property

Description Returns an Address object that represents the ship from address of the current transaction line item. See Values on an Address Object for more information about from where the possible values for this address are sourced, depending on the transaction type.

This property returns null if there is no shipping address specified.

Use this object, for example, to calculate tax amounts based on the shipping address for where the transaction originates. You can calculate tax amounts based on the values of the address, including city, state, zip code, or country.

Note: You can also use location to load the location record.

Returns Address | NULL

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context) {
        ...
        var inputLines = context.input.lines;
        for (
          var inputLineIndex = 0;
          inputLineIndex < inputLines.length;
          inputLineIndex++
         ) {
            var address = inputLines[inputLineIndex].shipFromAddress;
            var city = address.city;
            var state = address.state;
            var zipCode = address.zip;
            var country = address.country;
            ...
        }
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

shipToAddress

Property Declaration	`Address shipToAddress`
Type	Line-level object property
Description	Returns an Address object that represents the ship to address of the current transaction line item. See Values on an Address Object for more information about from where the possible values for this address are sourced, depending on the transaction type. This property returns null if there is no shipping address specified. Use this object, for example, to calculate tax amounts based on the shipping address for the destination. You can calculate tax amounts based on the values of the address, including city, state, zip code, or country. **Note:** You can also use location to load the location record.
Returns	Address | NULL
Input Parameters	None

Example

```
/**
```

```
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context) {
        ...
        var inputLines = context.input.lines;
        for (
          var inputLineIndex = 0;
          inputLineIndex < inputLines.length;
          inputLineIndex++) {
            var address = inputLines[inputLineIndex].shipToAddress;
            var city = address.city;
            var state = address.state;
            var zipCode = address.zip;
            var country = address.country;
            ...
        }
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

Address

Type	Object
Description	Container object that contains methods and properties to access addresses. These include the following types of addresses associated with a transaction: ■ Transaction-level billing addresses. Use `TaxCalculationInput`.billFromAddress or `TaxCalculationInput`.billToAddress. ■ Transaction-level shipping addresses. Use `TaxCalculationInput`.shipFromAddress or `TaxCalculationInput`.shipToAddress. These include the following types of addresses associated with a line item: ■ Line item-level shipping addresses. Use `TaxCalculationInputLine`.shipFromAddress or `TaxCalculationInputLine`.shipToAddress. Use the following guidelines with this object: ■ See Values on an Address Object for more information about from where the possible values for the address are sourced, depending on the transaction type. ■ Each of the methods for the Address object return null if the property is not specified on the transaction. ■ You can use the getFieldValue(options) method to get the values of standard and custom address fields. Use the **Label** property for the custom address field as the `options.fieldName` parameter. For more information about custom address fields, see the help topic Creating Custom Address Fields.
Parent Object(s)	TaxCalculationInput

The following table describes the methods for the Address object:

Method	Description	Return Type
addr1	Returns the first line of an address.	string | NULL
addr2	Returns the second line of an address.	string | NULL
addr3	Returns the third line of an address.	string | NULL
addressee	Returns the addressee.	string | NULL
attention	Returns the name of specific person to whom a shipment is addressed if the **Addressee** is a Company or Department.	string | NULL
city	Returns the city.	string | NULL
country	Returns the country as a two-digit country code as defined by ISO 3166-1 alpha-2.	string | NULL
phone	Returns the phone number.	string | NULL
state	Returns the state.	string | NULL
zip	Returns the Zip Code (Postal Code).	string | NULL
getFieldValue(options) options.fieldName	Returns the value of a standard or custom address field.	string | NULL

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context) {
            var address;
            ...
            // Get address
            ...
            var city = address.city;
            var state = address.state;
            var zipCode = address.zip;
            var country = address.country;
           ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

TaxCalculationDate

Type	Transaction-level Object

Description	Object that represents the date value from the transaction record or the start or end date value of an accounting period. For example, on a sales order, the transactionDate property returns an object that represents the **Date** field on the transaction. Use this object to define plug-in functionality based on a date. You can define plug-in implementation functionality based on the values of the day, month, or year. The year is represented as a 4 digit integer and day and month are two digit integers. You can access the `TaxCalculationDate` object with the transactionDate, postingPeriodStartDate, and postingPeriodEndDate properties.
Parent Object(s)	TaxCalculationInput

The following table lists the methods for the `TaxCalculationDate` object:

Method	Return Type
day	number
month	number
year	number

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context) {
        var input = context.input;
        ...
        var transDate = input.transactionDate;
        var day = transDate.day;
        var month = transDate.month;
        var year = transDate.year;
        ...
    }

    return {
        calculateTax: calculateTax
    }
}
```

TaxCalculationInputLineType

Type	Line-item level object
Description	Object that represents the line type of an TaxCalculationInputLine object. This object has four enumerated values: ■ ITEM ■ SHIPPING ■ HANDLING

- EXPENSE

Use this value to calculate tax differently for different line types. Retrieve this value with lineType.

For example, on Purchase Order and Vendor Bill transactions, a line item on the transaction may be of type `TaxCalculationInputLineType.ITEM` or `TaxCalculationInputLineType.EXPENSE`. You can use this value to differentiate between the two types.

Parent Object(s) TaxCalculationInputLine

TaxCalculationInputLineReference

Type Line-item level object

Description Object that identifies an individual input line item on a transaction. Access and use this reference through the following objects:

- TaxCalculationInput. Each line item on the input to the SuiteTax calculation plug-in is represented by this object. Use reference to retrieve a reference to the object.
- TaxCalculationOutput. When you calculate tax on a transaction, you associate an output tax line with the input line items. Use createLine(options) to create a new tax line for an input line item. Also, use inputLineReference to get the reference to an input line associated with an output tax line through the interface output object.

Methods

- lineKey

Parent Object(s) TaxCalculationInputLine

lineKey

Function Declaration `String lineKey`

Type Object property

Description Returns a string that uniquely identifies a TaxCalculationInputLineReference object.

