Bank 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

Bank Connectivity Plug-in Interface Overview ................................................................. 1
Limitations of Using the Bank Connectivity Plug-in ....................................................... 2
Developing a Bank Connectivity Plug-in ......................................................................... 2
   Creating the Bank Connectivity Plug-in Script File .................................................. 2
   Obtaining the Host Key .............................................................................................. 4
Administering a Bank Connectivity Plug-in ................................................................. 4
   Enabling Features for a Bank Connectivity Plug-in .................................................. 4
   Adding the Plug-in Implementation .......................................................................... 5
   Defining the Configuration Requirements ................................................................ 5
   Activating the Bank Connectivity Plug-in .................................................................. 6
   Testing the Bank Connectivity Plug-in ...................................................................... 6
   Manually Running the Bank Connectivity Plug-in ...................................................... 6
Bank Connectivity Plug-in Interface Definition (Beta) ................................................... 7
getRequiredConfigurationFields (context) ................................................................... 8
downloadPreviousDayBankStatementFile (context) ...................................................... 8
Object Types ................................................................................................................ 10
   BankAccount ............................................................................................................. 10
   BankConnectivityPluginConfiguration ................................................................. 11
   RequiredConfigurationFieldsInput .......................................................................... 11
   DownloadPreviousDayBankStatementFileInput ..................................................... 12
   RequiredConfigurationFieldsOutput ....................................................................... 13
   DownloadPreviousDayBankStatementFileOutput .................................................... 15
Bank Connectivity Plug-in Interface Overview

Important: The Bank 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.

Use the Bank Connectivity Plug-in to make secure electronic connections to financial institutions, for inbound transmissions, through NetSuite. Provided that your financial institution can generate a bank statement file to a specific location on a server, the plug-in can get the file and then upload it to NetSuite for you.

Note: The Bank Connectivity Plug-in is deprecated as of NetSuite 2020.1. To directly connect to a financial institution and automate bank data imports, use the Financial Institution Connectivity Plug-in. For details, see.

Inbound transmission enables you to automate cash balance reporting and schedule account statement downloads.

After the administrator installs and implements the plug-in, users can define specific configuration requirements. For example, a plug-in using Secure File Transfer Protocol (sFTP) would need authentication information and the location of the sFTP server.

The plug-in is capable of getting the results immediately when it makes a request for end-of-day bank statements. The plug-in formats the bank statement file and then sends it to NetSuite. After the import is complete, continue with other stages of the Cash Management workflow, such as transaction matching and reconciliation. There may be cases where NetSuite requires user input for bank authentication. NetSuite saves credentials securely.

The following diagram shows the synchronous workflow for the requestPreviousDayBankStatement function:

- NetSuite Code
- Request bank statement
- Plug-in
- Download bank statement
- Bank
- Return bank statement
- Save and store bank statement in the database

The Bank Connectivity Plug-in works in conjunction with the Parser plug-in. Therefore, the Bank Connectivity Plug-in supports any format that has a Parser plug-in.

The sample plug-in (sFTP plug-in) provides Host-to-Host (direct) Connectivity using sFTP, which is the supported communication protocol. NetSuite supports any created plug-in using the sFTP protocol.

For more information about the Bank 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>Limitations of Using the Bank Connectivity Plug-in</td>
</tr>
</tbody>
</table>
Limitations of Using the Bank Connectivity Plug-in

When using the Bank Connectivity Plug-in, keep in mind the following limitations:

- To connect to a financial institution to get bank statements, you have to manually trigger the sFTP connection. For details, see the help topic Automating Imports with the Financial Institution Connectivity Plug-in.
- The plug-in does not support IP address whitelisting.
- You cannot retrieve ZIP files.

