Financial Institution Connectivity Plug-in
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

Financial Institution Connectivity Plug-in Interface Overview .......................................................... 1
Benefits of Using the Financial Institution Connectivity Plug-in ...................................................... 3
Limitations of Using the Financial Institution Connectivity Plug-in .................................................. 4
Developing a Financial Institution Connectivity Plug-in ................................................................. 4
  Enabling Features for a Financial Institution Connectivity Plug-in ............................................ 4
  Creating the Financial Institution Connectivity Plug-in Script File ............................................ 4
Administering a Financial Institution Connectivity Plug-in ........................................................... 7
  Enabling Features for a Financial Institution Connectivity Plug-in ............................................ 7
  Adding the Financial Institution Connectivity Plug-in Implementation ..................................... 7
Configuration for the Financial Institution Connectivity Plug-in ................................................. 8
Testing the Financial Institution Connectivity Plug-in ................................................................. 8
Running the Financial Institution Connectivity Plug-in ............................................................... 9
Financial Institution Connectivity Plug-in Interface Definition ....................................................... 10
  getConfigurationIFrameUrl(context) ..................................................................................... 11
    pluginConfiguration ........................................................................................................ 12
    configurationIFrameUrl ................................................................. 13
  getAccounts(context) ......................................................................................................... 13
    pluginConfiguration ........................................................................................................ 13
    addAccount ................................................................................................................ 14
  getTransactionData(context) ............................................................................................... 15
    pluginConfiguration ........................................................................................................ 16
    accountRequestsJSON .................................................................................................. 16
    addDataChunk ........................................................................................................... 16
      returnAccountRequestsJSON .................................................................................... 16
Error Propagation .................................................................................................................... 17
Financial Institution Connectivity Plug-in Interface Overview

**Important:** The Financial Institution Connectivity Plug-in uses SuiteScript 2.0. Your entry point scripts and supporting library scripts must also use SuiteScript 2.0. For information, see the help topics SuiteScript 2.0 Script Creation Process and SuiteScript 2.0.

To create a Financial Institution Connectivity Plug-in, you need to implement the Financial Institution Connectivity API as a SuiteScript plug-in interface. For details, see Financial Institution Connectivity Plug-in Interface Definition. You can use the Financial Institution Connectivity Plug-in to directly connect to financial institutions, for inbound transmissions, through NetSuite. Inbound transmission enables you to automate cash balance reporting and schedule account statement downloads. Unlike NetSuite 2019.2 where manual import was required, the new connectivity plug-in lets you automate the bank data import process.

If your administrator has created a format profile configured to directly connect to a financial institution, NetSuite can automatically import data from your accounts daily. Creation of the format profile invokes the plug-in, which initiates a bank data import and creates a daily import schedule. Subsequently editing and saving the format profile also initiates an import. Updated account data and bank balances appear on the Match Bank Data page.

For details on configuring format profiles, see the help topic Creating Format Profiles. For information on manually updating your bank feeds, see the help topic Updating Imported Bank Data.

After the administrator performs the necessary one-time setup on a format profile record, the accountant or general financial user can continue with other stages of the Cash Management workflow, such as transaction matching and reconciliation. There may be cases where the plug-in requires user credentials for a financial institution. NetSuite itself does not ask for or store these credentials, but it enables the plug-in to save them securely. This may be done by a third-party service or by a customization related to the plug-in.

The following diagram shows the synchronous workflow for the format profile record:
getConfigurationIFrameUrl() and getAccounts() are interface functions of the Financial Institution Connectivity Plug-in interface, and they do the following:

- **getConfigurationIFrameUrl(context)** - Retrieves a URL for configuring a connection to your financial institution. This function is invoked when you do one of the following:
  - Create a new format profile and select a Financial Institution Connectivity Plug-in from the Connectivity Method dropdown list
  - Edit an existing format profile that uses a Financial Institution Connectivity Plug-in

- **getAccounts(context)** - Retrieves all available accounts from a financial institution, so that you can create account mappings to corresponding accounts in NetSuite. This method queries the status of the financial institution accounts, including the date or time since the last update.