Returns string

Parent object TaxCalculationInputLineReference

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context){
        ...
        lineTaxAmountMap[inputLine.reference.lineKey] = 0;
        ...
        lineTaxAmountMap[outputLine.inputLineReference.lineKey] += lineTaxDetailAmount;
        ...
        for (var lineKey in lineTaxAmountMap)
        {
```

```
            var lineTaxAmount = lineTaxAmount[lineKey];
            ...
        }
    }

    return {
        calculateTax: calculateTax
    }
}
```

TaxCalculationInputSourceTransaction

Type	Transaction-level object
Description	Object that represents the transaction from which the current transaction being processed by the plug-in implementation originated. For example, if you created a sales order from an opportunity, and the current transaction is the sales order, this object represents the Opportunity record. Use this object to access the internal NetSuite ID for the source record and the source record type. For more information, see transformationSourceTransactions.
Methods	■ recordId ■ recordType
Parent Object(s)	TaxCalculationInput
Child Object(s)	n/a

recordId

Property Declaration	`Number recordId`
Type	Object property
Description	Returns the internal NetSuite ID for a source record represented by a TaxCalculationInputSourceTransaction object. Use this property to get properties for the record that originated the current record being processed by a SuiteTax plug-in implementation. For example, you can use this property and recordType with the SuiteScript API record.load(options) to access the source record for the current transaction. For more information, see transformationSourceTransactions.
Returns	number
Parent object	TaxCalculationInputSourceTransaction

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
```

```
  */
define(['N/record'], function(record) {
    function calculateTax(context){
        var sourceTransactions = context.input.transformationSourceTransactions;

        for (
          var i = 0;
          i < sourceTransactions.length;
          i++
        ) {
            var transaction = sourceTransactions[i];
            var transactionId = transaction.recordId;
            var recordType = transaction.recordType;
            var record = record.load({
              type: recordType,
              id: transactionId
            });
            ...
            }
    }

    return {
        calculateTax: calculateTax
    }
}
```

recordType

Property Declaration	`String recordType`
Type	Object property
Description	Returns the type of a source record represented by a TaxCalculationInputSourceTransaction object. Use this property to get properties for the record that originated the current record being processed by a SuiteTax plug-in implementation. For example, you can use this property and recordId with the SuiteScript API record.load(options) to access the source record for the current transaction. For more information, see transformationSourceTransactions.
Returns	string
Parent object	TaxCalculationInputSourceTransaction

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
        var sourceTransactions = context.input.transformationSourceTransactions;
```

```
        for (
          var i = 0;
          i < sourceTransactions.length;
          i++
        ) {
            var transaction = sourceTransactions[i];
            var transactionId = transaction.recordId;
            var recordType = transaction.recordType;
            var record = record.load({
              type: recordType,
              id: transactionId
            });
            ...
        }
    }

    return {
        calculateTax: calculateTax
    }
}
```

TaxCalculationInputDiscount

Type	Transaction-level object
Description	Object that represents a discount applied to a transaction.
Methods	■ amount ■ itemId reference
Parent Object(s)	TaxCalculationInput
Child Object(s)	■ TaxCalculationInputDiscountDetail ■ TaxCalculationInputDiscountReference

amount

Property Declaration	`String amount`
Type	Transaction-level object property
Description	Returns the amount of the discount.
Returns	string
Parent object	TaxCalculationInputDiscount

Example

```
/**
  * @NApiVersion 2.0
```

```
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
        ...
        var discounts = context.input.discounts;
        if (discounts)
        {
            for (
              var i = 0;
              i < discounts.length;
              i++
            )
            {
            var d = discounts[i];
            context.notifications.addNotice('Discount '' + d.reference + '' '
                + ' with item ID ' + d.itemId + ' has an amount of ' + d.amount);
            }
        }
    }

    return {
        calculateTax: calculateTax
    }
});
```

itemId

Property Declaration	`Number itemId`
Type	Transaction-level object method
Description	Returns the internal NetSuite ID for a discount item. For more information about item types in NetSuite, see the help topics Discount Items and Creating Item Records.
Returns	number
Parent object	TaxCalculationInputDiscount

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
        ...
        var discounts = context.input.discounts;
        if (discounts)
        {
            for (
              var i = 0;
```

```
            i < discounts.length;
            i++
          )
          {
              var d = discounts[i];
              context.notifications.addNotice('Discount '' + d.reference + '' '
                  + ' with item ID ' + d.itemId + 'an amount of ' + d.amount);
          }
      }
  }

  return {
      calculateTax: calculateTax
  }
});
```

reference

Property Declaration	`TaxCalculationInputDiscountReference reference`
Type	Transaction-level object property
Description	Returns a reference to a transaction-level discount as a `TaxCalculationInputDiscountReference` object.
Returns	`TaxCalculationInputDiscountReference`
Parent object	TaxCalculationInputDiscount

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
        ...
        var discounts = context.input.discounts;
        if (discounts)
        {
            for (
              var i = 0;
              i < discounts.length;
              i++
            )
            {
                var d = discounts[i];
                context.notifications.addNotice('Discount '' + d.reference + '' '
                    + ' with item ID ' + d.itemId + ' has an amount of ' + d.amount);
            }
        }
    }

    return {
```

```
        calculateTax: calculateTax
    }
});
```

TaxCalculationInputDiscountDetail

Type	Line-level object
Description	Object that represents a discount portion applied to a single line of a transaction.
Methods	■ amount ■ discountReference
Parent Object(s)	TaxCalculationInputDiscount
Child Object(s)	n/a

amount

Property Declaration	`String amount`
Type	Line-level object property
Description	Returns the amount of the discount portion applied to a single line of a transaction.
Returns	string
Parent object	TaxCalculationInputDiscountDetail

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
        var input = context.input;
        var discounts = input.discounts;
        var discountsMap = {};
        if (discounts)
        {
            for (
              var i = 0;
              i < discounts.length;
              i++)
            {
                var discount = discounts[i];
                discountsMap[discount.reference] = discount; // save the discount by its key
            }
        }
        ...
        var lines = input.lines;
```

```
        for (
          var i = 0;
          i < lines.length;
          i++
        )
        {
            var line = lines[i];
            var lineDiscounts = line.discounts;
            for (var j = 0; j < lineDiscounts.length; j++)
            {
                var detail = lineDiscounts[j];
                var discount = discountsMap[detail.discountReference]; // load the 'parent' discount associated
 to this discount detail
                context.notifications.addNotice(
                  'Line #'
                  + i + '
                  has a discount portion '
                  + detail.amount +
                  ' out of '
                  + discount.amount
                );
            }
    }

    return {
        calculateTax: calculateTax
    }
});
```

discountReference

Property Declaration	`TaxCalculationInputDiscountReference discountReference`
Type	Line-level object property
Description	Returns a reference to the TaxCalculationInputDiscount which this TaxCalculationInputDiscountDetail is part of.
Returns	TaxCalculationInputDiscountReference
Parent object	TaxCalculationInputDiscountDetail

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
        var input = context.input;
        var discounts = input.discounts;
        var discountsMap = {};
        if (discounts)
        {
```