Developing a Bank Connectivity Plug-in

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

- Creating the Bank Connectivity Plug-in Script File
- Obtaining the Host Key

Creating the Bank Connectivity Plug-in Script File

You must implement each Bank 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 basic sFTP connection using the 'N/sftp' SuiteScript module in the plug-in implementation script file:

```javascript
/**
 * @NApiVersion 2.0
 * @NScriptType bankConnectivityPlugin
 */
define(["N/sftp", "N/file"],
function(sftp, file) {
    return {
        getRequiredConfigurationFields: function (context) {
            //Connection details
            context.output.addConfigurationField({type:"TEXT", label:"URL", dataName:"url"});
            context.output.addConfigurationField({type:"TEXT", label:"User name", dataName:"username"});
        }
    };
})
```

Note: The 'N/sftp' SuiteScript module supports regular expressions as of 2 | 2019.2. For details, see the help topic N/sftp Module.
For details and breakdowns of the sample, see Bank Connectivity Plug-in Interface Definition (Beta).

The following table describes the interface functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getRequiredConfigurationFields (context)</code></td>
<td>Define the configuration requirements for your bank connectivity implementation. For example, a specific bank may require a secret user name and password to connect. This function is called when you open the plug-in configuration page. (used to define the fields you see on that page)</td>
</tr>
<tr>
<td><code>downloadPreviousDayBankStatementFile (context)</code></td>
<td>Request a statement from the bank. When you request a statement from NetSuite, the plug-in calls the function which makes the download request to the bank. The bank sends the statement back to the plug-in, then the plug-in formats and sends the statement to NetSuite. Continue with other stages of the Cash Management workflow, like transaction matching.</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 Plug-in Implementation and Activating the Bank Connectivity Plug-in.
- Due to limitations with how strings are handled in NetSuite, you cannot use the JavaScript `case` statement in the plug-in implementation script file.
- The plug-in allows up to 1000 usage units.

Obtaining the Host Key

The host key can be obtained using OpenSSH’s `ssh-keyscan` tool:

```
ssh-keyscan -t <hostKeyType> -p <port> <hostDomain>
```

Each bank has different conventions for file location and naming. Extend this sample to match the way your bank provides daily bank statements. Use the following set of basic configuration fields to download a specific file from the sFTP server:

- Download File Name – The specific file to download from the server.
- Download Directory – The folder location on the server (path relative to the user’s home directory) where the file resides.
- Download Timeout – The number of seconds to allow the file to download.
- Bank Statement Format — Any format that has a Parser plug-in is supported.

If any of these configuration fields are not required for your sFTP connection to a bank, you can remove them or change the values as required.

Administering a Bank Connectivity Plug-in

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

- Enabling Features for a Bank Connectivity Plug-in
- Adding the Plug-in Implementation
- Defining the Configuration Requirements
- Activating the Bank Connectivity Plug-in
- Testing the Bank Connectivity Plug-in
- Manually Running the Bank Connectivity Plug-in

Enabling Features for a Bank Connectivity Plug-in

To begin development of a Bank 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 is checked. If necessary, check the box and agree to the Terms of Service.
3. Click Save.

Adding the Plug-in Implementation

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

To add the plug-in implementation:

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 will provide one for you after you click Save. As a developer, when you write code that uses 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 accessible to the owner of the implementation. Choose Released to have the implementation accessible 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 appears on the Execution Log subtab for the implementation after you create it. Go to Customization &gt; Plug-ins &gt; Plug-in Implementations, select your implementation, and then click the Execution Log subtab.</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.

Defining the Configuration Requirements

After the administrator installs and implements the plug-in, you can define specific configuration requirements. The options available depend on the set-up configuration in the JavaScript file.
You may require a secret username or password to connect to the bank.

**To define configuration requirements:**

2. For your plug-in implementation, click **Edit**. NetSuite displays the implementation page for your plug-in.
3. Click **Configure**. NetSuite dynamically generates the form from the plug-in implementation.
4. If defined by the plug-in, you may need to set:
   - URL for the connecting bank
   - user name and password
   - host key and key type
   - port
   - file download information
5. Click **Save**. NetSuite saves your configuration for the plug-in.

**Activating the Bank Connectivity Plug-in**

To use the plug-in, activate your implementation. For more information, see the help topic Managing Custom Plug-in Implementations.

**To activate the plug-in:**

1. To activate the plug-in implementation, go to Customization > Plug-ins > Manage Plug-ins.
2. Under **Bank Connectivity**, check the box next to name of your implementation.
3. Click **Save**.

