Adobe Sign

API Implementation Guide

Last Updated: April 27, 2016

Table of Contents

Overview **3**

Getting Started **3**

Adobe Sign Interface **4**

Creating An Integration Key **4**

Retrieving Your Integration Key **11**

Revoking Access and OAuth Tokens **13**

Creating Your Application **16**

Configuring OAuth for Your Application **19**

Adobe Sign Scenarios **24**

Scenario 1: Sending & Tracking from an Application with REST-based API **24**

Scenario 2: Sending & Tracking from an Application with SOAP-based API **25**

Scenario 3: Embedding Adobe Sign eSigning in An Application **27**

Exposing Additional Adobe Sign Actions **28**

Adobe Sign Events **29**

Event System Requirements **29**

List of Supported Events **29**

Using Adobe Sign Events **31**

Adding an Event Handler on the Parent Page 31

Embedding the Adobe Sign UI in an iFrame 31

Overview

Developers can build a variety of different integrations with Adobe Sign using a web services API for communications. A web service is a standards-based, secure and scalable method of establishing communications between systems over the Web. Once built, integrations allow users to initiate the Adobe Sign signing experience entirely from within the external application. Developers can also incorporate the functionality of Adobe Sign into their external applications by embedding the Adobe Sign user interface (UI) within those applications. External applications can also receive status updates in real-time for transactions initiated using Adobe Sign. These external applications can also retrieve and store copies of the signed agreements.

This document contains information on the API integration process. It includes the following sections:

- Adobe Sign Interface—This section provides instructions on how to use the Adobe Sign interface to establish an integration with an external application. It includes information on document keys and the configuration of OAuth. Note that the Adobe Sign API interface is only available to Enterprise Premium [P2] customers.
- Adobe Sign Scenarios—This section describes various scenarios for using the Adobe Sign APIs to integrate with an application. Both SOAP and REST-based APIs are covered.
- Adobe Sign API Events—This section provides information on integrating Adobe Sign and external applications by embedding the Adobe Sign user interface (UI) into those external applications. It also covers how to send information about events or actions in Adobe Sign to external applications.

Getting Started

Developers can sign up online for a free unlimited Adobe Sign Developer account at https://secure.echosign.com/public/upgrade?type=developer. Key developer resources are available as follows:

REST

- REST-based Documentation: https://secure.echosign.com/redirect/latestRestApiMethods

SOAP

- SOAP-based Documentation: https://secure.echosign.com/redirect/latestApiMethods
- Adobe Sign API WSDL: https://secure.echosign.com/redirect/latestApiWsdl

Note: Although the SOAP-based API is supported, we strongly encourage customers to implement using the REST-based API. Future development efforts will center on the REST API.

SDK with sample code

- Development SDK: https://secure.echosign.com/redirect/latestApiDevelopersKit

OAuth

- OAuth Documentation: https://secure.na1.echosign.com/public/static/oauthDoc.jsp

Adobe Sign Interface

Developers can integrate Adobe Sign with proprietary solutions (business applications or company websites) using the Adobe Sign REST or SOAP-based APIs. Developers can also incorporate the functionality of the Adobe Sign into their external applications by embedding the Adobe Sign user interface (UI) within those applications.

Developers must be authorized to access the Adobe Sign data that the integration will need to modify or create. The preferred authorization method involves using the Adobe Sign OAuth permission model, which complies with the OAuth 2.0 specification. However, if the solution does not support OAuth, you can also use an Integration Key.

The SOAP-based API works with legacy API Keys, the new Integration Keys, and OAuth tokens. The REST-based API works with Integration Keys and OAuth tokens.

Note: Starting with document service 16, API keys have been deprecated and are no longer supported. If you have a legacy application and have used a legacy API key to integrate, we recommend that you replace this key with an Integration Key or an OAuth token to provide additional security and customization options.

Creating An Integration Key

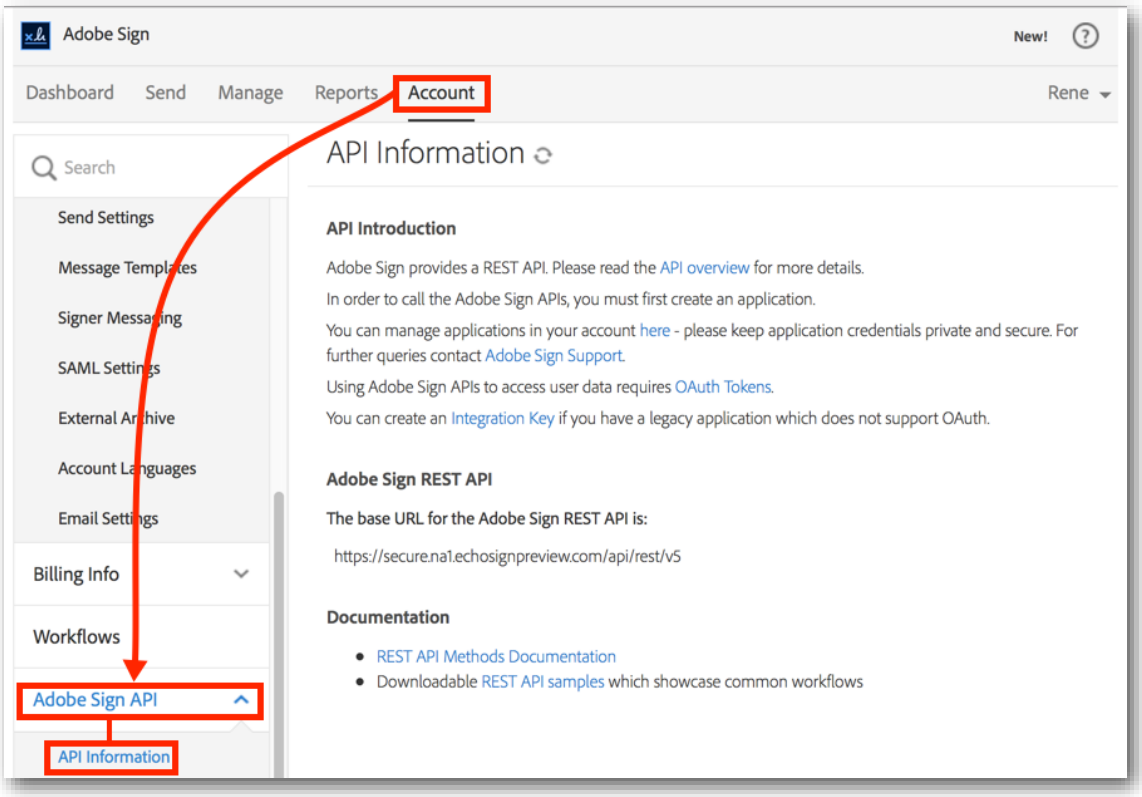
OAuth is the preferred permission model for when integrating Adobe Sign with external applications. (See the Using OAuth to Access Adobe Sign APIs document for more information on OAuth). However, you can create and use an Integration Key to integrate with applications that do not support OAuth or if you prefer to not use OAuth with your application. Only an Account or Group Admin can create an Integration Key.

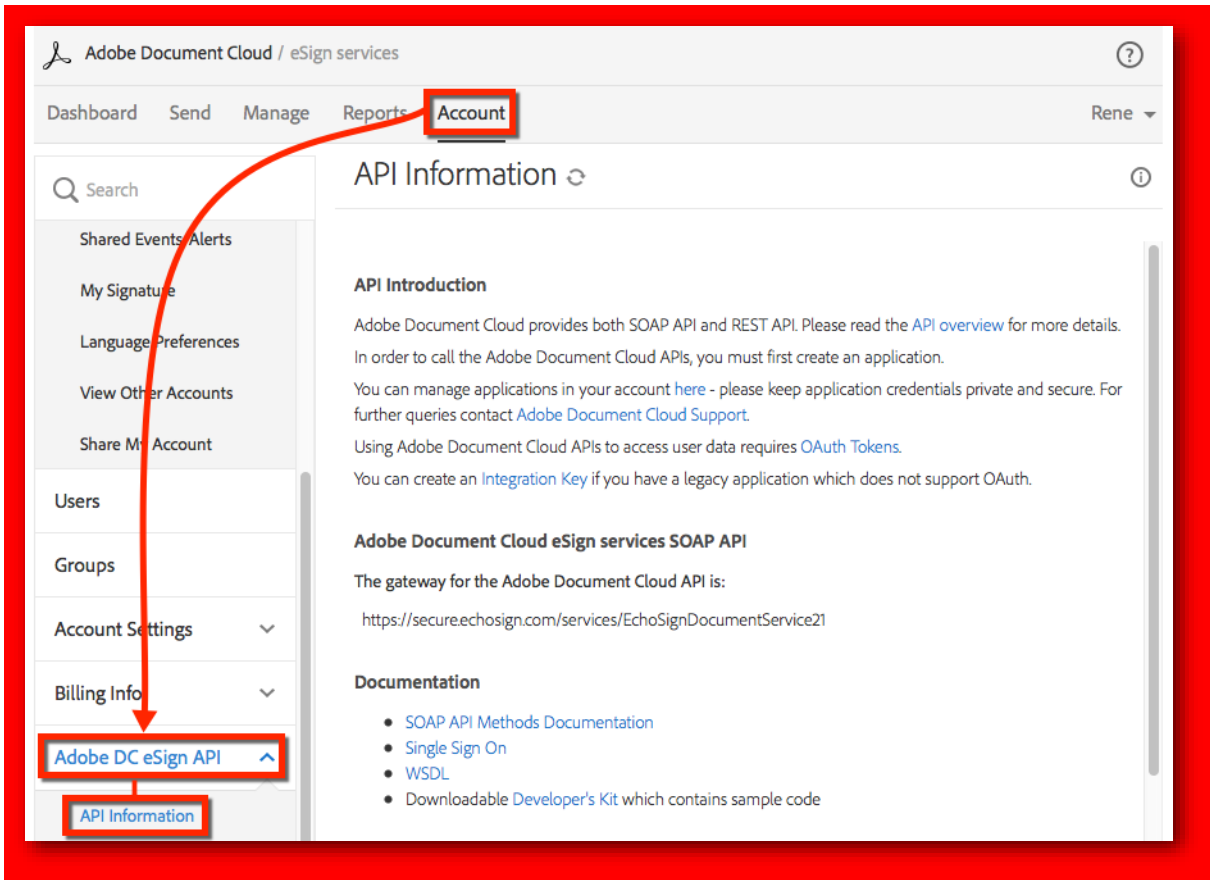
Note: If you plan to integrate more than one application using an Integration Key, we recommend that you create a unique Integration Key for each.