```
            for (
              var i = 0;
              i < discounts.length;
              i++)
            {
                var discount = discounts[i];
                discountsMap[discount.reference] = discount; // save the discount by its key
            }
        }
        ...
        var lines = input.lines;
        for (
          var i = 0;
          i < lines.length;
          i++
        )
        {
            var line = lines[i];
            var lineDiscounts = line.discounts;
            for (
              var j = 0;
              j < lineDiscounts.length;
              j++
            )
            {
                var detail = lineDiscounts[j];
                var discount = discountsMap[detail.discountReference]; // load the 'parent' discount associated
 to this discount detail
                context.notifications.addNotice(
                  'Line #'
                  + i + '
                  has a discount portion '
                  + detail.amount +
                  ' out of '
                  + discount.amount
                );
            }
        }

    return {
        calculateTax: calculateTax
    }
});
```

TaxCalculationInputDiscountReference

Type	Object
Description	Object that provides a reference to a TaxCalculationInputDiscount object.
Properties	▪ key
Parent Object(s)	TaxCalculationInputDiscount
Child Object(s)	n/a

key

Property Declaration	`String key`
Type	Object property
Description	Returns a string that uniquely identifies a TaxCalculationInputDiscountReference object.
Returns	string
Parent object	TaxCalculationInputDiscountReference

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {

    function calculateTax(context){
        ...
        var input = context.input;
        // for a transaction with exactly one discount:
        var discount = input.discounts[0];
        ...
        // for a transaction line with discounts applied to it:
        var discountDetail = input.lines()[0].discounts()[0];
        ...
        // discount details and their 'parent' discounts are identified by a matching reference key
        discount.geference.key == discountDetail.giscountReference.key
        ...
    }

    return {
        calculateTax: calculateTax
    }
});
TaxCalculationOutput
```

TaxCalculationOutput

Type	Interface output object
Description	Contains all output values produced by the tax calculation performed by a SuiteTax plug-in implementation.
Properties	■ addLine(options) ■ createLine(options) ■ nexus ■ taxRegistration ■ taxSummaryLines ■ overrideNexus(options)

- setTaxSummaryLine(options)

Child Objects

- TaxCalculationOutputLine
- TaxCalculationOutputSummaryLine

addLine(options)

Function Declaration	`void addLine(options)`
Type	Transaction-level object property
Description	Adds an TaxCalculationOutputLine object to the TaxCalculationOutput interface output object. Use this method to add the **`TaxCalculationOutputLine`** object to the interface output object to pass the tax values calculated by the plug-in implementation back to NetSuite. Each line item on the transaction for which you want to charge taxes must have an associated **`TaxCalculationOutputLine`** object added to the interface output object. Use createLine(options) to create the **`TaxCalculationOutputLine`** object before you add it to the interface output object.
Returns	void
Input Parameters	`TaxCalculationOutputLine options.outputLine` — Object created with createLine(options) that contains all tax details for a line item on a transaction.
Parent object	TaxCalculationOutput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {

    function calculateTax(context){
        ...
        var input = context.input;
        var inputLines = input.lines;

        for (
          var inputLineIndex = 0;
          inputLineIndex < inputLines.length;
          inputLineIndex++
        )
        {
              // create the tax line
              outputLine = output.createLine({lineReference: inputLines[inputLineIndex].reference});
              // get the line amount
              amount = inputLines[inputLineIndex].amount;
              // Calculate the tax

              ...

              // add the tax detail to the taxes for the line item
```

```
            outputLine.addTaxDetail({
                taxCode: taxCode,
                taxationType: taxType,
                taxRate: taxRate,
                taxAmount: taxAmount,
                taxBasis: taxBasis,
                taxCalculationDetail: description
            });

            // add the TaxCalculationOutputLine tax detail to the output object
            output.addLine({outputLine: outputLine});
           ...
        }
    }

    return {
        calculateTax: calculateTax
    }
});
```

createLine(options)

Function Declaration	`TaxCalculationOutputLine createLine(options)`
Type	Transaction-level object property
Description	Creates a TaxCalculationOutputLine object for a specific line item reference. Each line item on the transaction for which you want to charge taxes must have an associated `TaxCalculationOutputLine` object added to the TaxCalculationOutput interface output object. If the line does not exist in the TaxCalculationInput interface input object, this method throws an error.
Returns	void
Input Parameters	`TaxCalculationInputLineReference options.lineReference` — Reference to the corresponding input line for which you want to create an output line and add tax details.
Parent object	TaxCalculationOutput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {

    function calculateTax(context){
        ...
        var input = context.input;
        var inputLines = input.lines;

        for (
```

```
          var inputLineIndex = 0;
          inputLineIndex < inputLines.length;
          inputLineIndex++
        )
        {
              // create the tax line
              outputLine = output.createLine({lineReference: inputLines[inputLineIndex].reference});
              // get the line amount
              amount = inputLines[inputLineIndex].amount;
              // Calculate the tax

              ...

              // add the tax detail to the taxes for the line item
              outputLine.addTaxDetail({
                  taxCode: taxCode,
                  taxationType: taxType,
                  taxRate: taxRate,
                  taxAmount: taxAmount,
                  taxBasis: taxBasis,
                  taxCalculationDetail: description
              });

              // add the TaxCalculationOutputLine tax detail to the output object
              output.addLine({outputLine: outputLine});
              ...
        }
    }

    return {
        calculateTax: calculateTax
    }
});
```

lines

Property Declaration	`TaxCalculationOutputLine[] lines`
Type	Transaction-level object property
Description	Returns an array of TaxCalculationOutputLine objects that you added to the interface output object with addLine(options). Use this property to access all tax details added to the interface output object for all line items in the transaction. This property returns an empty array if no **`TaxCalculationOutputLine`** objects exist.
Returns	TaxCalculationOutputLine[]
Parent object	TaxCalculationOutput