The following diagram shows the asynchronous workflow for the bank data import process:

```
import Job
  Financial Institution Connectivity Plug-in
    getAccounts()
    getTransactionData()
      parseBankStatement()
  Parser Plug-in
  Financial Institution
```

getAccounts() and getTransactionData() are interface functions of the Financial Institution Connectivity Plug-in interface.

parseBankStatement() is an interface function of the Parser Plug-in interface.

**Note:** The Financial Institution Connectivity Plug-in works in conjunction with the Parser Plug-in. Therefore, the Financial Institution Connectivity Plug-in supports any format that has a Parser Plug-in.

All three functions are invoked when a bank data import is initiated, and they do the following:

- **getAccounts(context)** - Retrieves accounts from your financial institution, which appear on the Account Mapping subtab on the format profile record. This subtab is populated with information based on the financial institution accounts for which you have a configured connection. You can map your unmapped accounts to corresponding accounts in NetSuite.

  **Note:** For a successful import, you need at least one active account mapping.

- **getTransactionData(context)** - Retrieves your bank data from your financial institution. When invoked for the first time, NetSuite retrieves the last 60 calendar days of data, by default, as requested in the accountRequestsJSON property. If desired, you can modify the date range. For details, see...
Benefits of Using the Financial Institution Connectivity Plug-in

accountRequestsJSON. Subsequently, NetSuite only retrieves any new data created since the last successful import.

- **parseBankStatement** – Consumes the data retrieved by the Financial Institution Connectivity Plug-in and parses it into transactions. To use the connectivity plug-in, you need an existing or new implementation of the Parser Plug-in. To develop a Parser Plug-in, see the help topic Developing a Parser Plug-in.

The Financial Institution Connectivity Plug-in is supported by SuiteCloud Development Framework (SDF). SDF allows you to customize your NetSuite account from an integrated development environment (IDE) on your local computer. For details, see the help topic SuiteCloud Development Framework Overview.

For more information about the Financial Institution Connectivity Plug-in, see the following topics:

<table>
<thead>
<tr>
<th>NetSuite Role</th>
<th>For more information, see ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>All roles</td>
<td>Benefits of Using the Financial Institution Connectivity Plug-in</td>
</tr>
<tr>
<td></td>
<td>Limitations of Using the Financial Institution Connectivity Plug-in</td>
</tr>
<tr>
<td>Developer</td>
<td>Developing a Financial Institution Connectivity Plug-in</td>
</tr>
<tr>
<td>Administrator</td>
<td>Administering a Financial Institution Connectivity Plug-in</td>
</tr>
</tbody>
</table>

In addition to creating your own plug-in, you can use the Bank Feeds Connectivity Plug-in available through the Bank Feeds SuiteApp. This is useful if you want to connect to financial institutions in the United States or Canada but do not want to develop and install your own plug-in. The Bank Feeds SuiteApp is targeted for release in February 2020.

**Note:** This section only covers the development and administration of the Financial Institution Connectivity Plug-in. For details on using a Financial Institution Connectivity Plug-in to import bank data, see the help topic Creating Format Profiles.

Benefits of Using the Financial Institution Connectivity Plug-in

Reasons why you might want to create a connectivity plug-in rather than use the Bank Feeds SuiteApp include the following:

- To own and develop your own specific integration to a financial institution API you have that can be configured to your needs.

  **Note:** The Bank Feeds SuiteApp provides a generic way to connect to a large set of financial institutions.

- To enable your customers to connect to financial institutions that are not supported by the Bank Feeds SuiteApp. This includes any financial institutions outside of the United States and Canada.

  **Note:** The Bank Feeds SuiteApp is only available to connect to financial institutions in the United States or Canada.

- To let customers retrieve bank data from their financial institution without having to provide user credentials to a third-party service, which the Bank Feeds SuiteApp requires.
In addition to creating plug-ins, the Financial Institution Connectivity API enables you to develop your own connectivity applications that provide more options for the way that your system connects to financial institutions.

