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

Email Capture Plug-in Overview ................................................................................................ 1
Email Capture Process Flow .................................................................................................... 1
Developing an Email Capture Plug-in Implementation ............................................................. 2
  Create a Plug-in Implementation Script File ........................................................................ 3
  Add the Plug-in Implementation ....................................................................................... 4
  Test the Plug-in Implementation ...................................................................................... 6
  Bundle the Plug-in Implementation ................................................................................. 6
Administering an Email Capture Plug-in Implementation ........................................................... 7
  Enable Features for an Email Capture Plug-in Implementation ........................................... 7
  Install an Email Capture Plug-in Bundle ............................................................. 7
  Enable the Email Capture Plug-in Implementation ........................................................... 8
  Create an Email Alias and Set Up Forwarding ................................................................ 8
Email Capture Plug-in Interface Description .............................................................................. 10
  process(email) .................................................................................................................. 10
  Email .......................................................................................................................... 11
  Address ...................................................................................................................... 16
  Attachment .................................................................................................................. 18
Email Capture Plug-in Overview

Use the Email Capture Plug-in to define NetSuite behavior based on the properties and contents of an email message. When NetSuite receives an email sent to an email address associated with an implementation of the plug-in, NetSuite executes the business logic defined in the implementation. For example, you can use SuiteScript and an implementation of the Email Capture Plug-in to process invoices, automate task creation and assignment, or automate escalations through email.

In an Email Capture plug-in implementation, you can define behavior for the plug-in implementation based on properties for the email, including header information, body text, and attachments. Depending on your requirements, you can create multiple plug-in implementations to define different types of NetSuite functionality. NetSuite generates an email address to which you send email messages for processing by each implementation of the plug-in.

Solution providers create implementations of the plug-in to define business logic based on the properties and contents of an email message. The plug-in implementations can be bundled and distributed as SuiteApps. NetSuite account administrators can install the SuiteApp and then NetSuite users can send email messages to the plug-in implementation, which then processes the email message and executes the business logic defined in the plug-in implementation.

**Important:** The plug-in does not capture email generated in NetSuite and sent to the email address associated with the plug-in. These include, for example, saved search and notification emails. This is to prevent your NetSuite email infrastructure from malfunctioning owing to an infinite loop of email. To view the email, use a different email recipient that is not connected to a capture plug-in.

For information about the Email Capture 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>Email Capture Process Flow</td>
</tr>
<tr>
<td>Developer</td>
<td>Developing an Email Capture Plug-in Implementation</td>
</tr>
<tr>
<td>Administrator</td>
<td>Administering an Email Capture Plug-in Implementation</td>
</tr>
</tbody>
</table>

**Important:** The Email Capture plug-in does not currently support SuiteScript 2.0. Only SuiteScript 1.0 is supported.

Email Capture Process Flow

The following steps describe the processing of an email message with an Email Capture plug-in implementation:

- User sends an email message. The email address is the address associated with the plug-in implementation that appears on the Manage Plug-ins page.
- The message is received by NetSuite. NetSuite routes it to the appropriate plug-in implementation as an object. You can have multiple Email Capture plug-in implementations active in a single account.
- The email message is processed by a plug-in implementation using the business logic defined in the process(email) interface function of the associated Email Capture plug-in implementation.
Developing an Email Capture Plug-in Implementation

A single Email Capture plug-in implementation can process multiple types of email messages or you can use a different Email Capture plug-in implementation for each type of email message. To create a plug-in implementation, use a developer account to develop and test the plug-in implementation.

Optionally, use SuiteBundler to bundle the plug-in objects and distribute them to other NetSuite accounts. NetSuite administrators use the bundle to install the plug-in implementation in a NetSuite account and enable the implementation. For more information about administration tasks for a Email Capture Plug-in bundle, see Administering an Email Capture Plug-in Implementation.

The following table describes the basic steps in developing a single plug-in implementation of the Email Capture plug-in:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable features</td>
<td>Enable the Server SuiteScript feature.</td>
</tr>
<tr>
<td>Create script file</td>
<td>Create the script file that contains the Email Capture plug-in implementation.</td>
</tr>
<tr>
<td>Add the plug-in implementation</td>
<td>Add the plug-in implementation using the plug-in script file and any required utility files that you created in the previous step to the development account.</td>
</tr>
</tbody>
</table>
Developing an Email Capture Plug-in Implementation