Example

```
/**
  * @NApiVersion 2.0
```

```
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {

    function calculateTax(context){
        ...
        var taxItemLines = context.output.lines;

        for (
          var itemLineTaxIndex = 0;
          itemLineTaxIndex < taxItemLines.length;
          itemLineTaxIndex++
        )
        {
            // access all tax details added to transaction
            currentLine = taxItemLines[itemLineTaxIndex].inputLineReference.lineKey;
        }
    }

    return {
        calculateTax: calculateTax
    }
});
```

nexus

Property Declaration	`Number nexus`
Type	Transaction-level object property
Description	Returns the internal NetSuite ID for the nexus on a transaction. Use this property to access the actual transaction nexus in the TaxCalculationOutput interface output object. If the nexus is successfully overridden using the overrideNexus(options) method, this property returns the same value that was set by the overrideNexus(options) method. Otherwise, this property returns the same value as the nexus property in the TaxCalculationInput object.
Returns	number
Parent object	TaxCalculationOutput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    var deNexus = 7;
    var deTaxCode = 2;
    var deTaxType = 8;

    function calculateTax(context){
        ...
```

```
        var output = context.ouput;
        output.overrideNexus({nexus: deNexus});
        ...
        if (output.nexus == deNexus)
        {
            taxCode = deTaxCode;
            taxType = deTaxType;
        }
        ...
    }

    return {
        calculateTax: calculateTax
    }
});
```

taxRegistration

Property Declaration	`Number taxRegistration`
Type	Transaction-level object property
Description	Returns the internal NetSuite ID for the tax registration number associated with the nexus for a taxable transaction. Use this property to access the actual tax registration of the transaction in the TaxCalculationOutput interface output object. If the nexus is successfully overridden using the overrideNexus(options) method, this property returns the tax registration for the nexus that was set by the overrideNexus(options) method. Otherwise, this property returns the same value as the taxRegistration property in the TaxCalculationInput object.
Returns	number
Parent object	TaxCalculationOutput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    var deNexus = 7;
    var deTaxCode = 2;
    var deTaxType = 8;
    var deTaxRegistration = 12;

    function calculateTax(context){
        ...
        var output = context.ouput;
        output.overrideNexus({
          nexus: deNexus
        });
        ...
```

```
        if (output.taxRegistration == deTaxRegistration )
        {
            taxCode = deTaxCode;
            taxType = deTaxType;
        }
        ...
    }

    return {
        calculateTax: calculateTax
    }
});
```

taxSummaryLines

Property Declaration	`TaxCalculationOutputSummaryLine[] taxSummaryLines`
Type	Transaction-level object property
Description	Returns an array of TaxCalculationOutputSummaryLine objects that you added to the interface output object with setTaxSummaryLine(options). Use this property to access all tax summary details added to the interface output object for all line items in the transaction. This property returns an empty array if no `TaxCalculationOutputSummaryLine` objects exist.
Returns	TaxCalculationOutputSummaryLine[]
Parent object	TaxCalculationOutput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {

    function calculateTax(context){
        ...
        var taxItemSummaryLines = context.output.getTaxSummaryLines();
        var totalTax = 0;
        for (
          var summaryLineIndex = 0;
          summaryLineIndex < taxItemSummaryLines.length;
          summaryLineIndex++
        )
        {
           // access tax summary
           totalTax += taxItemSummaryLines[summaryLineIndex].taxTotal;
        }
        ...
    }
```

```
    return {
        calculateTax: calculateTax
    }
});
```

overrideNexus(options)

Function Declaration	`void overrideNexus(options)`
Type	Transaction-level object method
Description	Overrides the nexus and sets the corresponding tax registration on the transaction. This method enables the tax engine to override the original nexus value determined by NetSuite based on the nexus lookup logic. Only nexuses associated with the plug-in implementation can be used to override the transaction nexus using this method. After overriding the nexus using this method, you can use the nexus or taxRegistration property in the TaxCalculationOutput interface output object to access the final nexus or tax registration for the transaction. The tax engine cannot override the nexus if the **Override** box is checked on the **Tax Details** subtab of a transaction.
Returns	void
Input Parameters	`Number options.nexus`— Internal NetSuite ID for the new nexus.
Parent object	TaxCalculationOutput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    var deNexus = 7;

    function calculateTax(context){
        ...
        var input = context.input;
        var isDeBillingCountry = input.billToAddress != null && input.billToAddress.country == 'DE';
        if (!input.isTaxRegistrationOverridden && isDeBillingCountry && input.nexus != deNexus)
        {
            context.notifications.addNotice({message: 'Billing address is located in Germany. Changing nexus to
 the same country.'});
            output.overrideNexus({nexus: deNexus});
        }
        ...
    }

    return {
        calculateTax: calculateTax
    }
});
```

setTaxSummaryLine(options)

Function Declaration	`void setTaxSummaryLine(options)`
Type	Transaction-level object method
Description	Sets the tax summary line for a transaction. Each transaction for which you charge taxes contains a summary of the taxes charged for the transaction. This summary is the total taxes on a transaction for a specific tax code and tax type. To use this method, first calculate the tax amount for all line items that need to be charged for a specific tax code and tax type. Then use this method to add the summary to the TaxCalculationOutput interface output object. The summary total appears in the summary panel for the transaction in NetSuite. You must map the tax codes and tax types that you use in the plug-in implementation to the existing tax types in NetSuite. For example, if a tax type in NetSuite like VAT contains an internal NetSuite ID of 123, you must use 123 when you add the tax total for VAT to the transaction. For more information about tax types in NetSuite, see the help topic Tax Types Overview. This requirement also applies when you add tax detail to a specific line item with addTaxDetail(options). If the tax code or tax type does not exist in NetSuite, this method throws an error.
Returns	void
Input Parameters	▪ `options.taxCode` — Internal NetSuite ID for the tax code that want to add to a transaction summary. ▪ `options.taxType` — Internal NetSuite ID for the type of tax you want to add to a transaction summary. ▪ `options.taxAmount` — Total tax amount for a specific tax type on a transaction.
Parent object	TaxCalculationOutput

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
        ...
        // Tax Engine private function to calculate VAT taxes
        var vatTaxes = getVATTaxes(context.input.lines);
        var vatTaxType= 123;
        var thisTaxCode = 456;
        context.output.setTaxSummaryLine({
          taxCode: taxCode,
          taxationType: taxationType,
          taxAmount: taxAmount
        });
        ...
    }