Limitations of Using the Financial Institution Connectivity Plug-in

When using the Financial Institution Connectivity Plug-in, keep in mind the following limitation:

- The plug-in does not support IP address whitelisting.

**Note:** The Financial Institution Connectivity Plug-in lets you retrieve ZIP files, provided that your Parser Plug-in is capable of handling them.

Developing a Financial Institution Connectivity Plug-in

To develop a Financial Institution Connectivity Plug-in, complete the following steps:

- Enabling Features for a Financial Institution Connectivity Plug-in
- Creating the Financial Institution Connectivity Plug-in Script File

Enabling Features for a Financial Institution Connectivity Plug-in

To begin development of a Financial Institution Connectivity Plug-in, make sure that the Server SuiteScript feature is enabled on the development account.

**To enable features for the plug-in:**

1. Choose Setup > Company > Enable Features.
2. On the SuiteCloud subtab, make sure Server SuiteScript and Client SuiteScript are checked. If necessary, check each box and agree to the Terms of Service.
3. Click Save.

Creating the Financial Institution Connectivity Plug-in Script File

You must implement each Financial Institution Connectivity 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.
The following sample implements a Financial Institution Connectivity Plug-in that returns static data using the 'N/search' SuiteScript module in the plug-in implementation script file:

```
/**
 * @NApiVersion 2.x
 * @NScriptType fiConnectivityPlugin
 * @NModuleScope SameAccount
 */
define(['N/search'],
    function (search) {

        // internal function used to load configuration for this plug-in from a custom record
        function loadConfiguration(configurationId)
        {
            var searchResults = search.create({
                type: 'customrecord_sampleconfig',
                filters: [{
                    name: 'custrecord_configurationid',
                    operator: 'is',
                    values: [configurationId]
                }]
            });
            return searchResults.run().getRange({start: 0, end: 1});
        }

        function getConfigurationIFrameUrl(context)
        {
            var configurationId = context.pluginConfiguration.getConfigurationFieldValue({fieldName: "configuration_id");
            context.configurationIFrameUrl = "/app/site/hosting/scriptlet.nl?script=1&deploy=1&configurationId=" + configurationId;
        }

        function getAccounts(context)
        {
            var configurationId = context.pluginConfiguration.getConfigurationFieldValue({fieldName: "configuration_id");
            var configuration = loadConfiguration(configurationId);
            context.addAccount(
                accountMappingKey: "12345",
                displayName: "Checking (XXXX11)",
                accountType: "BANK",
                currency: "USD",
                groupName: "Bank of America",
                lastUpdated: "2020-06-30T01:23:45"
            );
        }

        function getTransactionData(context)
        {
            var configurationId = context.pluginConfiguration.getConfigurationFieldValue({fieldName: "configuration_id");
        }
    }
```

Note: For details on the 'N/search' SuiteScript module, see the help topic N/search Module.
var configuration = loadConfiguration(configurationId)
var accountRequests = JSON.parse(context.accountRequestsJSON);
if (accountRequests != null) {
    accountRequests.forEach(function (accountRequest) {
        var accountId = accountRequest.accountMappingKey;
        var fromDateTime = accountRequest.dataStartTime;
        var toDateTime = accountRequest.dataEndTime;

        var downloadedData = "... Retrieve the account data using a web service request or a file transfer...";

        context.addDataChunk({dataChunk: downloadedData});
    });
}
context.returnAccountRequestsJSON({accountsJson: context.accountRequestsJSON});

return {
    getConfigurationIFrameUrl: getConfigurationIFrameUrl,
    getAccounts: getAccounts,
    getTransactionData: getTransactionData
}

For details and breakdowns of the sample, see Financial Institution Connectivity Plug-in Interface Definition.