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test the plug-in implementation</td>
<td>Verify the behavior of the plug-in implementation. See Test the Plug-in Implementation.</td>
</tr>
<tr>
<td>Bundle the plug-in implementation (Optional)</td>
<td>Bundle both the plug-in implementation and other objects required by the plug-in implementation for distribution to other NetSuite accounts. See Bundle the Plug-in Implementation.</td>
</tr>
</tbody>
</table>

Create a Plug-in Implementation Script File

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

The following table describes the function and objects available in an Email Capture plug-in implementation script file:

<table>
<thead>
<tr>
<th>Function</th>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>process(email) interface function</td>
<td>Email object</td>
<td>Interface function that contains the implementation of the business logic of the Email Capture plug-in. You can use JavaScript and SuiteScript API functions to define the business logic. This function is mandatory for all Email Capture plug-in implementations and is called automatically by NetSuite when an email is sent to the email address associated with the plug-in implementation. For more information about how NetSuite uses this function, see Email Capture Process Flow.</td>
</tr>
<tr>
<td>Address object</td>
<td>Object that represents data from the from, to, cc, or reply-to headers in an email message sent to an Email Capture plug-in implementation.</td>
<td></td>
</tr>
<tr>
<td>Attachment object</td>
<td>Object that represents an attachment in an email message sent to an Email Capture plug-in implementation. Each Attachment object contains properties for the attachment file name, attachment type, and the value of the attachment file.</td>
<td></td>
</tr>
</tbody>
</table>
Developing an Email Capture Plug-in Implementation

Rules and Guidelines

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

- The plug-in script file can have any name, as long as it contains an implementation of the interface function.
- If you want to create utility files with helper functions for the main implementation file, you can include those files when you create the plug-in implementation for the Email Capture plug-in in NetSuite. See Add the Plug-in Implementation.
- You can use SuiteScript API functions in a plug-in implementation, for example, `nlapiLoadRecord(type, id, initializeValues)` and `nlapiSearchRecord(type, id, filters, columns)`. However, using these APIs may negatively affect the performance of the plug-in implementation. NetSuite recommends limiting the use of these APIs, where possible, to improve performance of the plug-in implementation. In general, searching for records yields better performance than loading NetSuite records. Governance limits also apply to these functions. The plug-in script file allows up to 1000 usage units when used with SuiteScript. For more information, see the help topic SuiteScript 1.0 API Governance.
- For security reasons, you are not permitted to send an email message to the email capture plugin from within NetSuite. You can only send a message to the email capture plugin using an external mail client. For example, it is not possible to submit a script upon customer creation, to send an email to the email capture plugin.

Add the Plug-in Implementation

When you have finished creating the plug-in implementation script file, create a new 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 create a new plug-in implementation:

1. Review the interface description for your core plug-in.

Email Capture Plug-in Guide
2. Create a JavaScript file for the implementation. Use this file to define the logic for the methods listed in the interface description.

3. In the UI, click Customization > Plug-ins > Plug-in Implementations > New.

4. In the **Script File** field, open the script file or add a new file.

5. Click **Create Plug-in Implementation**.

6. On the Select Plug-in Type page, click the Email Capture link.

7. 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. The plug-in implementation appears in the following locations:</td>
</tr>
<tr>
<td></td>
<td>Manage Plug-ins page. Page used by administrators to enable/disable the plug-in implementation in their account.</td>
</tr>
<tr>
<td></td>
<td>Bundle Builder. Select this name in the Bundle Builder to distribute the plug-in implementation to other accounts.</td>
</tr>
<tr>
<td>ID</td>
<td>Internal ID for the implementation for use in scripting. If you do not provide an ID, NetSuite provides one for you when you click <strong>Save</strong>.</td>
</tr>
<tr>
<td>Status</td>
<td>Current status for the implementation. Choose <strong>Testing</strong> to have the implementation accessible to the owner of the implementation. Choose <strong>Released</strong> to have the implementation accessible to all accounts in a production environment.</td>
</tr>
<tr>
<td>Log Level</td>
<td>Logging level you want for the script. Select <strong>Debug</strong>, <strong>Audit</strong>, <strong>Error</strong>, or <strong>Emergency</strong>. These messages appear on the <strong>Execution Log</strong> subtab for the plug-in implementation.</td>
</tr>
<tr>
<td>Execute As Role</td>
<td>Role that the script runs as. The <strong>Execute As Role</strong> field provides role-based granularity in terms of the permissions and restrictions of the executing script. The <strong>Current Role</strong> value indicates that the script executes with the permissions of the currently logged-in NetSuite user.</td>
</tr>
<tr>
<td>Note</td>
<td>You can create the custom role during testing to test the plug-in implementation with the proper role. The role requires the SuiteScript permission. You can then bundle the custom role to distribute it with the plug-in implementation. See <strong>Test the Plug-in Implementation</strong> and <strong>Bundle the Plug-in Implementation</strong>.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description of the implementation. The description appears for the implementation on the Plug-In Implementations page.</td>
</tr>
<tr>
<td>Owner</td>
<td>User account that owns the implementation. Default is the name of the logged in user.</td>
</tr>
<tr>
<td>Inactive</td>
<td>Indicates the plug-in implementation does not run in the account. Inactivate a plug-in implementation, for example, to temporarily disable it for testing purposes.</td>
</tr>
</tbody>
</table>