If you need to view or revoke an Integration Key after it is created, this can be done from the Access Tokens page. (See Retrieving Your Integration Key for more information.)

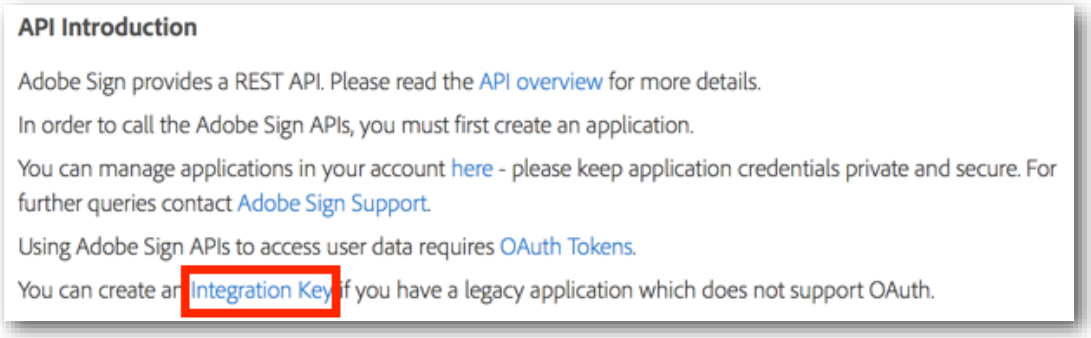
To create an Integration Key:

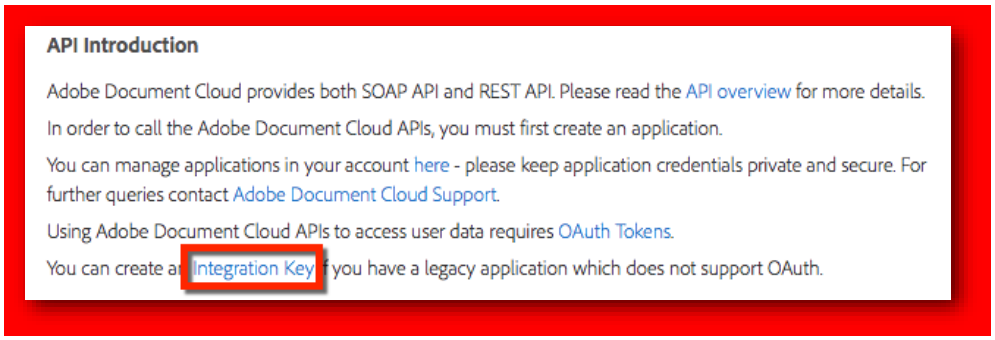
1. Click the **Account** tab, click on **Adobe Sign API**, then click **API Information.**



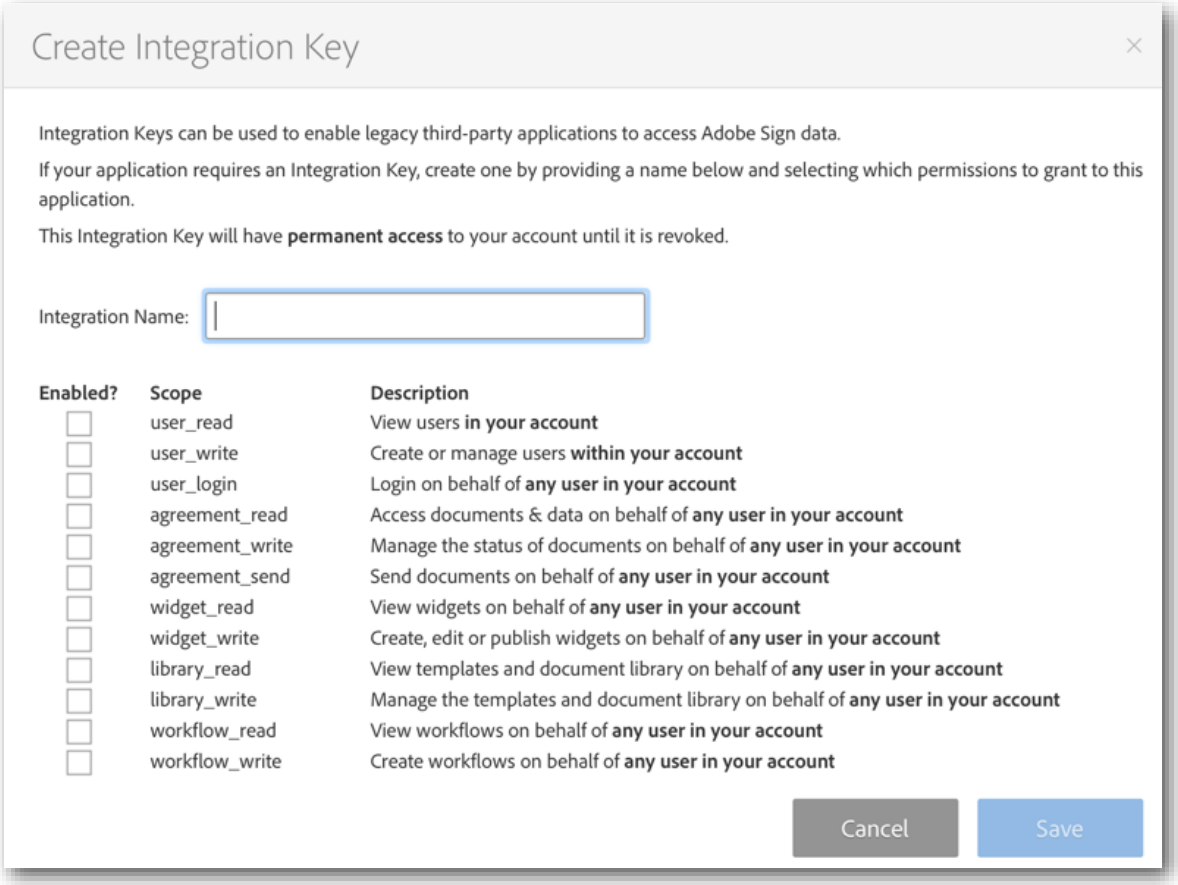


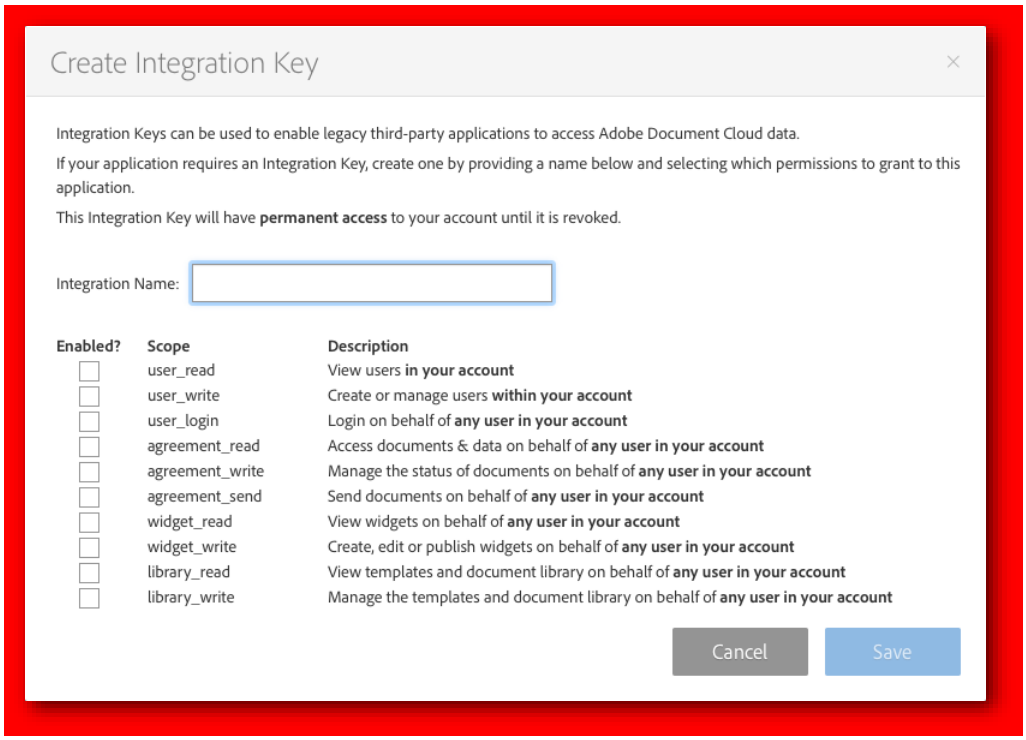
2. In the API Information page, click the **Integration Key** link.