The following table describes the interface functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getConfigurationIFrameUrl(context)</td>
<td>Retrieve a URL for configuring a connection to your financial institution. For most plug-in authors, this is a URL to a Suitelet. This URL appears on the Connectivity Configuration subtab on a format profile record.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> For details on Suitelets, see the help topic SuiteScript 2.0 Suitelet Script Type.</td>
</tr>
<tr>
<td>getAccounts(context)</td>
<td>Retrieve accounts from your financial institution. This function is invoked when you:</td>
</tr>
<tr>
<td></td>
<td>- Select the Account Mapping subtab on a format profile record</td>
</tr>
<tr>
<td></td>
<td>- Import bank data into NetSuite</td>
</tr>
<tr>
<td>getTransactionData(context)</td>
<td>Retrieve data from your financial institution. This function is invoked when the bank data import process is initiated.</td>
</tr>
</tbody>
</table>
### Rules and Guidelines for Creating a Plug-in Implementation Script File

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 to use with the plug-in implementation script file, you can include those files when you create the plug-in implementation in NetSuite. See Adding the Financial Institution Connectivity Plug-in Implementation.
- Due to limitations with how strings are handled in NetSuite, you cannot use the JavaScript `case` statement in the plug-in implementation script file.
- Each interface function allows up to 1000 usage units.

### Administering a Financial Institution Connectivity Plug-in

To install and set up a Financial Institution Connectivity Plug-in, complete the following steps:

- Enabling Features for a Financial Institution Connectivity Plug-in
- Adding the Financial Institution Connectivity Plug-in Implementation
- Configuration for the Financial Institution Connectivity Plug-in
- Testing the Financial Institution Connectivity Plug-in
- Running the Financial Institution Connectivity Plug-in

### Enabling Features for a Financial Institution Connectivity Plug-in

See Enabling Features for a Financial Institution Connectivity Plug-in.

### Adding the Financial Institution Connectivity Plug-in Implementation

After creating the plug-in script file, upload and implement the file in NetSuite.

**To add the plug-in implementation:**

1. In NetSuite, go to Customization > Plug-ins > Plug-in Implementations > New.
NetSuite displays the Upload Plug-in Implementation page.

2. For the Script File, click the plus icon.
3. Click Choose File, then select the JavaScript file.
4. Click Save.
5. Click Create Plug-in Implementation.
6. On the Plug-in Implementation page, enter the following information:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>User-friendly name for the implementation. This name appears in the plug-in implementation list.</td>
</tr>
<tr>
<td>ID</td>
<td>Internal ID for the implementation for use in scripting. If you do not provide an ID, NetSuite</td>
</tr>
<tr>
<td></td>
<td>will provide one for you after you click Save. As a developer, when you write code that uses</td>
</tr>
<tr>
<td></td>
<td>this implementation, you will reference the implementation using this ID.</td>
</tr>
<tr>
<td>Status</td>
<td>Current status for the implementation. Choose Testing to have the implementation</td>
</tr>
<tr>
<td></td>
<td>accessible to the owner of the implementation. Choose Released to have the implementation accessible</td>
</tr>
<tr>
<td></td>
<td>in a production environment</td>
</tr>
<tr>
<td>Log Level</td>
<td>Logging level you want for the script. Select Debug, Audit, Error, or Emergency. The log information</td>
</tr>
<tr>
<td></td>
<td>appears on the Execution Log subtab for the implementation after you create it. Go to Customization</td>
</tr>
<tr>
<td></td>
<td>&gt; Plug-ins &gt; Plug-in Implementations, select your implementation, and then click the Execution Log</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description of the implementation.</td>
</tr>
<tr>
<td>Owner</td>
<td>User account that owns the implementation. Default is the name of the current user</td>
</tr>
</tbody>
</table>

7. On the Unhandled Errors subtab, select which individual(s) will be notified if script errors occur. By default the Notify Script Owner box is checked.

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

8. Click Save.

Configuration for the Financial Institution Connectivity Plug-in

All configuration for the Financial Institution Connectivity Plug-in is done on the Format Profile page. The configuration steps required depend on the plug-in's initial implementation.

For details on configuring connectivity to a financial institution, see the help topic Financial Institution Connectivity Configuration.

Testing the Financial Institution Connectivity Plug-in

To test the Financial Institution Connectivity Plug-in, run the plug-in to import bank data and then view your import.
To run the plug-in, see Running the Financial Institution Connectivity Plug-in.