**Testing the Bank Connectivity Plug-in**

To test the plug-in, run the plug-in and then go to Transactions > Bank > Import Online Banking Data > List to view the downloaded statement.

**Manually Running the Bank Connectivity Plug-in**

If you want to run the plug-in, you can manually invoke the plug-in to download and import bank statement transactions. For details, see the help topic Automating Imports with the Financial Institution Connectivity Plug-in.
Bank Connectivity Plug-in Interface Definition (Beta)

**Warning:** The Bank Connectivity API is deprecated as of NetSuite 2020.1. To use a connectivity plug-in that can directly connect you to a financial institution and automate bank data imports, use the Financial Institution Connectivity API. For details, see the help topic Financial Institution Connectivity Plug-in Interface Definition.

The plug-in interface includes the following functions.

**Important:** You cannot change these function signatures and their return types in a Bank Connectivity Plug-in implementation.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getRequiredConfigurationFields (context)</code></td>
<td>Define the configuration requirements for your bank connectivity implementation. For example, a specific bank may require a secret user name and password to connect. This function is called when you open the plug-in configuration page.</td>
</tr>
<tr>
<td><code>downloadPreviousDayBankStatementFile (context)</code></td>
<td>Request a statement from the bank. When you request a statement from NetSuite, the plug-in calls the function which makes the download request to the bank. The bank sends the statement back to the plug-in, then the plug-in formats and sends the statement to NetSuite. Continue with other stages of the Cash Management workflow, like transaction matching.</td>
</tr>
</tbody>
</table>

The plug-in interface includes the following objects.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BankAccount</code></td>
<td>To provide the plug-in with required information, this object contains a small subset of data from a bank account record in NetSuite.</td>
</tr>
<tr>
<td><code>BankConnectivityPluginConfiguration</code></td>
<td>This object contains all the user-supplied configuration data (field values) for this plug-in.</td>
</tr>
<tr>
<td><code>RequiredConfigurationFieldsInput</code></td>
<td>The <code>getRequiredConfigurationFields (context)</code> function uses this object.</td>
</tr>
<tr>
<td><code>DownloadPreviousDayBankStatementFileInput</code></td>
<td>The <code>downloadPreviousDayBankStatementFile (context)</code> function uses this object. When the bank sends a requested statement back to the plug-in, the plug-in formats and sends the statement to NetSuite.</td>
</tr>
<tr>
<td><code>RequiredConfigurationFieldsOutput</code></td>
<td>The <code>getRequiredConfigurationFields (context)</code> function uses this object. The plug-in must use this object to provide any configuration fields it requires the user to complete. Based on this configured output, the required fields populate the Bank Connectivity Plug-In Configuration page in NetSuite.</td>
</tr>
</tbody>
</table>
getRequiredConfigurationFields (context)

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DownloadPreviousDayBankStatementFileOutput</td>
<td>The <code>downloadPreviousDayBankStatementFile (context)</code> function uses this object. When the bank sends a requested statement back to the plug-in, the plug-in formats and sends the statement to NetSuite.</td>
</tr>
</tbody>
</table>

**Function**

```javascript
void getRequiredConfigurationFields(RequiredConfigurationFieldsContext context) {
  // Connection details
  context.output.addConfigurationField({type: "TEXT", label: "URL", dataName: "url"});
  context.output.addConfigurationField({type: "TEXT", label: "Username", dataName: "username"});
  context.output.addConfigurationField({type: "PASSWORD", label: "Password", dataName: "password"});
  context.output.addConfigurationField({type: "TEXT", label: "Host Key", dataName: "hostKey"});
  context.output.addConfigurationField({type: "TEXT", label: "Host Key Type", dataName: "hostKeyType"});
  context.output.addConfigurationField({type: "TEXT", label: "Port", dataName: "port"});

  // File download details
  context.output.addConfigurationField({type: "TEXT", label: "Download Filename", dataName: "filename"});
  context.output.addConfigurationField({type: "TEXT", label: "Download Directory", dataName: "directory"});
  context.output.addConfigurationField({type: "TEXT", label: "Download Timeout", dataName: "timeout"});
  context.output.addConfigurationField({type: "TEXT", label: "Bank statement format", dataName: "format"});
}
```