The *Create Integration Key* dialog displays.

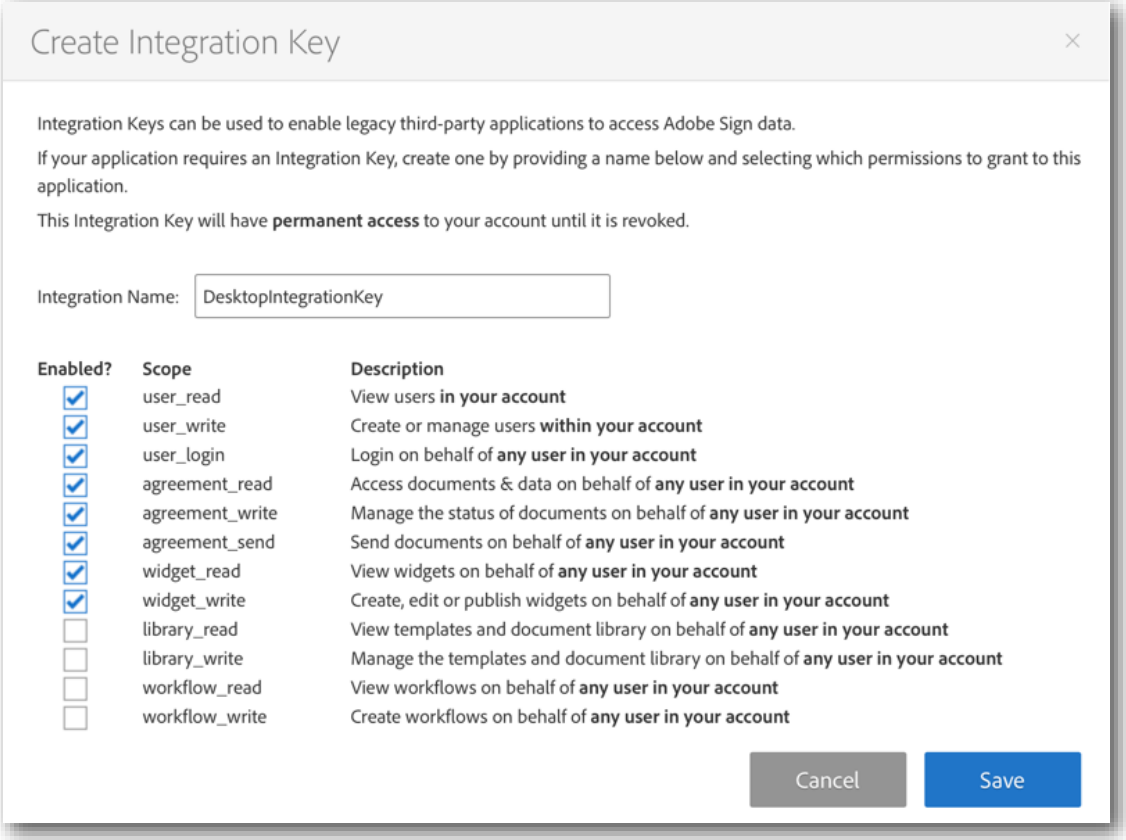


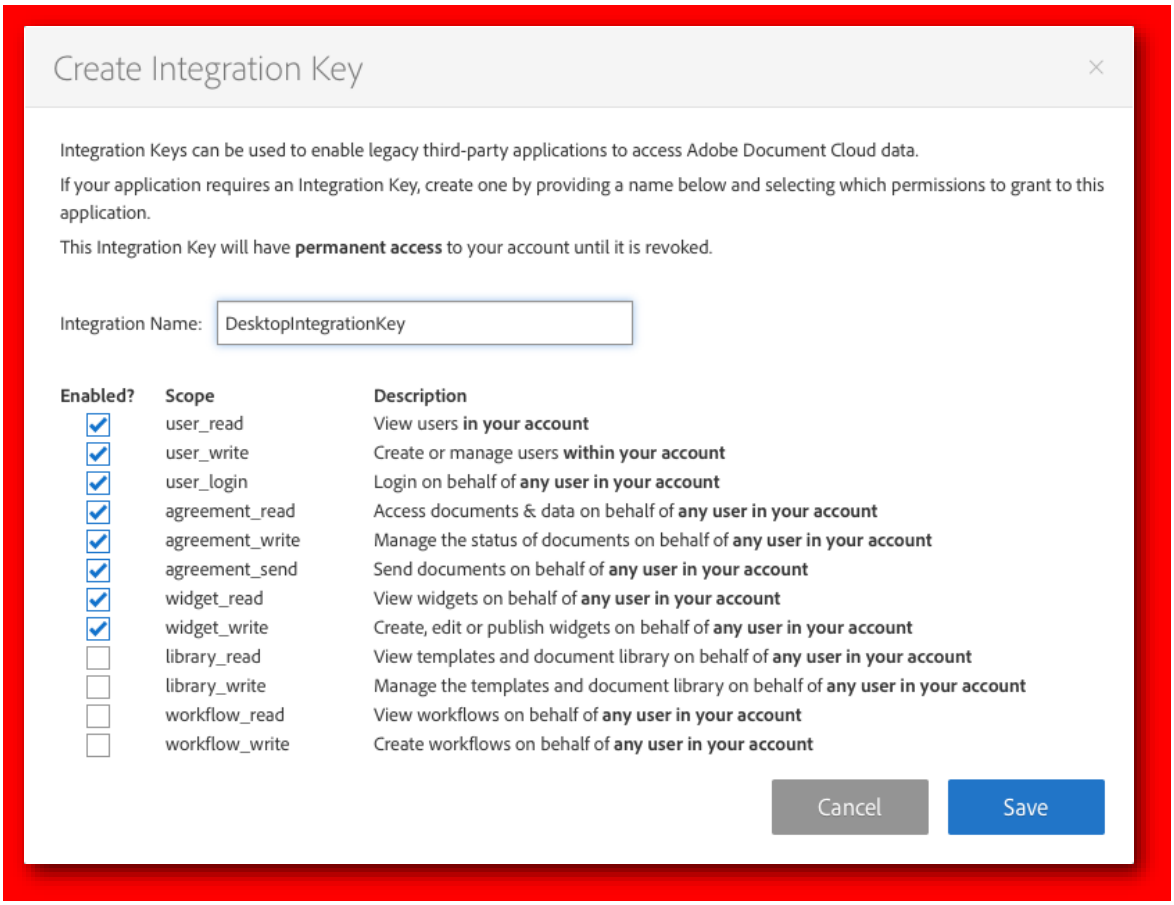


3. In *Create Integration Key* dialog, do the following:

   a. Enter an *Integration Name*. Names must be alphanumeric, but can also include underscores (_). The maximum length is 255 characters. The name should reflect its end purpose. For example, if the Integration Key will be used to integrate with an Office365 or DropBox application, you might use *office365IntegrationKey* or *DropBoxIntegrationKey* respectively. Please note this name. You will need to know it to retrieve the key that is created. (See Retrieving Your Integration Key for more information.)

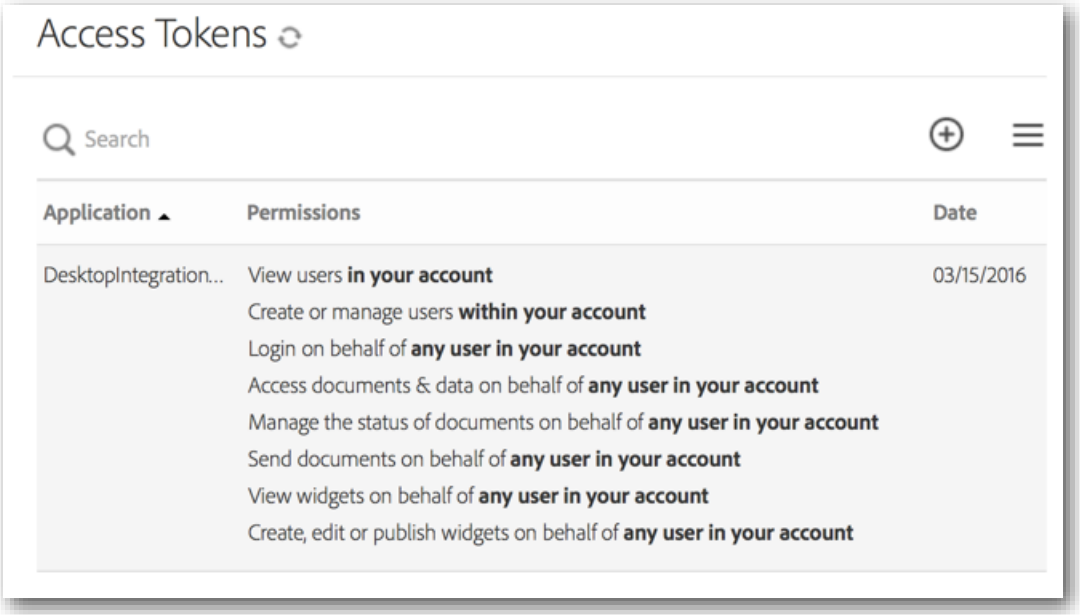
   b. Enable the scope options for your application. To enhance security, enable only those options that are required by your application. (The example below shows one possible set of options.)