If you receive an import error, look at the plug-in's execution log for details. Go to Customization > Plug-ins > Plug-in Implementations. Click the View link of the Financial Institution Connectivity Plug-in, and then on the Plug-in Implementation page, click the Execution Log subtab.

To view your imported data:

- Go to Transactions > Bank > Import Online Banking Data > List. This page enables you to track your import status and view the details of your imported bank data. For details, see the help topic Viewing Imported Account Statements.

- Go to Transactions > Bank > Match Bank Data. To view your current bank balance and imported bank data for an account, select an account from the Account dropdown list. The Imported Bank Data grid displays any new data imported since the last successful import. For details, see the help topic Matching Bank Data.

When you run the plug-in for the first time, not only is an import initiated, but a daily import schedule is created. To verify that you are receiving new data each day, you can check the Imported Statement List or Match Bank Data page daily.

### Running the Financial Institution Connectivity Plug-in

To run the Financial Institution Connectivity Plug-in, you must configure and save a format profile that uses the plug-in to directly connect you to a financial institution. For details, see the help topic Creating Format Profiles. To use the plug-in available through the Bank Feeds SuiteApp, see the help topic “Bank Feeds SuiteApp”.

> **Note:** To perform a successful import, you need at least one active account mapping on the format profile record.

When you save the format profile for the first time, NetSuite automatically downloads the last 60 days of data from all financial institution accounts mapped. NetSuite also creates a daily import schedule. For more information, see the help topic Automatic Bank Data Import.

After the plug-in is run for the first time, you can run subsequent imports in the following ways:

- Edit and save an existing format profile configured for direct connectivity (see the help topic Creating Format Profiles). NetSuite imports any new data from your existing accounts since the last successful import. For any new accounts, NetSuite imports the last 60 days of data.

  > **Note:** This is the primary method to run and test your plug-in.

- Initiate a manual update of data for an account on the Match Bank Data page by clicking the Update Imported Bank Data button. NetSuite imports any new data from the account since the last successful import, which appears in the Imported Bank Data grid. For details, see the help topic Updating Imported Bank Data

  > **Note:** You should only use this method in the following scenarios:

  - You have a need to test the plug-in for a single financial institution account.
  - The selected account does not contain the latest data.
  - There was an import error.

To verify that the plug-in is working, see Testing the Financial Institution Connectivity Plug-in.
Financial Institution Connectivity Plug-in Interface Definition

The plug-in interface includes the following functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| `getConfigurationIFrameUrl(context)` | Retrieve a URL for configuring a connection to your financial institution. For most plug-in authors, this is a URL to a Suitelet that appears on the Connectivity Configuration subtab on the Format Profile page. This function is invoked when you:
  - Create a new format profile record and select a Financial Institution Connectivity Plug-in
  - Edit an existing format profile record configured with the Financial Institution Connectivity Plug-in |
| `getAccounts(context)` | Get accounts from your financial institution. This function is invoked when you:
  - Select the Account Mapping subtab on the Format Profile page
  - Import bank data into NetSuite |
| `getTransactionData(context)` | Get account data from your financial institution. This function is invoked when the bank data import process is started. |

**Note:** To import successfully, you require at least one active account mapping on the format profile record.

The plug-in interface includes one interface object called `pluginConfiguration`, which retrieves user-supplied configuration data (field values) for this plug-in.

All three interface functions listed above use the `pluginConfiguration` object:

- For `getConfigurationIFrameUrl(context)`, the object retrieves the field values needed to return the proper URL.
- For `getAccounts(context)`, the object retrieves the field values needed to obtain the list of accounts from a financial institution.
- For `getTransactionData(context)`, the object retrieves the field values needed to retrieve data from your financial institution.