    return {
        calculateTax: calculateTax
```

```
    }
});
```

TaxCalculationOutputLine

Type	Object
Description	Represents all tax details associated with a specific line item in a transaction. Each line item on the transaction for which you want to charge taxes must have an associated `TaxCalculationOutputLine` object. Use createLine(options) to create the `TaxCalculationOutputLine` object and then use addTaxDetail(options) to add the details for each applicable tax type and tax code to the `TaxCalculationOutputLine` object. The tax details for each tax type are represented as a TaxCalculationOutputLineDetail object. For example, if both GST/HST and PST tax apply to a transaction, each `TaxCalculationOutputLine` object requires two `TaxCalculationOutputLineDetail` objects, one for each tax type. After you add tax details for each applicable tax type, use addLine(options) to add the `TaxCalculationOutputLine` object to the TaxCalculationOutput interface output object.
Methods and Properties	■ addTaxDetail(options) ■ taxDetails ■ inputLineReference.
Parent Object(s)	TaxCalculationOutput
Child Object(s)	TaxCalculationOutputLineDetail

addTaxDetail(options)

Function Declaration	`TaxCalculationOutputLineDetail addTaxDetail(options)`
Type	Line item-level object method
Description	Adds a TaxCalculationOutputLineDetail object to a parent TaxCalculationOutputLine object. Use createLine(options) to create the `TaxCalculationOutputLine` object and then use this method to add the details for each applicable tax type and tax code to the `TaxCalculationOutputLine` object. You must map the tax codes and tax types that you use in the plug-in implementation to the existing tax codes and tax types in NetSuite. For example, if a tax type in NetSuite like VAT contains an internal NetSuite ID of 123, you must use 123 when you add the tax total for VAT to the line item. To use this method, you must first calculate the tax amount for each applicable tax type and for each applicable tax code. You must call this method for each relationship between tax types and tax codes. For example, if a line item requires two tax types, and each tax has two applicable tax codes, you must call this method four times, one for each combination of tax type and tax code. This requirement also applies for the tax type when you add a tax summary line to a transaction with setTaxSummaryLine(options). If the tax type or tax code does not exist in NetSuite, this method throws an error.
Returns	TaxCalculationOutputLineDetail
Input Parameters	■ `Number options.taxCode` — Internal NetSuite ID for the tax code for which you want to add tax details to a line item in a transaction.

- `Number options.taxationType` — Internal NetSuite ID for the type of tax you want to add tax details to a line item in a transaction.
- `String options.taxRate` — Tax rate for the tax code of the line item. This rate is specified by the plug-in implementation and does not reflect the values in the NetSuite UI.
- `String options.taxAmount` — Tax amount as a string.
- `String options.taxBasis` — Amount of the transaction line item for which tax was calculated by the plug-in implementation.
- `String options.taxCalculationDetail` — String value that provides information to the NetSuite user on how the plug-in implementation calculated the tax. This is a free-form string that can contain any string value.

Parent object TaxCalculationOutputLine

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
        var inputLines = input.lines;

        for (
          var inputLineIndex = 0;
          inputLineIndex < inputLines.length;
          inputLineIndex++
        )
        {
            // create the tax line
            outputLine = context.output.createLine({lineReference: inputLines[inputLineIndex].reference});
            // get the line amount
            amount = inputLines[inputLineIndex].amount;
            // Calculate the tax

            ...

            // add the tax detail to the taxes for the line item
            outputLine.addTaxDetail({
                taxCode: taxCode,
                taxationType: taxType,
                taxRate: taxRate,
                taxAmount: taxAmount,
                taxBasis: taxBasis,
                taxCalculationDetail: description
            });

            // add the TaxCalculationOutputLine tax detail to the output object
            output.addLine({outputLine: outputLine});
            ...
        }
    }

    return {
        calculateTax: calculateTax
```

```
    }
});
```

taxDetails

Property Declaration	`TaxCalculationOutputLineDetail[] taxDetails`
Type	Line item-level object property
Description	Returns the array of TaxCalculationOutputLineDetail objects added to an TaxCalculationOutputLine object with addTaxDetail(options). Use this property to get the array of `TaxCalculationOutputLineDetail` objects and define plug-in implementation functionality based on the `TaxCalculationOutputLineDetail` objects added to the TaxCalculationOutputLine. This property returns an empty array if no `TaxCalculationOutputLineDetail` objects are added.
Returns	TaxCalculationOutputLineDetail[]
Input Parameters	None
Parent object	TaxCalculationOutputLine

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
        ...
        var outputLines = context.output.lines;

        for (
          var outputLineIndex = 0;
          outputLineIndex < outputLines.length;
          outputLineIndex++
        )
    {
            // get the current tax item line
            var currTaxLine = outputLines[outputLineIndex];
            var taxDetails = currTaxLine.taxDetails;
            for (var taxDetailIndex = 0; taxDetailIndex < taxDetails.length; taxDetailIndex++)
            {
                var taxDetailObj = taxDetails[taxDetailIndex];
                // Process one tax detail line here
                ...
           }
           ...
        }
    }

    return {
```

```
        calculateTax: calculateTax
    }
});
```

inputLineReference

Property Declaration	`TaxCalculationInputLineReference inputLineReference`
Type	Object method
Description	Returns a TaxCalculationInputLineReference object for a TaxCalculationOutputLine object. The returned `TaxCalculationInputLineReference` object is a reference to the transaction input line for which you created an output tax line with createLine(options).
Returns	TaxCalculationInputLineReference
Parent object	TaxCalculationOutputLine

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context){
        var taxItemLines = context.output.lines;

        for (
          var itemLineTaxIndex = 0;
          itemLineTaxIndex < taxItemLines.length;
          itemLineTaxIndex++
        )
        {
            // access all tax details added to transaction
            var currentLineRef = taxItemLines[itemLineTaxIndex].inputLineReference.lineKey;
            ...
        }

  }

    return {
        calculateTax: calculateTax
    }
});
```

TaxCalculationOutputLineDetail

Type	Object
Description	Object that contains the details for a tax calculation for one tax type and one tax code. Create this object and add it to TaxCalculationOutputLine with addTaxDetail(options).

You can get all `TaxCalculationOutputLineDetail` objects added for a specific line item in a transaction to TaxCalculationOutputLine `TaxCalculationOutputLine` with taxDetails.

Parent Object(s) TaxCalculationOutputLine

Child Object(s) n/a

The following table describes the properties you use to access `TaxCalculationOutputLineDetail` object properties:

Property	Return Value	Description
String taxAmount	string	Amount of tax for the tax type and tax code.
Number taxationType	number	Internal NetSuite ID for the type of tax associated with the tax details.
String taxBasis	string	Amount of the transaction line item to which tax was applied by the plug-in implementation.
String taxCalculationDetail	string	String value that provides information to the NetSuite user on how the plug-in implementation calculated the tax. This is a free-form string that can contain any string value.
Number taxCode	number	Internal NetSuite ID for the tax code associated with the tax details.
String taxRate	string	Tax rate for the tax code of the line item. This rate is specified by the plug-in implementation and does not reflect the values in the NetSuite UI.