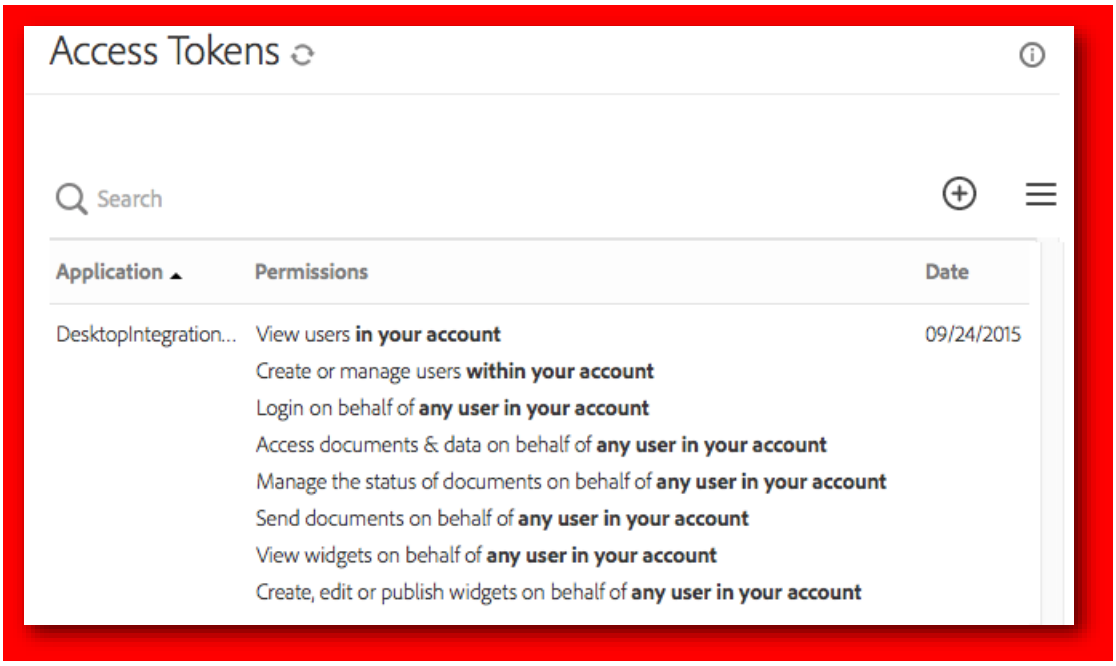



Create Integration Key
Integration Keys can be used to enable legacy third-party applications to access Adobe Document Cloud data.
If your application requires an Integration Key, create one by providing a name below and selecting which permissions to grant to this application.
This Integration Key will have permanent access to your account until it is revoked.
Integration Name:
DesktopIntegrationKey
Enabled?
Scope
Description
user_read
View users in your account
user_write
Create or manage users within your account
user_login
Login on behalf of any user in your account
agreement_read
Access documents & data on behalf of any user in your account
agreement_write
Manage the status of documents on behalf of any user in your account
agreement_send
Send documents on behalf of any user in your account
widget_read
View widgets on behalf of any user in your account
widget_write
Create, edit or publish widgets on behalf of any user in your account
library_read
View templates and document library on behalf of any user in your account
library_write
Manage the templates and document library on behalf of any user in your account
workflow_read
View workflows on behalf of any user in your account
workflow_write
Create workflows on behalf of any user in your account
Cancel
Save

c. Click the **Save button**.

The new Integration Key displays in the *Access Tokens* page.



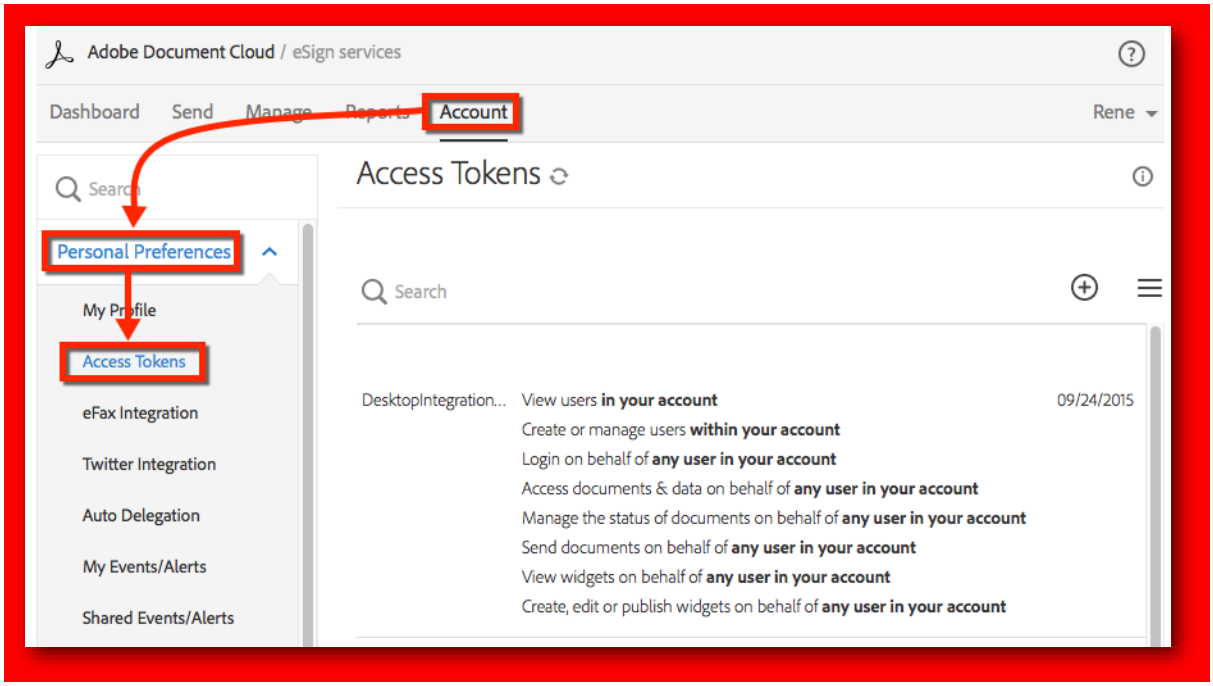


Retrieving Your Integration Key

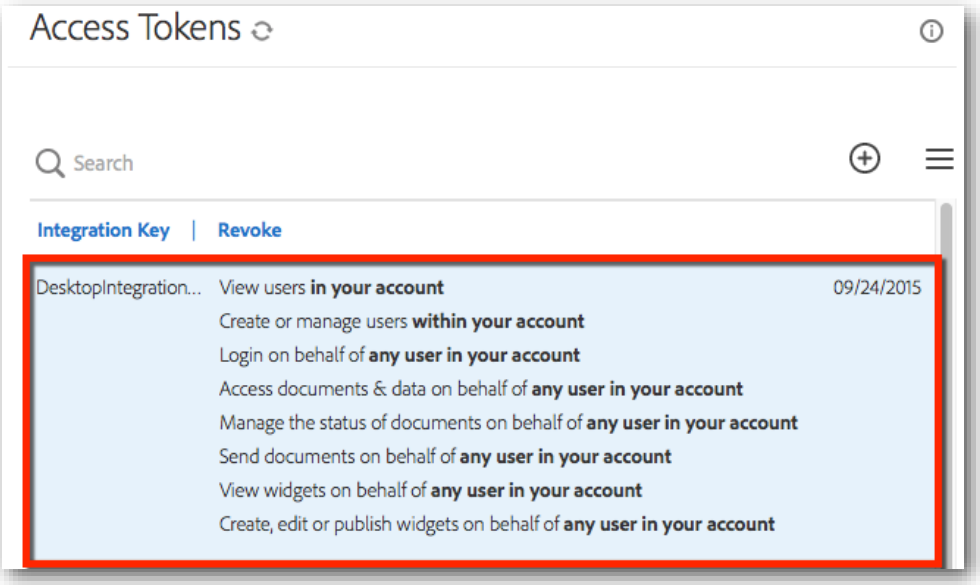
You can retrieve Integration Keys using the Access Tokens page. Only an Account or Group Admin can retrieve an Integration Key.