For more details, see `addConfigurationField`.

downloadPreviousDayBankStatementFile (context)

**Function Declaration**

```javascript
void downloadPreviousDayBankStatementFile (DownloadPreviousDayBankStatementFileContext context) {
  // Function to request the previous day's statement from the bank.
  // When you request a statement from NetSuite, the plug-in calls this function,
  // which triggers the download request to the bank. The bank sends the statement
  // back to the plug-in, then the plug-in formats and sends the statement to NetSuite.
}
```
Returns

void

Input Parameters

DownloadPreviousDayBankStatementFileContext

Example

The following is an example of the whole downloadPreviousDayBankStatementFile function:

```javascript
downloadPreviousDayBankStatementFile: function (context) {
    var config = context.input.pluginConfiguration;
    var connection = sftp.createConnection({
        url: config.getConfigurationFieldValue({fieldName: "url"}),
        passwordGuid: config.getConfigurationFieldValue({fieldName: "password"}),
        hostKey: config.getConfigurationFieldValue({fieldName: "hostKey"}),
        hostKeyType: config.getConfigurationFieldValue({fieldName: "hostKeyType"}),
        username: config.getConfigurationFieldValue({fieldName: "username"}),
        port: parseInt(config.getConfigurationFieldValue({fieldName: "port"}))
    });
    var downloadedFile = connection.download({
        filename: config.getConfigurationFieldValue({fieldName: "filename"}),
        directory: config.getConfigurationFieldValue({fieldName: "directory"}),
        timeout: parseInt(config.getConfigurationFieldValue({fieldName: "timeout"}))
    });
    context.output.saveBankStatementFile({file: downloadedFile, bankStatementFormat: config.getConfigurationFieldValue({fieldName: "format"})});
}
```

As in this example, first create a convenience variable to easily reference the plug-in configuration values, which are provided as input to the function:

```javascript
var config = context.input.pluginConfiguration;
```

Create the sFTP connection by using the SuiteScript 'N/sftp' module.

Provide the configuration values for the fields from the defined `getRequiredConfigurationFields (context)`. The `fieldName` property in `config.getConfigurationFieldValue` maps directly to the `dataName` property in the `getRequiredConfigurationFields` definition. To define each configuration field line in `getRequiredConfigurationFields`, see `addConfigurationField`.

```javascript
var connection = sftp.createConnection({
    url: config.getConfigurationFieldValue({fieldName: "url"}),
    passwordGuid: config.getConfigurationFieldValue({fieldName: "password"}),
    hostKey: config.getConfigurationFieldValue({fieldName: "hostKey"}),
    hostKeyType: config.getConfigurationFieldValue({fieldName: "hostKeyType"}),
    username: config.getConfigurationFieldValue({fieldName: "username"}),
    port: parseInt(config.getConfigurationFieldValue({fieldName: "port"}))
});
```

Using the set configuration field values for the file name and directory, download the file to the NetSuite server.

Based on your bank's conventions, you may need to create functions to determine the file name or directory. For example, your bank may require the file name to include the date.

In our sample, there is a single directory containing a file that regenerates with new content once a day.

```javascript
var downloadedFile = connection.download({
```
Lastly, provide the downloaded file back to the NetSuite system. Once in NetSuite, the administrator can import the file and automatic matching runs on the statement transactions.

```javascript
context.output.saveBankStatementFile({file: downloadedFile, bankStatementFormat: config.getConfigurationFieldValue({fieldName:"format"})});
```

## Object Types

This section describes the data objects for the plug-in, which the functions use. These objects modify the specified data and include the following types:

- **BankAccount**
- **BankConnectivityPluginConfiguration**
- **RequiredConfigurationFieldsInput**
- **DownloadPreviousDayBankStatementFileInput**
- **RequiredConfigurationFieldsOutput**
- **DownloadPreviousDayBankStatementFileOutput**

### BankAccount

**Type** Interface object

**Description**
To provide the plug-in with required information, this object contains a small subset of data from a bank account record in NetSuite.