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context) {
        ...
        var outputLines = context.output.lines;

        for (var outputLineIndex = 0; outputLineIndex < outputLines.length; outputLineIndex++) {
            // get the current tax item line
            var currTaxLine = outputLines[outputLineIndex];
            var taxDetails = currTaxLine.taxDetails;
            for (var taxDetailIndex = 0; taxDetailIndex < taxDetails.length; taxDetailIndex++) {
                var taxDetail = taxDetails[taxDetailIndex];
                var thisTaxType = taxDetail.taxType;
                var thisTaxCode = taxDetail.taxCode;
                var thisTaxAmount = taxDetail.taxAmount;
                var thisTaxBasis = taxDetail.taxBasis;
                var thisTaxRate = taxDetail.taxRate;
                var thisTaxDetail = taxDetail.taxCalculationDetail;

                // process one tax detail line here
                ...
            }
        }
    }
```

```
    return {
        calculateTax: calculateTax
    }
}
```

TaxCalculationOutputSummaryLine

Type	Object
Description	Contains the total of taxes on a transaction for a specific tax type. The values for this object are passed back to NetSuite by the plug-in implementation and appear in the summary panel for a transaction. This object has the following properties that you can access with the `TaxCalculationOutputSummaryLine` properties: ■ Tax code. Internal NetSuite ID for the tax code. ■ Tax type. Internal NetSuite ID for the type of tax. ■ Tax amount. Total amount of the tax type calculated for the transaction. After you calculate the total taxes for a tax type for a transaction, use setTaxSummaryLine(options) to create this object. Use taxSummaryLines to get an array that represents all the `TaxCalculationOutputSummaryLine` objects created with `setTaxSummaryLine(options)`.
Methods	■ taxType ■ TaxTotal
Parent Object(s)	TaxCalculationOutput
Child Object(s)	n/a

The following table describes the methods you use to access `TaxCalculationOutputSummaryLine` object properties:

Property	Return Value	Description
Number taxationType	number	Internal NetSuite ID for the type of tax.
Number taxCode	number	Internal NetSuite ID for the tax code.
String taxTotal	string	Amount of tax for the tax code and tax type.

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context) {
        ...
        var output = context.output;

        // get summary of all tax types and amounts applicable to this transaction
        var allSummaryLines = out.taxSummaryLine;
```

```
        for (var summaryIndex = 0; summaryIndex < allSummaryLines.length; summaryIndex++)
        {
            var thisTaxSummary = allSummaryLines[summaryIndex];

            var thisTaxCode = thisTaxSummary.taxCode;
            var thisTaxType = thisTaxSummary.type;
            var thisTaxAmount = thisTaxSummary.taxAmount;
            ...
        }
    }

    return {
        calculateTax: calculateTax
    }
}
```

TaxCalculationNotificationList

Type	Interface object
Description	Object that contains methods to add notifications to a transaction during tax calculation by a SuiteTax plug-in implementation. NetSuite displays the notifications at the top of the NetSuite window when the plug-in implementation calls one of the notification methods. Each of the methods takes a string as an input parameter that you can use to display information to NetSuite users.
Methods	See TaxCalculationNotificationList Property or Methods.
Parent Object(s)	n/a
Child Object(s)	n/a

TaxCalculationNotificationList Property or Methods

The following table lists the methods available to the `TaxCalculationNotificationList` object:

Property or Method Declaration	Description
`String locale`	Returns the locale of the current browser window as a string, for example, `en_US`. Use this method to display notifications specific to the browser locale.
`void addError(options)`	Displays the error message string in the browser. In addition, using this method returns the user to the transaction in edit mode, to allow the user to fix the error. You cannot save the transaction until the error is corrected.
`void addWarning(options)`	Displays the warning message string in the browser. The plug-in implementation completes the tax calculation and NetSuite saves the transaction. Use this method to display potential issues or non-fatal errors to the NetSuite users.
`void addNotice(options)`	Displays the notice message string in the browser. The plug-in implementation completes the tax calculation and NetSuite saves the transaction.

Property or Method Declaration	Description
	Use this method to display informative information to the NetSuite user.
`String options.message`	Used with the methods in this table to add a text message to the error.

Note: There is no functional difference between `addWarning(options)` and `addNotice(options)`. Each message appears with a different label in the browser to differentiate between the two types.

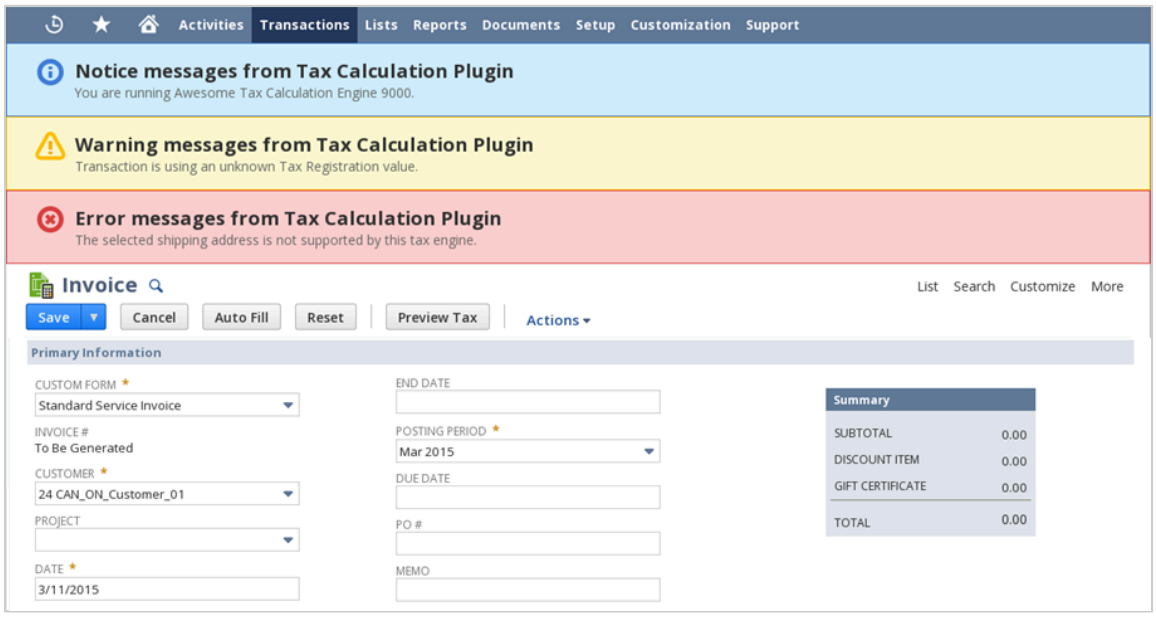
TaxCalculationNotificationList Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['R/runtime'], function(runtime) {
    function calculateTax(context) {
        ...
        var account = runtime.accountId;
        ...
        context.notifications.addNotice(account + 'running Awesome Tax Calculation Engine 9000.');
        ...
    }