8. On the **Scripts** subtab, in the **Implementation** list, change the JavaScript file that contains the implementation of the plug-in, if required.

9. On the **Scripts** subtab, in the **Library Script File** list, select any utility script files or supporting library files that are required by the plug-in script file.

10. On the **Unhandled Errors** subtab, specify who to notify if script errors occur.
    - To notify the user that is logged in and running the script, check the **Notify Current User** box.
To notify the script owner, check the Notify Script Owner box. The Notify Script Owner box is checked by default.

To notify all administrators, check the Notify All Admins box.

To notify a group about the error, select the group to notify. Only existing groups that were set up in are available.

Enter individual email addresses in the Notify Emails field. Separate multiple email addresses with a semi-colon.

11. Click Save.

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

**Note:** You can use SuiteCloud Development Framework (SDF) to manage email capture plug-in implementations as part of file-based customization projects. For information about SDF, see the help topic SuiteCloud Development Framework Overview.

You can use the Copy to Account feature to copy an individual email capture plug-in implementation to another of your accounts. When you edit an email capture plug-in implementation, a clickable Copy to Account option is available in the upper right corner. For information about Copy to Account, see the help topic Copy to Account Overview.

### Debugging a Core Plug-in Implementation

#### Test the Plug-in Implementation

To test an Email Capture plug-in implementation, perform the following tasks:

- Enable the implementation. Use the Manage Plug-ins page to enable the plug-in implementation in the development account for testing. The Manage Plug-ins page includes the email address associated with the plug-in implementation. See Enable the Email Capture Plug-in Implementation.

- Create a custom role for the plug-in implementation. The Execute As Role field determines the role under which the plug-in implementation script runs. This role requires the SuiteScript permission. For more information about the Execute As Role field, see Add the Plug-in Implementation. For more information about custom roles in NetSuite, see the help topic Customizing or Creating NetSuite Roles.

- Send email messages to the address for the plug-in implementation to test the 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 Email Capture plug-in implementation, you can 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 Email Capture plug-in implementation script file and utility files, custom roles, and the plug-in implementation object, create a bundle with SuiteBundler. After you create the bundle, administrators can install the bundle in production accounts.

The following table lists the objects you must include in the bundle and their location on the Select Objects page in the Bundle Builder:
## Administering an Email Capture Plug-in Implementation

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

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

- **Enable Features for an Email Capture Plug-in Implementation**
- **Install an Email Capture Plug-in Bundle**
- **Enable the Email Capture Plug-in Implementation**
- **Create an Email Alias and Set Up Forwarding**

### Enable Features for an Email Capture Plug-in Implementation

Before you install an Email Capture Plug-in SuiteApp, make sure that the Server SuiteScript feature is enabled in the account.

**To enable features for the Email Capture Plug-in:**

1. Choose Setup > Company > Enable Features.
2. On the **SuiteCloud** subtab, enable the Server SuiteScript feature.
3. If necessary, check the box and agree to the Terms of Service.
4. Click **Save**.

### Install an Email Capture Plug-in Bundle

A developer can create an implementation of the Email Capture 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 an Email Capture plug-in bundle:**

**Important:** Before attempting to install this bundle, please check with your NetSuite Administrator that the bundle has been shared on your account. See the help topic **Sharing a Bundle**.

1. Go to Customization > SuiteBundler > Search & Install Bundles (Administrator role).
2. On the Install Bundle page, select **Advanced** and choose **Production Account** in the **Location** dropdown.

---
3. In the Keywords, enter Email Capture Bundle and click Enter.
4. Select the bundle from the list.
5. 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 bundle may not have been shared with your account. To get access to the bundle, contact the developer or NetSuite Technical Support.

Enable the Email Capture Plug-in Implementation

Enable the Email Capture plug-in implementation to activate it in your account.