To retrieve an Integration Key:

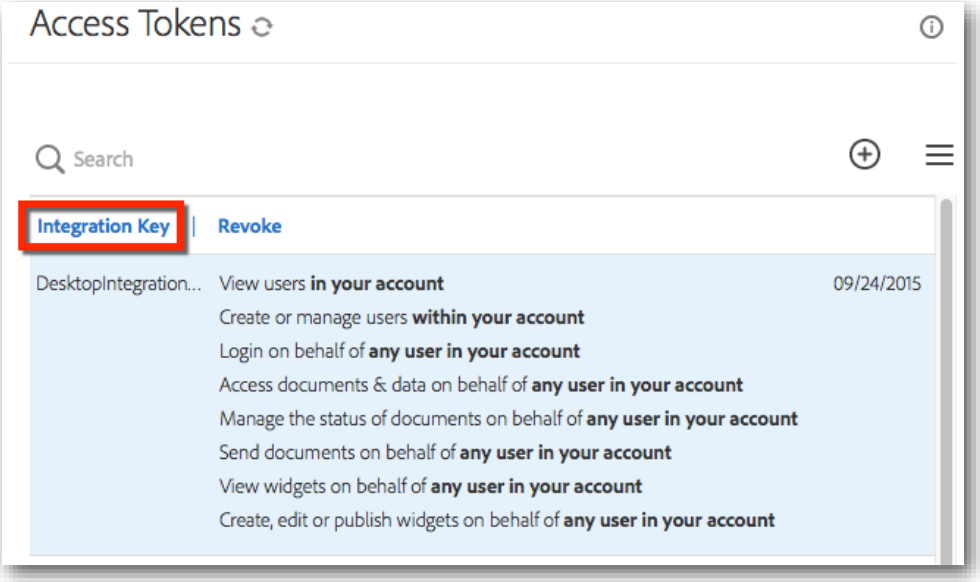
1. Click the **Account** tab, click on **Personal Preferences**, then click **Access Tokens.**



2. In the *Access Tokens* page, click to select the row for your Integration Key access token.



3. Click **Integration Key.**



4. A dialog displaying the Integration Key displays.

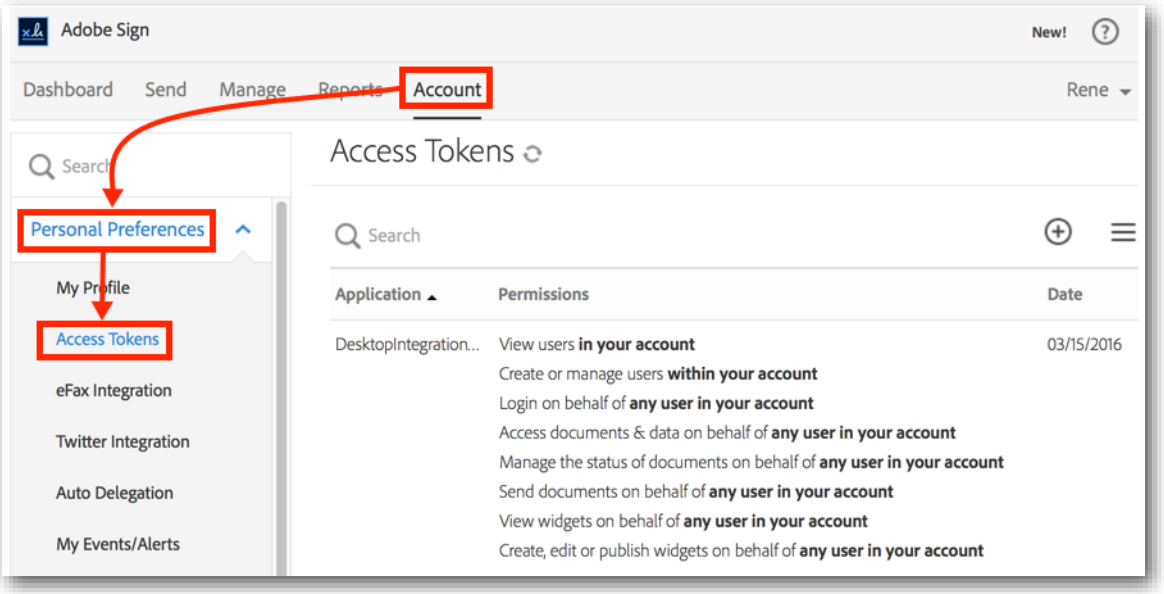
5. Click **OK** to close the dialog.

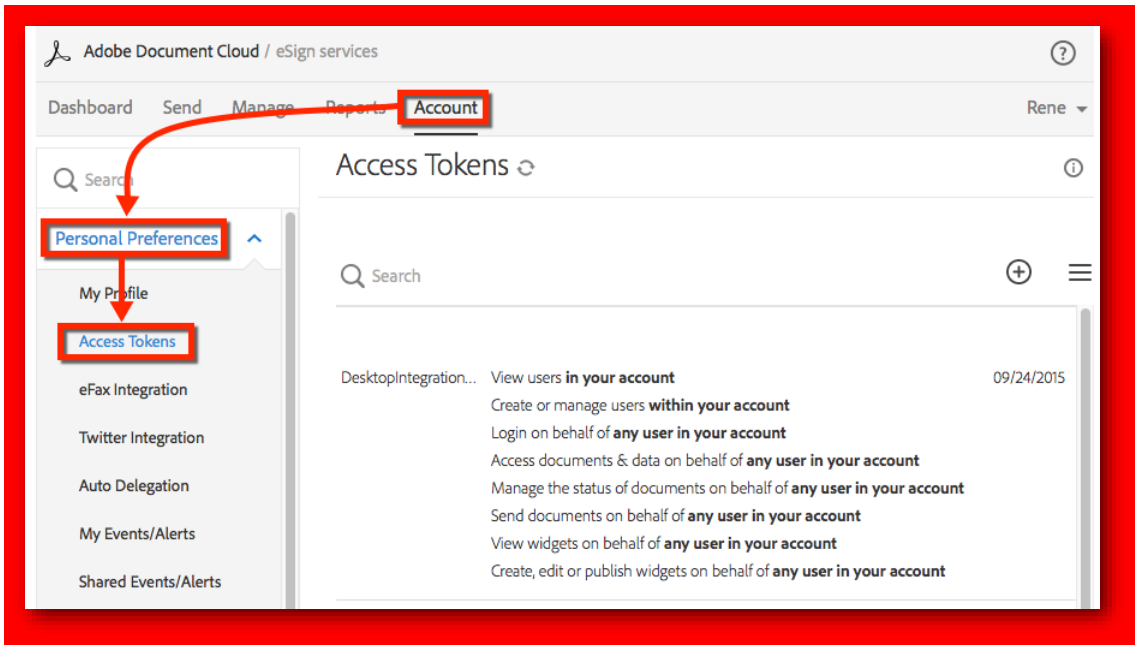
Revoking Access and OAuth Tokens

Access tokens as well as OAuth refresh tokens can be revoked. If an access token is revoked and it has a corresponding refresh token, the refresh token is also revoked. Only an Account or Group Admin can revoke an Access or OAuth Token.

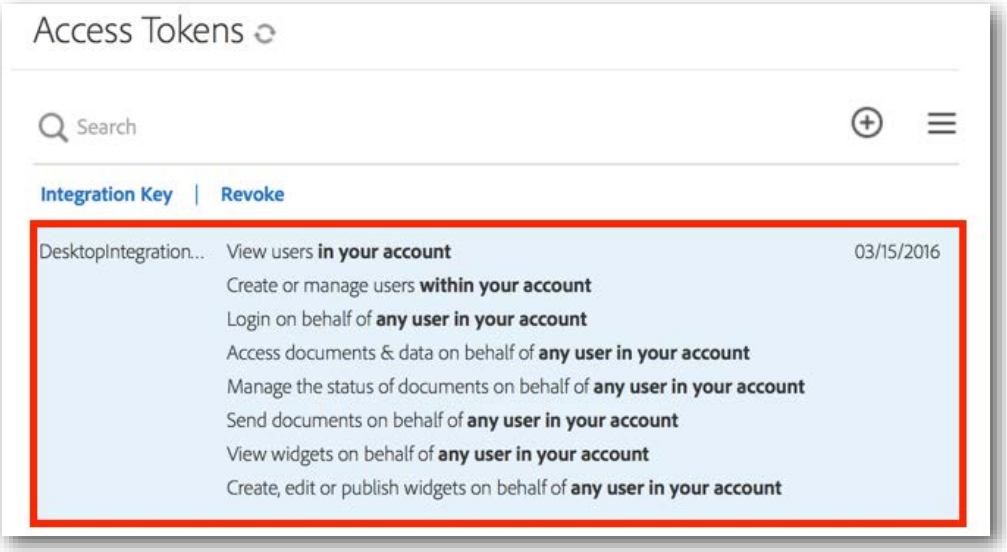
To revoke an Access or OAuth Token:

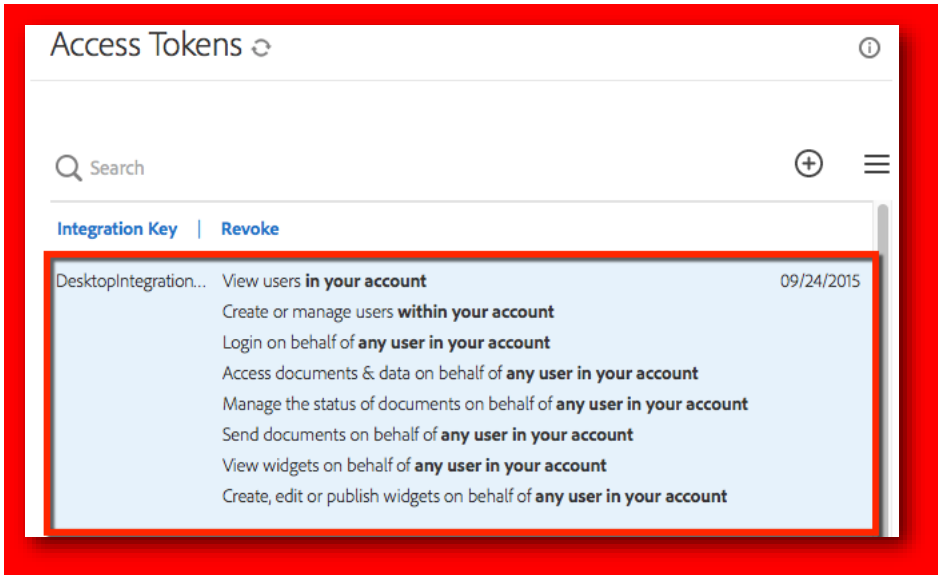
1. Click the **Account** tab, click on **Personal Preferences**, then click **Access Tokens.**