The `pluginConfiguration` context object includes the following functions and properties:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getConfigurationFieldValue</code></td>
<td>Retrieves a named configuration value. For all Financial Institution Connectivity Plug-ins that you develop, the following value is available:</td>
</tr>
</tbody>
</table>
### Function Description

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>configuration_id</strong></td>
<td>A format profile-level unique identifier which can be used to refer to a format profile, even if the record is not created. The <code>pluginConfiguration</code> context object uses this object function. This function belongs to the following interface functions:</td>
</tr>
<tr>
<td><strong>getConfigurationIFrameUrl(context)</strong></td>
<td>The URL to a user interface for configuring a connection to your financial institution. This URL appears on the Connectivity Configuration subtab on the Format Profile page. The <code>getConfigurationIFrameUrl(context)</code> interface function uses this property.</td>
</tr>
<tr>
<td><strong>addAccount</strong></td>
<td>Adds a financial institution account that you want to map to a corresponding NetSuite account. The list of accounts you are mapping appears on the Account Mapping subtab on the Format Profile page. The <code>getAccounts(context)</code> interface function uses this object function.</td>
</tr>
<tr>
<td><strong>accountRequestsJSON</strong></td>
<td>Contains the list of accounts required for the plug-in to query bank data, as well as the required data date ranges. The information retrieved is provided as a JSON format string. The <code>getTransactionData(context)</code> interface function uses this property.</td>
</tr>
<tr>
<td><strong>addDataChunk</strong></td>
<td>Transmits the financial institution's data file to NetSuite as a series of chunks. <code>addDataChunk</code> encrypts the incoming chunks before storing them in the database. Each chunk size has a 25-million character limit. The <code>getTransactionData(context)</code> interface function uses this object function.</td>
</tr>
<tr>
<td><strong>returnAccountRequestsJSON</strong></td>
<td>Returns the list of accounts queried by the plug-in, as well as the reason for any query failures. If an account is provided by <code>accountRequestsJSON</code> but not returned by the Financial Institution Connectivity API, it is assumed that there is no account-specific failure. The information returned must be in a JSON format string. The <code>getTransactionData(context)</code> interface function uses this object function.</td>
</tr>
</tbody>
</table>

---

**getConfigurationIFrameUrl(context)**

**Function Declaration**

```javascript
function getConfigurationIFrameUrl(context)
```

**Type**

Interface Function

**Description**

Provides the user interface for configuring a connection to your financial institution. For most plug-in authors, this is a URL to a Suitelet. This URL appears on the Connectivity Configuration subtab on a format profile record.

**Returns**

void

**Input Parameters**

- `context` - a JavaScript object with the following properties:
  - `pluginConfiguration`
  - `configurationIFrameUrl`
**getConfigurationIFrameUrl(context)**

```javascript
function getConfigurationIFrameUrl(context)
{
    var configurationId = context.pluginConfiguration.getConfigurationFieldValue({fieldName: "configuration_id");
    context.configurationIFrameUrl = "/app/site/hosting/scriptlet.nl?script=1&deploy=1&configurationId=" + configurationId;
}
```

**Example**

```javascript
context.pluginConfiguration.getConfigurationFieldValue({fieldName: "configuration_id"});
```
configurationIFrameUrl

**Type**
string

**Description**
The URL to a user interface for configuring a connection to your financial institution. This URL appears on the Connectivity Configuration subtab on the Format Profile page.

For an example where the property configurationIFrameUrl is used, see 
getConfigurationIFrameUrl(context).

getAccounts(context)

**Function Declaration**
function getAccounts(context)

**Type**
Interface Function

**Description**
This function allows the plug-in to retrieve accounts from your financial institution. It is invoked when you select the Account Mapping subtab on a format profile record or import bank data into NetSuite.

To add a financial institution account, see addAccount.

**Returns**
void

**Input Parameters**
context - a JavaScript object with the following properties:
- pluginConfiguration
- addAccount

**Example**

```javascript
function getAccounts(context) {
    var configurationId = context.pluginConfiguration.getConfigurationFieldValue({fieldName: "configuration_id"});
    var configuration = loadConfiguration(configurationId)
    context.addAccount({
        accountMappingKey: "12345",
        displayName: "Checking (XXXX11)",
        accountType: "BANK",
        currency: "USD",
        groupName: "Bank of America",
        lastUpdated: "2020-06-30T01:23:45"
    });
}
```

pluginConfiguration

See pluginConfiguration.

getConfigurationFieldValue

See getConfigurationFieldValue.
### addAccount

**Function Declaration**

```javascript
function addAccount(options)
```

**Type**

Object function

**Description**

This function allows the plug-in to add a financial institution account you want to map to a corresponding NetSuite account. This function should be called for each financial institution account. NetSuite displays the supplied accounts on the Account Mapping subtab on the Format Profile page.