    return {
        calculateTax: calculateTax,
        onTransactionEvent: onTransactionEvent
    }
}
```

TaxCalculationNotificationList User Interface Example

The following screenshot shows how notifications appear to users during the execution of a SuiteTax plug-in implementation:



defineAdditionalFields(context)

Function Declaration	`void defineAdditionalFields(context)`
Type	Interface function
Description	Defines the additional fields on a transaction that NetSuite passes to the plug-in implementation. The fields can be standard NetSuite fields or custom fields added to the record form for the transaction type. Use this function to pass additional field values to the plug-in implementation that are not supported by the methods available to the TaxCalculationInput interface input object. Use the field values to define additional plug-in implementation functionality for tax calculation on the transaction. For example, you can add a custom field named **Memo** to a sales order record form. Then, use the context.addField(options) addSublistField method to specify that NetSuite passes the value of the **Memo** custom field to the plug-in implementation. After you define the field names with this function, use getAdditionalFieldValue(options) in the calculateTax(context) interface function to get the field value. You cannot use this function to define any fields that do not exist on the form for the transaction type. You can also use this function to pass additional line-item field values to the plug-in implementation that are not supported by the methods available to the TaxCalculationInputLine object. For example, you can add a custom field named Price to an item record form. Then, use the context.addSublistField(options) method to specify that NetSuite passes the value of the Price custom field to the plug-in implementation. After you define the field names with this function, use TaxCalculationInputLine.getAdditionalFieldValue(options) in the calculateTax(context) interface function to get the field value. **Note:** Updating the value of the fields declared in this function resets the values on the Tax Details subtab of the transaction.
Returns	void
Parameters	AdditionalFieldsContext

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context) {
        ...
    }

    function defineAdditionalFields(context) {
        context.addField({fieldId: 'memo'});
        context.addSublistField({
          sublistId: 'item',
          fieldId: 'description'
```

```
        });
    }

    return {
        calculateTax: calculateTax,
        defineAdditionalFields: defineAdditionalFields
    }
}
```

AdditionalFieldsContext

Type	Parameter object
Description	With this object, you can call the `addField` and `addSublist` fields to request additional parameters.

addField

Function Declaration	`context.addField(options)`
Type	Parameter object method
Description	Adds a transaction-level field to the list of additional fields requested by the tax calculation plugin.
Input Parameters	object options string `options.fieldID` — ID of the field to add

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context) {
        ...
    }

    function defineAdditionalFields(context) {
        context.addField({fieldId: 'memo'});
        context.addSublistField({
          sublistId: 'item',
          fieldId: 'description'
        });
    }

    return {
        calculateTax: calculateTax,
        defineAdditionalFields: defineAdditionalFields
    }
});
```

addSublistField

Function Declaration	`context.addSublistField(options)`
Type	Parameter object method
Description	Adds a line-level field to the list of additional line fields requested by the tax calculation plugin.
Returns	void
Input Parameters	object options string `options.fieldID` — ID of the sublist field to add string `options.sublistID`— ID of the sublist containing the field

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define([], function() {
    function calculateTax(context) {
        ...
    }

    function defineAdditionalFields(context) {
        context.addField({fieldId: 'memo'});
        context.addSublistField({
          sublistId: 'item',
          fieldId: 'description'
        });
    }

    return {
        calculateTax: calculateTax,
        defineAdditionalFields: defineAdditionalFields
    }
});
```

onTransactionEvent(context)

Property Declaration	`void onTransactionEvent(context)`
Type	Interface function
Description	Defines the functionality for a SuiteTax plug-in implementation when specific events occur on the transaction record. You can define plug-in functionality for when a user voids or deletes a transaction. This function is called whenever a user performs one of the supported event actions on the record type for the SuiteTax plug-in implementation. The plug-in implementation executes the logic in the function after the system performs the void or delete action. The purpose of this

function is informational only. Potential errors thrown within this function do not affect the voiding or deletion in any way.

String literals on the TransactionEventCode object (accessible through TransactionEvent) define the supported events.

Use the methods available to the TaxCalculationInput object to get information about the transaction and use TransactionEvent to get the type of event that occurred.

> **Note:** This function is not called when a user clicks **Preview Tax** on a supported taxable transaction. In addition, while the function must be declared in the plug-in implementation script file, it is optional to include any logic for it.

Returns	void
Parameters	`TaxCalculationinput context.input` `TransactionEvent context.event`

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context) {
        ...
    }

    function onTransactionEvent(context) {
        var input = context.input;
        var entity = record.load({type: input.entityType, id: input.entity});
        ...
        entity.submit();
    }

    return {
        calculateTax: calculateTax,
        onTransactionEvent: onTransactionEvent
    }
});
TransactionEvent
```

TransactionEvent

Type	Interface object
Description	Contains the property to access the event code for when a supported taxable transaction is voided or deleted.
Properties	■ code
Parent Object(s)	n/a
Child Object(s)	TransactionEventCode

code

Function Declaration	`TransactionEventCode code`
Type	Object property
Description	Returns the TransactionEventCode object. Use this property to determine if a transaction was voided or deleted.
Returns	TransactionEventCode
Parent object	TransactionEvent

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context) {
        ...
    }

    function onTransactionEvent(context) {
        ...
        var event = context.event.code;

        if (event == 'void') // transaction was voided
        {
            ...
        }
        else if (event == 'delete') // transaction was deleted
        {
              ...
        }
        ...
    }

    return {
        calculateTax: calculateTax,
        onTransactionEvent: onTransactionEvent
    }
});
```

TransactionEventCode

Type	Object
Description	Object that represents the event type of a TransactionEvent object. This object has two enumerated values: ■ "DELETE" ■ "VOID"

	Use this value to determine the type of event that occurred to trigger the onTransactionEvent(context) interface function. The interface function is called on all of the event types and executed before the NetSuite functionality that triggered the function.
Methods	None
Parent Object(s)	TransactionEvent
Child Object(s)	n/a

Example

```
/**
  * @NApiVersion 2.0
  * @NScriptType taxCalculationPlugin
  */
define(['N/record'], function(record) {
    function calculateTax(context) {
        ...
    }

    function onTransactionEvent(context) {
        ...
        var event = context.event.code;

        if (event == 'void') // transaction was voided
        {
            ...
        }
        else if (event == 'delete') // transaction was deleted
        {
            ...
        }
        ...
    }

    return {
        calculateTax: calculateTax,
        onTransactionEvent: onTransactionEvent
    }
});
```

SuiteTax Plug-in Reference

Use the information in this section as reference when you develop, run, and test a SuiteTax plug-in implementation.