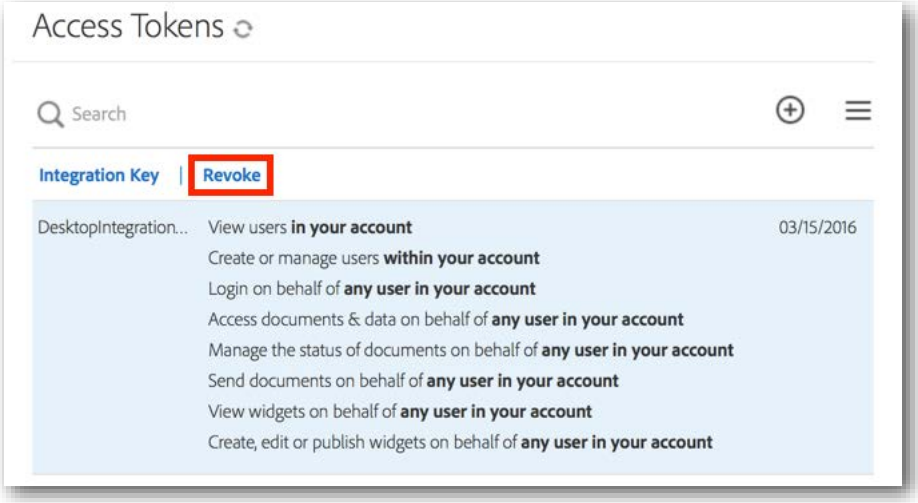


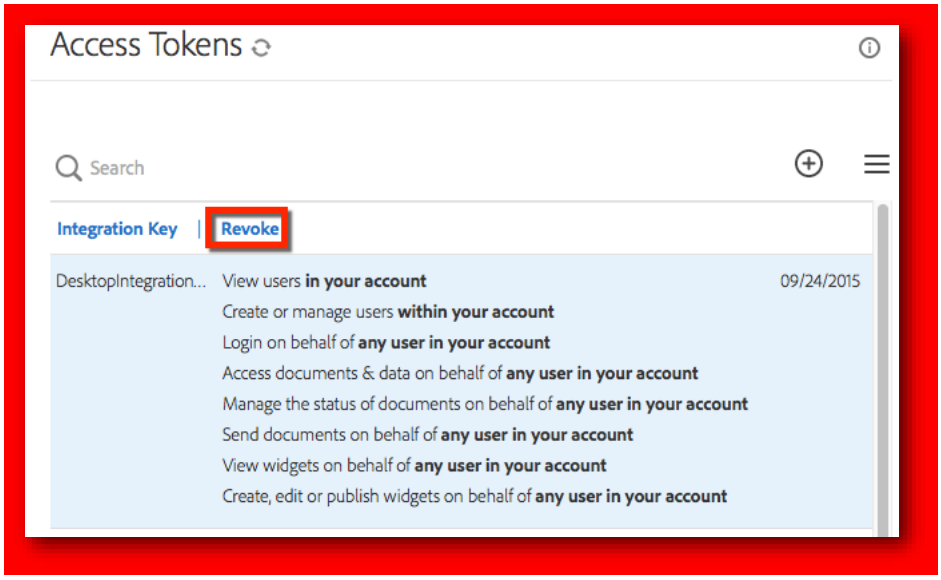
2. Select the row for your Access or OAuth Token.


Access Tokens
Search
Integration Key | Revoke
DesktopIntegration...
View users in your account
03/15/2016
Create or manage users within your account
Login on behalf of any user in your account
Access documents & data on behalf of any user in your account
Manage the status of documents on behalf of any user in your account
Send documents on behalf of any user in your account
View widgets on behalf of any user in your account
Create, edit or publish widgets on behalf of any user in your account

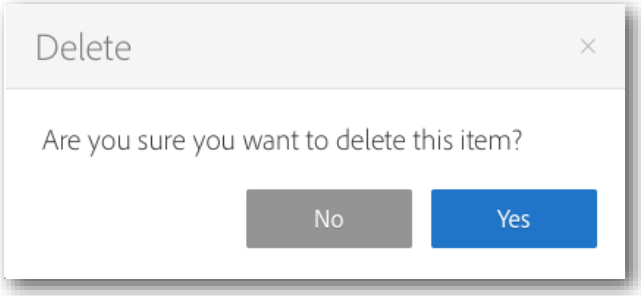

Access Tokens
Search
Integration Key | Revoke
DesktopIntegration...
View users in your account
09/24/2015
Create or manage users within your account
Login on behalf of any user in your account
Access documents & data on behalf of any user in your account
Manage the status of documents on behalf of any user in your account
Send documents on behalf of any user in your account
View widgets on behalf of any user in your account
Create, edit or publish widgets on behalf of any user in your account

3. Click **Revoke**.





4. In the *Delete* dialog, click **Yes**.



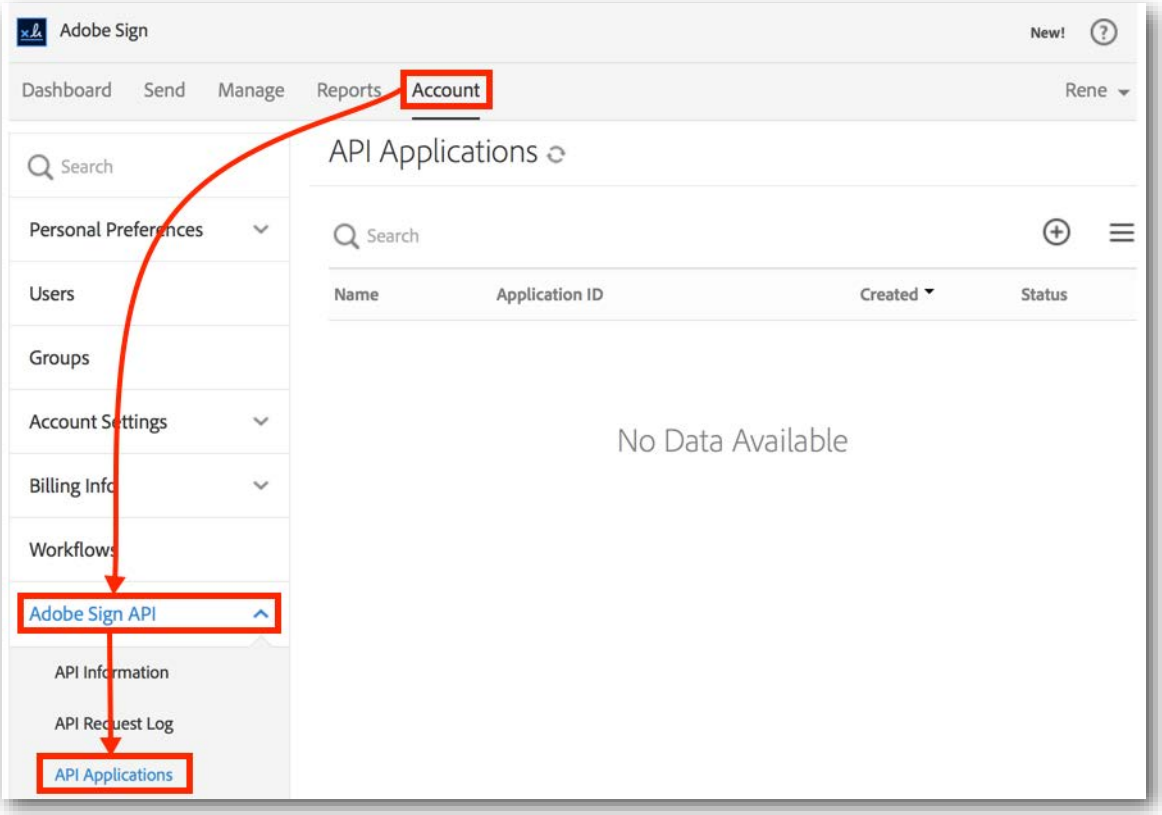
The *Access Token* page redisplays showing that the token has been deleted.