**Object Functions**
- `getAccountNumber()`
- `getId()`

**Parent Object(s)** N/A

**Child Object(s)** N/A

**getAccountNumber()**

**Function Declaration**
```
string getAccountNumber()
```

**Type** Object function

**Description**
This function returns the account number of the bank account record in NetSuite.

**Returns**
string

**Input Parameters**
None

**Parent object** BankAccount
getId()

Function Declaration: string getId()
Type: Object function
Description: This function returns the internal ID of the bank account record in NetSuite.
Returns: string
Input Parameters: None
Parent object: BankAccount

BankConnectivityPluginConfiguration

Type: Interface object
Description: This object contains all the user-supplied configuration data (field values) for this plug-in.
Object Functions: getConfigurationFieldValue
Parent Object(s): DownloadPreviousDayBankStatementFileInput
Child Object(s): N/A

getConfigurationFieldValue

Function Declaration: string getConfigurationFieldValue(string dataName)
Type: Object function
Description: NetSuite uses this function to return the user-supplied configuration value for a required configuration field.
This function retrieves the data using the defined keys in the addConfigurationField function's dataName parameter.
Returns: string – the user-supplied configuration value.
Input Parameters: dataName – the name of the key used to access the field when accessing the BankConnectivityPluginConfiguration input.
Parent object: BankConnectivityPluginConfiguration

Example

```context.output.saveBankStatementFile({file: downloadedFile, bankStatementFormat: config.getConfigurationFieldValue({fieldName:"format"}))});```

RequiredConfigurationFieldsInput

Type: Interface object
Description: The getRequiredConfigurationFields (context) function uses this object.
Object Functions
- `getLocale`
- `getPluginConfiguration`

Parent Object(s) N/A
Child Object(s) N/A

### getLocale

**Function Declaration**
```java
string getLocale()
```

**Type** Object function

**Description** NetSuite uses this object function to return the location of the current user session (for example, 'en_US').

**Returns** `string` – the locale string for the current session.

**Input Parameters** None

**Parent object**
- `DownloadPreviousDayBankStatementFileInput`
- `RequiredConfigurationFieldsInput`

### getPluginConfiguration

**Function Declaration**
```java
BankConnectivityPluginConfiguration getPluginConfiguration()
```

**Type** Object function

**Description** NetSuite uses this object function to return the user-supplied configuration for the required configuration fields, which the `getRequiredConfigurationFields(context)` function defines.

**Returns** `BankConnectivityPluginConfiguration` – the user-supplied configuration, if it exists. If the user does not supply any details for the configuration, this function returns null.

**Input Parameters** None

**Parent object**
- `DownloadPreviousDayBankStatementFileInput`
- `RequiredConfigurationFieldsInput`

### DownloadPreviousDayBankStatementFileInput

**Type** Interface object

**Description** The `downloadPreviousDayBankStatementFile(context)` function uses this object.

When you request a statement from NetSuite, the plug-in calls the function which triggers the download request to the bank. The bank sends the statement back to the plug-in.

**Object Functions**
- `getLocale`
- `getPluginConfiguration`
- `getSecureFileCabinetFolderId()`
**Object Types**

**Child Object(s)** BankConnectivityPluginConfiguration

---

### getSecureFileCabinetFolderId()

**Function Declaration**

```java
int getSecureFileCabinetFolderId()
```

**Type**

Object function

**Description**

NetSuite uses this object function to return the ID of the folder to which the bank statement file must be saved.

**Returns**

int – the ID of the folder.

**Input Parameters**

None

**Parent object**

DownloadPreviousDayBankStatementFileInput

---

### RequiredConfigurationFieldsOutput

**Type**

Interface object

**Description**

The `getRequiredConfigurationFields (context)` function uses this object.

The plug-in must use this object to provide any configuration fields it requires the user to complete. Based on this configured output, the required fields populate the Bank Connectivity Plug-In Configuration page in NetSuite.

**Object Functions**

- addConfigurationField
- RequiredConfigurationFieldsContainer

**Parent Object(s)**

N/A

**Child Object(s)**

N/A

---

### addConfigurationField

**Function Declaration**