**Returns**

void

**Input Parameters**

options - a JavaScript object with the following properties, all of which are strings:

- accountMappingKey – The financial institution account's unique identifier. For compliance reasons, this cannot be a credit card number.
  
  To check if the account mapping key contains a credit card number, the following takes place:
  
  □ NetSuite removes all non-numeric characters.
  
  □ If the remaining string is less than 13 characters or greater than 20 characters, NetSuite accepts the account mapping key since it does not contain a credit card number.
  
  □ If the remaining string is greater than 12 characters and less than 21 characters, NetSuite performs a Luhn check. If the check passes, NetSuite blocks you from using the account mapping key since it is highly likely to contain a credit card number. If the check fails, NetSuite accepts the account mapping key.

- displayName – The financial institution account name capable of being displayed.

- accountType – The account type that should map to a NetSuite account type. Valid types include:
  
  □ ACCOUNTS_PAYABLE
  
  □ ACCOUNTS_RECEIVABLE
  
  □ BANK
  
  □ COGS
  
  □ CREDIT_CARD
  
  □ DEFERRED_EXPENSE
  
  □ DEFERRED_REVENUE
  
  □ EQUITY
  
  □ EXPENSE
  
  □ FIXED_ASSET
  
  □ INCOME
  
  □ LONG_TERM LIABILITY
  
  □ NON_POSTING
  
  □ OTHER_ASSET
  
  □ OTHER_CURRENT_ASSET
  
  □ OTHER_CURRENT LIABILITY
  
  □ OTHER_EXPENSE
  
  □ OTHER_INCOME
  
  □ STATISTICAL
  
  □ UNBILLED_RECEIVABLES
getAccounts(context)

- currency – The financial institution account currency code, which is used on the Account Mapping subtab to filter by NetSuite accounts with the same currency. The currency input parameter must be a valid ISO 4217 currency code.
- groupName – The financial institution name capable of being displayed, which is used for grouping accounts from the same financial institution.
- lastUpdated – The last time the financial institution updated data for an account. This is useful because if the financial institution has not updated the account since the last import, then you know there is no new data to import. If getAccounts() returns a lastUpdated value for an account, NetSuite calls getTransactionData(), and the dataEndTime field for that account is populated based on the lastUpdated value. This hints to the plug-in that NetSuite is requesting bank data up to the date provided by the lastUpdated value.

getTransactionData(context)

**Function Declaration**

```javascript
function getTransactionData(context)
```

**Type**

Interface Function

**Description**

This function allows the plug-in to invoke the Parser plug-in to parse content into transactions. This happens when a bank data import is initiated.

**Note:** For a successful import, you need at least one active account mapping on the format profile record.

**Returns**

void

**Input Parameters**

context - a JavaScript object with the following properties:

- pluginConfiguration
- accountRequestsJSON
- addDataChunk
- returnAccountRequestsJSON

**Example**

```javascript
function getTransactionData(context)
{
    var configurationId = context.pluginConfiguration.getConfigurationFieldValue({fieldName:
        "configuration_id"});
    var configuration = loadConfiguration(configurationId)
    var accountRequests = JSON.parse(context.accountRequestsJSON);
    if (accountRequests != null) {
        accountRequests.forEach(function (accountRequest) {
            var accountId = accountRequest.accountMappingKey;
            var fromDateTime = accountRequest.dataStartTime;
            var toDateTime = accountRequest.dataEndTime;
            var downloadedData = "... Retrieve the account data using a web service request or a file transfer...";

            context.addDataChunk({dataChunk: downloadedData});
        });
    }
    context.returnAccountRequestsJSON({accountsJson: context.accountRequestsJSON});
}
```
pluginConfiguration