**To enable the Email Capture plug-in implementation:**

1. As an administrator go to: Customization > Plug-ins > Manage Plug-ins.
2. Under Email Capture Plugin, check the Log Email Details box next to the name of the plug-in implementation.

3. Click Save.

**Note:** NetSuite automatically generates a unique capture email address for each plug-in implementation. The email address appears next to the plug-in implementation name on the Manage Plug-in Implementations page. The plug-in implementation executes when users send an email message to the address associated with the plug-in implementation.

Create an Email Alias and Set Up Forwarding

NetSuite generates an email address for use with the plug-in implementation. Users can send email messages to the implementation's email address for processing. However, you may want to create
administering an email capture plug-in implementation

an email alias on your company mail server to mask the actual email address required by the plug-in implementation.

for example, if your account ID was 123456, the NetSuite-generated email address for a plug-in implementation look something like **emails.123456.xxx.xxx@1234567.email.netsuite.com** and the implementation processes email messages to create invoice records in NetSuite. You can create an email alias named **invoices@example.com** on your company mail server that forwards to the actual implementation email address.

you can then release **invoices@example.com** as the email address for your customers to use when sending invoices.
Email Capture Plug-in Interface Description

Interface Function

The Email Capture plug-in interface includes the following functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>process(email)</td>
<td>Interface function that contains the implementation of the business logic of the Email Capture plug-in. You can use JavaScript and SuiteScript API functions to define the business logic.</td>
</tr>
<tr>
<td></td>
<td>This function is mandatory for all Email Capture plug-in implementations and is called automatically by NetSuite when an email is sent to the email address associated with the plug-in implementation.</td>
</tr>
<tr>
<td></td>
<td>For more information about how NetSuite uses this function, see Email Capture Process Flow.</td>
</tr>
</tbody>
</table>

Email Capture Plug-in Object Model

The following diagram shows the Email Capture plug-in object model:

```
EMAIL
  - From
  - To[]
  - Cc[]
  - ReplyTo

ADDRESS
  - Name
  - Email

ATTACHMENT
  - Name
  - Type
  - Value

process(email)

Function Declaration    void process(Email email)
Type                     Interface function
Description              Interface function that contains the implementation of the business logic of the Email Capture plug-in. NetSuite calls this function when it receives an email message sent to the email address associated with the plug-in implementation. See Email Capture Process Flow.
Use the methods available to the **Email** object to retrieve the properties of the email message, such as sender and recipient fields as **Address** objects, subject and body fields, sent date, and attachments as **Attachment** objects.

**Returns**

```
void
```

**Parameters**

- **Email**

**Example**

```javascript
function process(email) {
    // log all information in email
    var fromAddress = email.getFrom();
    logAddress('from', fromAddress);

    var to = email.getTo();
    for (var indexTo in to) {
        logAddress('to', to[indexTo]);
    }

    var cc = email.getCc();
    for (var indexCc in cc) {
        logAddress('cc', cc[indexCc]);
    }

    logAddress('replyTo', email.getReplyTo());
    log('sent', email.getSentDate());
    log('subject', email.getSubject());
    log('text body', email.getTextBody());
    log('html body', email.getHtmlBody());

    var attachments = email.getAttachments();
    for (var indexAtt in attachments) {
        logAttachment('att', attachments[indexAtt]);
    }
    ...
}
```

**Email**

**Type**

Interface input object

**Description**

Object that represents an email message sent to the Email Capture plug-in implementation. Use the methods available to the Email object to retrieve the following properties:

- Sender and recipient headers. The **From**, **To**, **Cc**, and **ReplyTo** properties of the email message as an **Address** object. See `getFrom()`, `getTo()`, `getCc()`, and `getReplyTo()`.
- Date. Date the email message was sent. See `getSentDate()`.
- Subject field and email body. The value of the subject field and the value of the email body as a text string or HTML string. See `getSubject()`, `getTextBody()`, and `getHtmlBody()`.
- Attachment. An array of all attachments as **Attachment** objects. See `getAttachments()`.
Methods

- `getFrom()`
- `getTo()`
- `getCc()`
- `getReplyTo()`
- `getSentDate()`
- `getSubject()`
- `getTextBody()`
- `getHtmlBody()`
- `getAttachments()`

Parent Object(s)  n/a

Child Object(s)
- `Address`
- `Attachment`

---

**getFrom()**

**Function Declaration**

Address getFrom()

**Type**

Object method

**Description**

Returns an array of `Address` objects that represent the `from` header on an email message sent to an Email Capture plug-in implementation.

**Returns**

`Address[]`

**Since**

2015.1

**Input Parameters**

None.

**Parent object**

- Email

**Example**