SuiteTax Plug-in Guidelines and Best Practices

Use the best practices in the following table when you develop a plug-in implementation for the SuiteTax plug-in:

Guideline/Best Practice	Description
Group custom fields	If the SuiteTax solution requires custom fields on NetSuite records to collect information relevant to tax calculation, make sure you include the fields where they are most useful. For example, you can create a subtab specifically to group custom fields relevant to tax calculation, instead of on the main tab for the record. In general, make sure custom fields do not appear significant to users for whom they are not applicable.
Centralize tax engine configuration	If your SuiteTax solution makes use of, for example, custom records to track configurations for the tax engine, group those configurations together on a single page in NetSuite. Grouping configurations in this manner increases usability; administrators can complete all necessary configurations in a single location.
Minimize plug-in execution time	You should optimize the performance of the plug-in implementation to return tax calculation results within 10 seconds. This includes the amount of time required for the plug-in implementation to contact any third-party site that returns tax calculation information back to NetSuite. A longer period of time negatively impacts user experience with the plug-in implementation performance.
Include tax configuration in plug-in implementation bundle	Any dependent records or objects required by the implementation should be included in the bundle, or created by a bundle installation script file. You should not require administrators to perform manual configuration after installation of the plug-in implementation bundle. For more information about installation script files, see the help topic Bundle Installation Scripts.
Limit creation of custom records by plug-in implementation	If possible, you should not create records or perform other complex data processing operations in the plug-in implementation script file. In addition, where possible, you should limit the use of SuiteScript APIs that load records, to improve performance of the plug-in implementation. In general, searching for NetSuite records yields better performance than loading records.
Use TaxCalculationNotificationList	You should use the TaxCalculationNotificationList object and its related methods to display the following information to the user:

Guideline/Best Practice	Description
	■ Errors or warnings produced by the plug-in implementation ■ Information about how tax was calculated **Note:** You can use the locale property to display messages that match the user locale.
Log plug-in execution to the plug-in implementation	Use the N/log Module to send details to the Execution Log tab of the plug-in implementation object.
Properly display provider-specific information	If you want to display logos or custom fields, make sure that the associated plug-in functionality only displays the custom information on the correct records. For example, if the plug-in does not calculate tax for specific types of transactions, make sure that any information displayed by the plug-in implementation does not appear on those transactions.
Persist only final tax details	You can pass transaction details to a third-party tax system for tax calculation purposes. However, you should not save tax details generated from the following events: ■ The **Preview Taxes** button is clicked on a transaction ■ Tax details are from unapproved or non-posting transactions You can use the postingTransaction or preview methods to determine if the tax details should be saved on the remote system.
Provide additional information about the tax calculation on the **Details** column on the **Tax Details** subtab for the transaction	The addTaxDetail(options) method includes the taxCalculationDetail parameter. You can use this parameter to pass a free-form string value back to the transaction. This string value appears on the **Tax Details** subtab for the transaction, in the **Details** column.
Rounding of tax details and summary amounts	Make sure that the tax calculations performed by the SuiteTax plug-in implementation round tax results to two decimal places. If a tax value is greater than two decimal places, NetSuite truncates the amount to two, but does not round.
Create custom role for **Execute as Role** plug-in implementation property	The **Execute as Role** property determines the role that the plug-in implementation executes on. You should create a custom role that has the appropriate level of permissions needed to function correctly. **Important:** Either do not select any subsidiaries in the **Subsidiaries** field for the Role record, or select all subsidiaries where the tax engine that represents the plug-in implementation can be selected.

Values on an Address Object

The Address object is returned by the billFromAddress, billToAddress, shipFromAddress, and shipToAddress methods on the TaxCalculationInput object, and by the shipFromAddress and shipToAddress methods on a in a SuiteTax plug-in implementation.

The source for an Address object values is different for sales transactions and purchase transactions. See the following tables for details:

Address Type	Sales Transactions Address Sourcing
Bill From	Subsidiary record ■ Main address
Bill To	Customer record ■ Default billing address is preselected. ■ User can override this address.
Ship From — Transaction-Level	Ship From address sourcing is per transaction: ■ If a location is entered for a transaction: □ If the location has an address, use this address. □ If the location does not have an address, return null. ■ If a location is not entered for a transaction, source the address from the subsidiary: □ If the subsidiary has a shipping address, use this address. □ If the subsidiary does not have a shipping address, use its main address.
Ship From — Line Level	Ship From address sourcing is per line item: ■ If a location is entered for a line item: □ If the location has an address, use this address. □ If the location does not have an address, return null. ■ If a location is not entered for a line item, use the transaction-level Ship From address. ⚠ **Important:** Line-level locations override transaction-level address information.
Ship To — Transaction Level	If **Enable Line Item Shipping** is not checked, transaction-level address is sourced from the customer record. ■ Default shipping address is preselected. ■ User can override this address. Otherwise, null is returned.
Ship To — Line Level	If **Enable Line Item Shipping** is checked, line-level addresses are sourced from the customer record. ■ Default shipping address is preselected. ■ User can override this address on each line. Otherwise, null is returned.

Address Type	Purchase Transactions Address Sourcing
Bill From	Vendor record ■ Default billing address is preselected. ■ User can override this address.
Bill To	Subsidiary record

	▪ Main address
Ship From	Vendor record ▪ Default shipping address is preselected. ▪ User can override this address. **Note:** TaxCalculationInputLine.shipFromAddress always returns null.
Ship To — Transaction-Level	Ship To address sourcing is per transaction: ▪ If a user has entered a Ship To Select address for the transaction, this address is used. ▫ For drop-ship transactions, the customer's address may be available for selection as a Ship To address. ▪ If a user has not entered a Ship To Select address for the transaction, and: ▫ If a location is entered for a transaction: ▪ If the location has an address, use this address. ▪ If the location does not have an address, return null. ▫ If a location is not entered for a transaction, source the address from the subsidiary: ▪ If the subsidiary has a shipping address, use this address. ▪ If the subsidiary does not have a shipping address, use its main address.
Ship To — Line-Level	Ship To address sourcing is per line item: ▪ If a user has entered both a Ship To customer and a Ship To Select address for the transaction, that address is used. ▪ If a user has not entered a Ship To customer or a Ship To Select address for the transaction: ▫ If a location is entered for a line item: ▪ If the location has an address, use this address. ▪ If the location does not have an address, return null. ▫ If a location is not entered for a line item, use the transaction-level Ship To address. **Important:** Line-level locations override transaction-level address information, except in the case of drop-ship, where line-level locations are ignored.