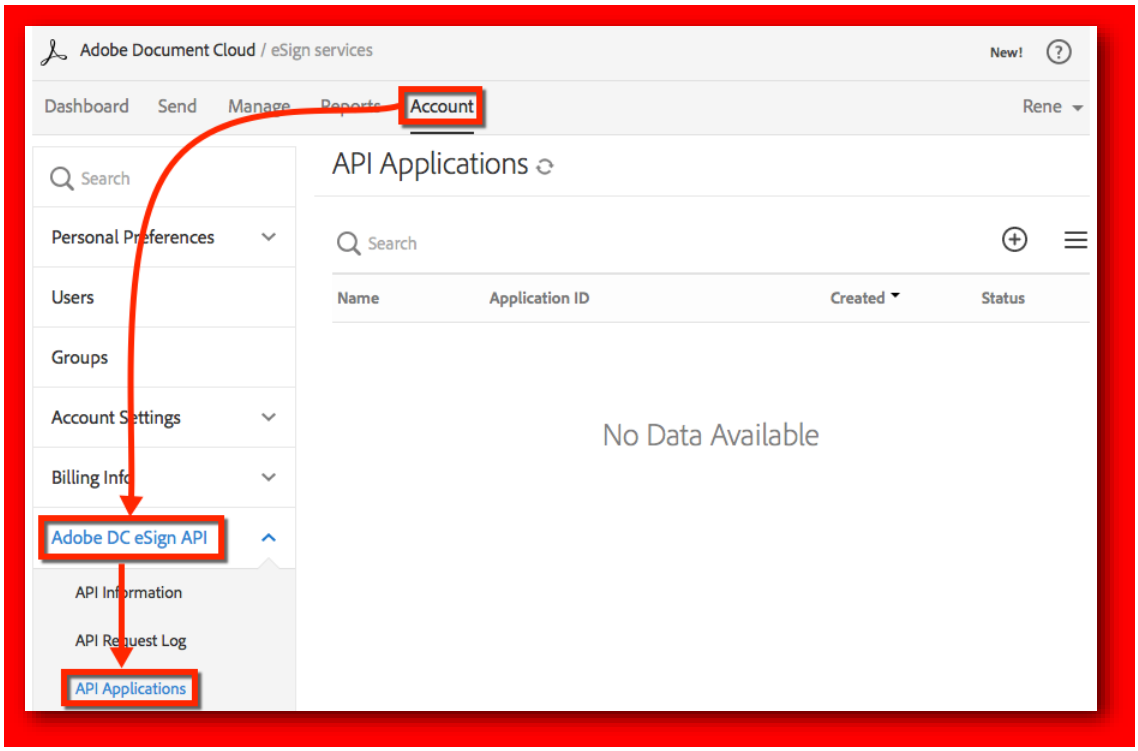
Creating Your Application

You must create your application in Adobe Sign before you can authorize and integrate it. (See also Using OAuth to Access Adobe Sign APIs for more information). Only an Account or Group Admin can create an application.

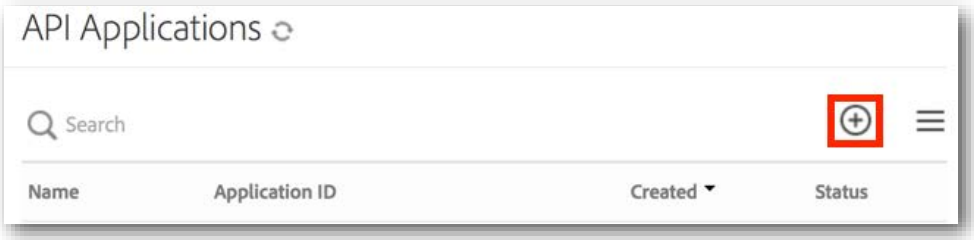
To create an application:

1. Click the **Account** tab, click on **Adobe Sign API**, then click **API Applications.**



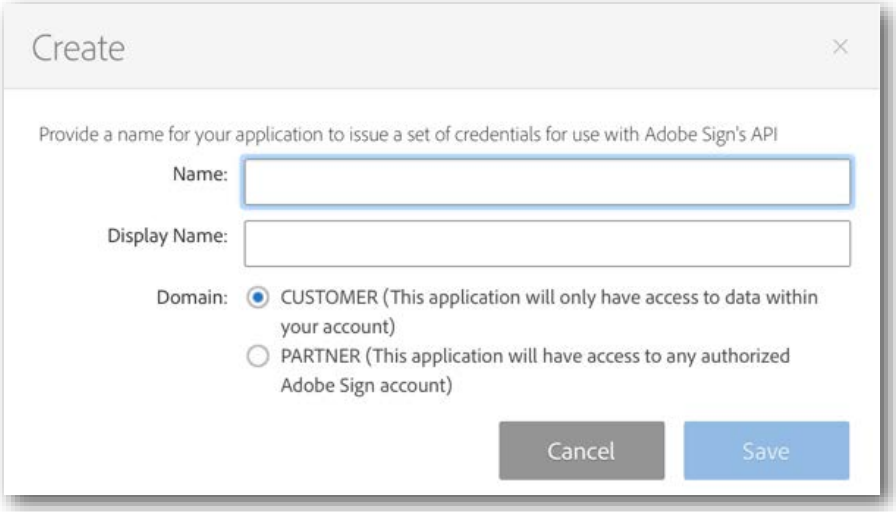


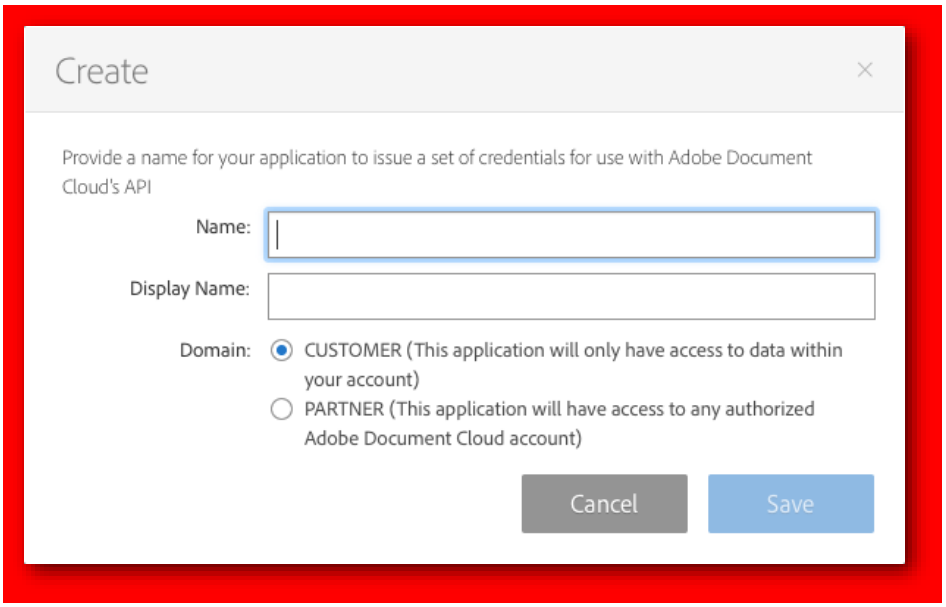
2. Click the Create button (⊕).



3. In the *Create* dialog, do the following:

   a. Enter a *Name* and *Display Name* for your application.





   b. Select one of the following **Domain** options:

      - **Customer**—Specifies that the application can only access data within your account. Select this option you're developing an application for internal use.
      - **Partner**—Specifies that the application can access any authorized Adobe Sign account. Select this option if you're developing an application for other users.

      Note: Partner applications will not have full access to other accounts until they have been Certified. Click here to learn more about Certification.

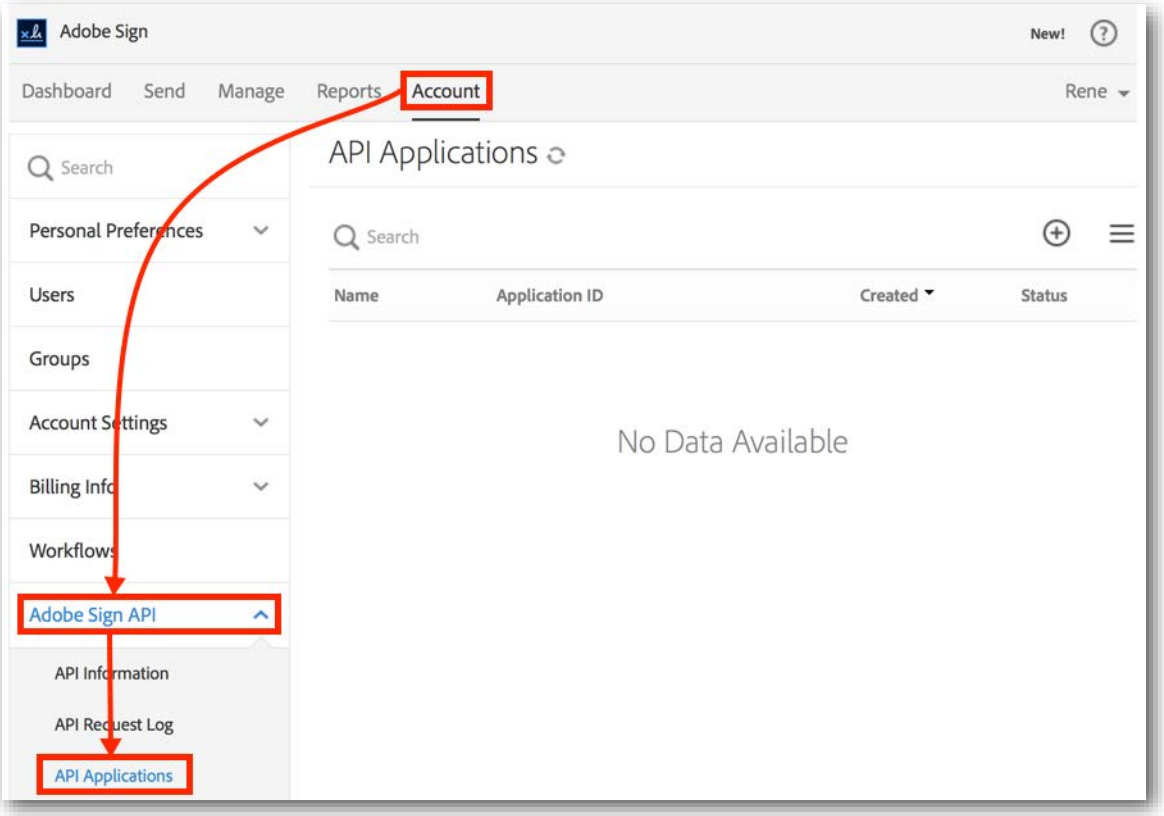
   c. Click **Save**.