```javascript
function process(email) {
    ...
    var fromAddress = email.getFrom();
    logAddress('from', fromAddress);
    ...
}
```

---

**getTo()**

**Function Declaration**

Address getTo()

**Type**

Object method

**Description**

Returns an array of `Address` objects that represent the `to` header on an email message sent to an Email Capture plug-in implementation.

**Returns**

`Address`

**Since**

2015.1
**Input Parameters**

None.

**Parent object**

- Email

**Example**

```javascript
function process(email) {
  ...
  var to = email.getTo();
  for (var indexTo in to)
  {
    logAddress('to', to[indexTo]);
  }
  ...
}
```

**getCc()**

<table>
<thead>
<tr>
<th>Function Declaration</th>
<th>Address� getCc()</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Object method</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Returns an array of Address objects that represent the cc header on an email message sent to an Email Capture plug-in implementation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Returns</strong></th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Since</strong></td>
<td>2015.1</td>
</tr>
<tr>
<td><strong>Input Parameters</strong></td>
<td>None.</td>
</tr>
<tr>
<td><strong>Parent object</strong></td>
<td>Email</td>
</tr>
</tbody>
</table>

**Example**

```javascript
function process(email) {
  ...
  var cc = email.getCc();
  for (var indexCc in cc)
  {
    logAddress('cc', cc[indexCc]);
  }
  ...
}
```

**getReplyTo()**

<table>
<thead>
<tr>
<th>Function Declaration</th>
<th>Address� getReplyTo()</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Object method</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Returns an Address object that represents the reply-to header on an email message sent to an Email Capture plug-in implementation.</td>
</tr>
</tbody>
</table>
process(email)

Returns
Address
Since
2015.1
Input Parameters
None.
Parent object
■ Email

Example

```javascript
function process(email) {
    ...
    var replyAddr = email.getReplyTo();
    ...
}
```

getsendDate()

Function Declaration
Date getSentDate()
Type
Object method
Description
Returns a JavaScript Date object that represents the sent date on an email message sent to an Email Capture plug-in implementation.
Returns
Date
Since
2015.1
Input Parameters
None.
Parent object
■ Email

Example

```javascript
function process(email) {
    ...
    var date = email.getSentDate();
    ...
}
```

getSubject()

Function Declaration
string getSubject()
Type
Object method
Description
Returns a string that represents the subject field in an email message sent to an Email Capture plug-in implementation. Returns an empty string if the email message does not contain text in the subject field.
Returns
string
Since
2015.1
**Input Parameters**
None.

**Parent object**
- Email

**Example**

```javascript
function process(email) {
    ...
    var subject = email.getSubject();
    ...
}
```

**getTextBody()**

**Function Declaration**
```javascript
string getTextBody()
```

**Type**
Object method

**Description**
Returns a string that represents the plaintext part of the body of an email message sent to an Email Capture plug-in implementation. Returns null if there is no text in the body of the email message.

If there is text in the body field, this method returns a string regardless of the format of the email message. For example, if an email message was sent in HTML format, the string contains the text of the body field without any HTML formatting.

**Returns**
string

**Since**
2015.1

**Input Parameters**
None.

**Parent object**
- Email

**Example**

```javascript
function process(email) {
    ...
    log('text body', email.getTextBody());
    ...
}
```

**getHtmlBody()**

**Function Declaration**
```javascript
string getHtmlBody()
```

**Type**
Object method

**Description**
Returns a string that represents the HTML content included in the body field of an email message sent to an Email Capture plug-in implementation. Returns null if there is no HTML content in the body of the email message.

For example, this method returns null if the body of an email message contains plain text with no formatting.

**Returns**
string
process(email)

Since 2015.1
Input Parameters None.
Parent object ▪ Email

Example

```javascript
function process(email) {
    ...
    log('html body', email.getHtmlBody());
    ...
}
```

getAttachments()

Function Declaration Attachment[] getAttachments()
Type Object method
Description Returns an array of Attachment objects that represent the attachments included in an email message. Returns an empty array of the email message if does not include an attachment.
Use the methods available to the Attachment object to get the name, type, and value of each attachment.
Returns Attachment[]
Since 2015.1
Input Parameters None.
Parent object ▪ Email

Example