```java
void addConfigurationField(string type, string label, string dataName, boolean required)
```

**Type**

Object function

**Description**

Add a required configuration field for this plug-in to be available on the Bank Connectivity Plug-In Configuration page in NetSuite.

Since SuiteScript cannot provide type-safety for the field type, choose the type parameter from one of the following string values:

- PASSWORD - For the mandatory password field, which provides passwords to SuiteScript modules, including the 'N/sftp' module. The password is securely stored as a token that these modules will be able to exchange for the actual password. If you need to store a password in plain-text for whatever reason, all of the configuration field data is stored securely
- TEXT - A plain text box
- TEXTAREA - A text area for multi-line content

**Returns**

void
### Input Parameters

- **type** – A string chosen from the list of field types (PASSWORD, TEXT, or TESTAREA).
- **label** – The label to display on the form.

**Note:** The label parameter can be localized by taking into account the input's `getLocale` function. Otherwise, this output object should be deterministic.

- **dataName** – The name of the key used to access the field when accessing the `BankConnectivityPluginConfiguration` input.
- **required** – Specifies whether a field is required. If true, the field is mandatory. The default is false.

### Parent object

- `RequiredConfigurationFieldsOutput`
- `DownloadPreviousDayBankStatementFileInput`

### Example

```javascript
getRequiredConfigurationFields: function (context) {
  // Connection details
  context.output.addConfigurationField({type: "TEXT", label: "URL", required: true, dataName: "url"});
  context.output.addConfigurationField({type: "TEXT", label: "User name", dataName: "username"});
  context.output.addConfigurationField({type: "PASSWORD", label: "Password", dataName: "password"});
  context.output.addConfigurationField({type: "TEXT", label: "Host Key", dataName: "hostKey"});
  context.output.addConfigurationField({type: "TEXT", label: "Host Key Type", dataName: "hostKeyType"});
  context.output.addConfigurationField({type: "TEXT", label: "Port", dataName: "port"});

  // File download details
  context.output.addConfigurationField({type: "TEXT", label: "Download Filename", dataName: "filename"});
  context.output.addConfigurationField({type: "TEXT", label: "Download Directory", dataName: "directory"});
  context.output.addConfigurationField({type: "TEXT", label: "Download Timeout", dataName: "timeout"});
  context.output.addConfigurationField({type: "TEXT", label: "Bank statement format", dataName: "format"});
},
```

This example mirrors all of the connection details provided by the 'N/sftp' module.

- **URL** – The URL of the bank's sFTP server.
- **User name** – The user name to connect to the bank's sFTP server.
- **Password** – The password for the user.
- **Host Key** – The fingerprint for the bank's sFTP server. For example, a fingerprint could be "c4:26:ab:cf:a0:15:0a:99:5f:f3:7b:ef:3b:19:d8:96". See Obtaining the Host Key.
- **Host Key Type** – The type of host key provided by the bank's server. The host key type is usually RSA.
- **Port** – The port the sFTP server is using, which is typically 22 since sFTP is over SSH.

### RequiredConfigurationFieldsContainer

**Function Declaration**

```javascript
RequiredConfigurationFieldsContainer()
```

**Type**

Object function

**Description**

NetSuite uses this function to extract the configurations field values that the plug-in returns.
Object Types

Returns
RequiredConfigurationFieldsContainer

Input Parameters
None

Parent object
RequiredConfigurationFieldsOutput

DownloadPreviousDayBankStatementFileOutput

Type
Interface object

Description
The `DownloadPreviousDayBankStatementFile` function uses this object. When the bank sends a requested statement back to the plug-in, the plug-in formats and sends the statement to NetSuite.

Object Function
- saveBankStatementFile

Parent Object(s)
N/A

Child Object(s)
N/A

saveBankStatementFile

Function Declaration
void saveBankStatementFile(File file, string bankStatementFormat)

Type
Object function

Description
NetSuite uses this function to save the bank statement file returned by the plug-in.

Returns
void

Input Parameters
file
bankStatementFormat

Parent object
DownloadPreviousDayBankStatementFileOutput