See pluginConfiguration.

getConfigurationFieldValue

See getConfigurationFieldValue.

accountRequestsJSON

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>A list of accounts required for the plug-in to query bank data, as well as the required data date ranges. The information retrieved is provided as a JSON format string in the following format:</td>
</tr>
</tbody>
</table>

```json
[{
  "dataStartTime":"2019-08-20T02:30:11",
  "dataEndTime":"2019-09-20T02:30:11",
  "accountMappingKey":"acct-123"},
* {
  "dataStartTime":"2019-08-20T02:30:11",
  "dataEndTime":"2019-09-20T02:30:11",
  "accountMappingKey":"acct-456"}
]
```

The properties embedded in the JSON string including the following:

- `dataStartTime` - The requested start time of the data to be retrieved. If the format profile is saved for the first time, NetSuite downloads data up to 60 calendar days old. After the first import, `dataStartTime` can also be dependent on the last successful import.
- `dataEndTime` - The requested end time of the data to be retrieved. `dataEndTime` is the time when a bank data import is initiated.
- `accountMappingKey` - The financial institution account's unique identifier

For an example where the property `accountRequestsJSON` is used, see `getTransactionData(context)`.

addDataChunk

<table>
<thead>
<tr>
<th>Function Declaration</th>
<th>function addDataChunk(options)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Interface Function</td>
</tr>
<tr>
<td>Description</td>
<td>This function allows you to transmit the financial institution's data file to NetSuite as a series of chunks. It encrypts the incoming chunks before storing them in the database. Each chunk size has a 25–million character limit.</td>
</tr>
<tr>
<td>Returns</td>
<td>void</td>
</tr>
<tr>
<td>Input Parameters</td>
<td>options — a JavaScript object with the property <code>dataChunk</code>, which is a chunk of data from your financial institution's data file. <code>dataChunk</code> is a string.</td>
</tr>
</tbody>
</table>

returnAccountRequestsJSON

<table>
<thead>
<tr>
<th>Function Declaration</th>
<th>function returnAccountRequestsJSON(options)</th>
</tr>
</thead>
</table>

### Type
| Interface Function |

### Description
This function allows you to return the list of accounts queried by the plug-in, as well as the reason for any query failures. The import job fails when one or more accounts return a failure reason during the import. Failure reasons are displayed on the Imported Statement List page.

If an account is provided by accountRequestsJSON but not returned by the Financial Institution Connectivity API, it is assumed that there is no account-specific failure.

### Returns
void

### Input Parameters
- `options` - a JavaScript object with the property `accountsJson`. `accountsJson` contains the list of financial institution accounts queried by the plug-in, as well as the reason for any query failures. This information is returned as a JSON format string:

```json
[{"dataStartTime":"2019-08-20T02:30:11", "dataEndTime":"2019-09-20T02:30:11", "accountMappingKey":"acct-123"},
 * {"dataStartTime":"2019-08-20T02:30:11", "dataEndTime":"2019-09-20T02:30:11", "accountMappingKey":"acct-456", "failureReason":"Invalid account"}
]
```

### Error Propagation
You should raise all user-facing error messages using the ‘N/error’ SuiteScript module. For details on this module, see the help topic [N/error Module](#).

In all user-facing errors, you must set the name property to FICONNECTIVITY_USER_ERROR. Errors not raised with the FICONNECTIVITY_USER_ERROR name property, as well as any exception not handled, are not surfaced to users. Instead, NetSuite displays a generic error message on the Format Profile page. Details about the error are available in the plug-in’s execution log.

The following sample code illustrates how to raise a user-facing error:

```javascript
throw error.create({  // error variable is the imported 'N/error' module
    name: 'FICONNECTIVITY_USER_ERROR',
    message: 'This message will be displayed on the Format Profile page'
});
```

The next sample code illustrates how to raise a non user-facing error:

```javascript
throw 'This message will not be displayed on the Format Profile page';
```