```javascript
function process(email) {
    ...
    var attachments = email.getAttachments();
    for (var indexAtt in attachments)
    {
        logAttachment('att', attachments[indexAtt]);
    }
    ...
}
```

Address

Type Object
Description Object that represents data from the from, to, cc, or reply-to headers in an email message sent to an Email Capture plug-in implementation.
Methods

- getName()
- getEmail()

Parent Object(s)

- Email

Child Object(s)

n/a

gtern() (function)

Function Declaration

string getName()

Type

Object method

Description

Returns a string that represents the name associated with an email address in the from, to, cc, or reply-to headers of an email message. Returns null if the address does not contain a name associated with the email address.

Returns

string

Since

2015.1

Input Parameters

None.

Parent object

- Address

Example

```javascript
function process(email) {
    ...
    var to = email.getTo();
    for (var indexTo in to)
    {
        logAddress('to', to[indexTo]);
    }
    ...
    function logAddress(label, address)
    {
        if (address)
        {
            nlapiLogExecution('DEBUG', 'Email - ' + label + ' : ' + address.getName() + ', ' + address.getEmail());
        }
    }
    ...
}
```

gtern() (function)

Function Declaration

string getEmail()

Type

Object method

Description

Returns a string that represents the email address in the from, to, cc, or reply-to headers of an email message.
The headers must contain an email address.

**Returns**
string

**Since**
2015.1

**Input Parameters**
None.

**Parent object**
- Address

**Example**

```javascript
function process(email) {
    ...
    var to = email.getTo();
    for (var indexTo in to) {
        logAddress('to', to[indexTo]);
    }
    ...
    function logAddress(label, address) {
        if (address) {
            nlapiLogExecution('DEBUG', 'Email - ' + label + ': ' + address.getName() + ', ' + address.getEmail());
        }
    }
    ...
}
```

**Attachment**

**Type**
Object

**Description**
Object that represents an attachment in an email message sent to an Email Capture plug-in implementation. Each Attachment object contains properties for the attachment file name, attachment type, and the value of the attachment file.

**Methods**
- getName()
- getType()
- getValue()

**Parent Object(s)**
- Email

**Child Object(s)**
- n/a

**getName()**

**Function Declaration**
string getName()

**Type**
Object method
process(email)

Description
Returns the file name for an attachment in an email message.

Returns
string

Since
2015.1

Input Parameters
None.

Parent object
- Attachment

Example

```javascript
function process(email) {
    ...
    var attachments = email.getAttachments();
    for (var indexAtt in attachments) {
        logAttachment('att', attachments[indexAtt]);
    }
    ...

    function logAttachment(label, attachment) {
        nlapiLogExecution('DEBUG', 'Att - ' + label + ': ' + attachment.getName() + ', ' + attachment.getType());
        nlapiLogExecution('DEBUG', 'Att - ' + label + ' - value: ' + attachment.getValue());
    }
    ...
}
```

getType()

Function Declaration
string getType()

Type
Object method

Description
Returns the file type of an attachment in an email message as a string. For example, this method returns PLAINTEXT, PDF, and MISCBINARY for text, PDF, and Microsoft Word files, respectively.

Returns
string

Since
2015.1

Input Parameters
None.

Parent object
- Attachment

Example

```javascript
function process(email) {
    ...
    var attachments = email.getAttachments();
    for (var indexAtt in attachments) {
        logAttachment('att', attachments[indexAtt]);
    }
    ...
```
getEmailAttachment

function logAttachment(label, attachment)
{
    nlapiLogExecution('DEBUG', 'Att - ' + label + ': ' + attachment.getName() + ', ' + attachment.getType());
    nlapiLogExecution('DEBUG', 'Att - ' + label + ' - value: ' + attachment.getValue());
}

getValue()

Function Declaration  string getValue()

Type  Object method

Description  Returns a text string for a text file attachment or base-64 encoded string for binary attachment types of an email message. You can use getType() to define the behavior of the plug-in implementation depending on the file type of the attachment.

Returns  string

Since  2015.1

Input Parameters  None.

Parent object  ■ Attachment

Example

function process(email) {
    ...
    var attachments = email.getAttachments();
    for (var indexAtt in attachments)
    {
        logAttachment('att', attachments[indexAtt]);
    }
    ...
    ...
}

function logAttachment(label, attachment)
{
    nlapiLogExecution('DEBUG', 'Att - ' + label + ': ' + attachment.getName() + ', ' + attachment.getType());
    nlapiLogExecution('DEBUG', 'Att - ' + label + ' - value: ' + attachment.getValue());
}

    ...
}