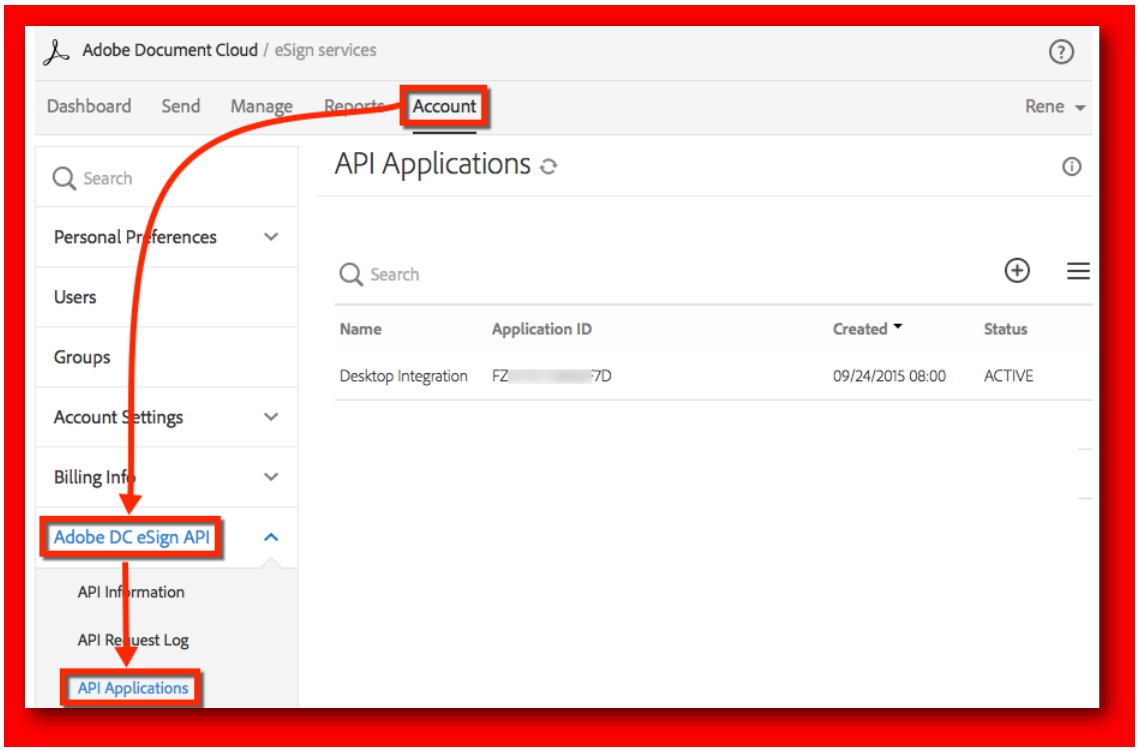
Configuring OAuth for Your Application

If you will be using OAuth with your application, you must configure OAuth. Please note your "Client Id" and "Client Secret". You will need this information to exchange tokens. Before you begin, please identify which Uniform Resource Identifier (URI) or URIs should be used. Please note that the redirectUri specified in your OAuth requests must belong to this list of URIs. You can include multiple uris separated by commas in your list. Only an Account or Group Admin can configure OAuth for an application.

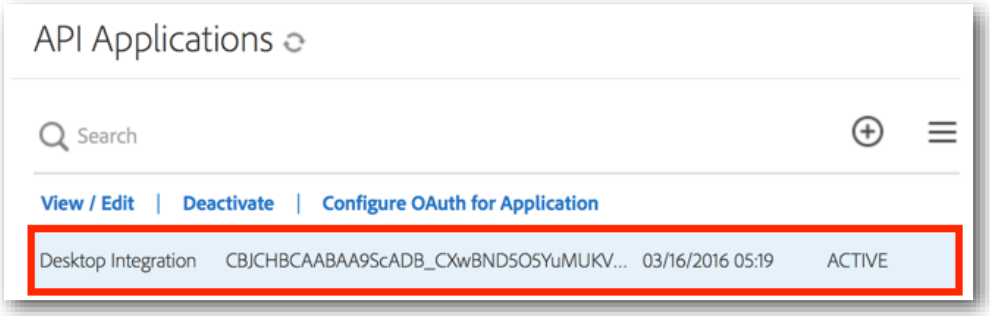
To configure OAuth for your application:

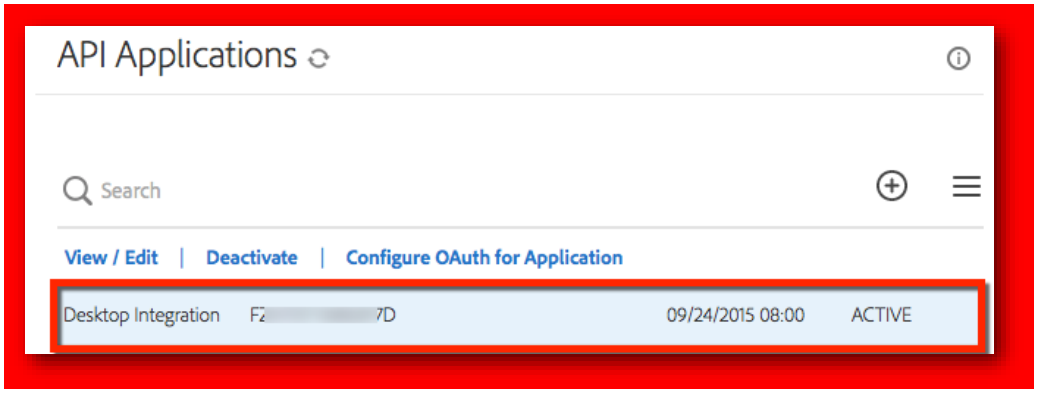
1. Click the **Account** tab, click on **Adobe Sign API**, then click **API Applications.**



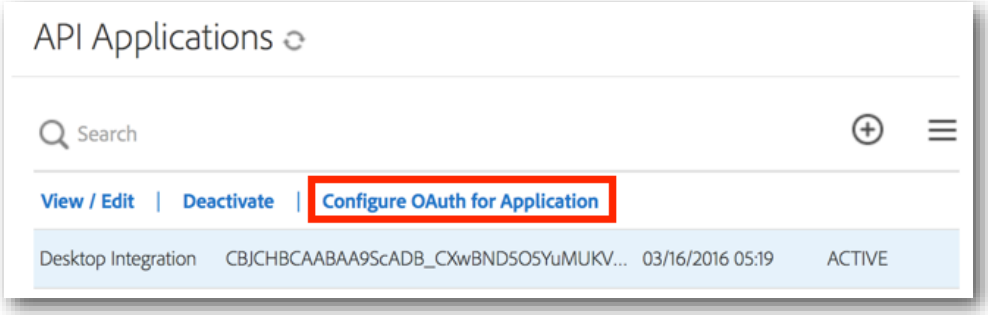


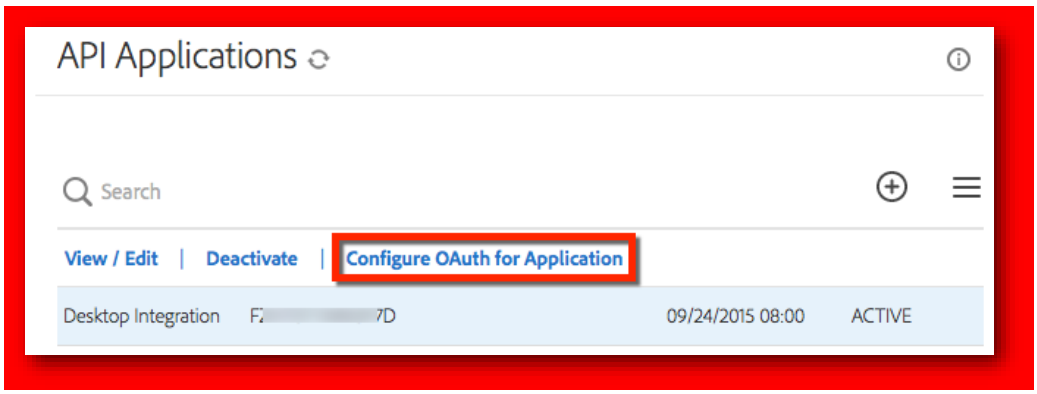
2. On the API Applications page, click to select the application to be configured.



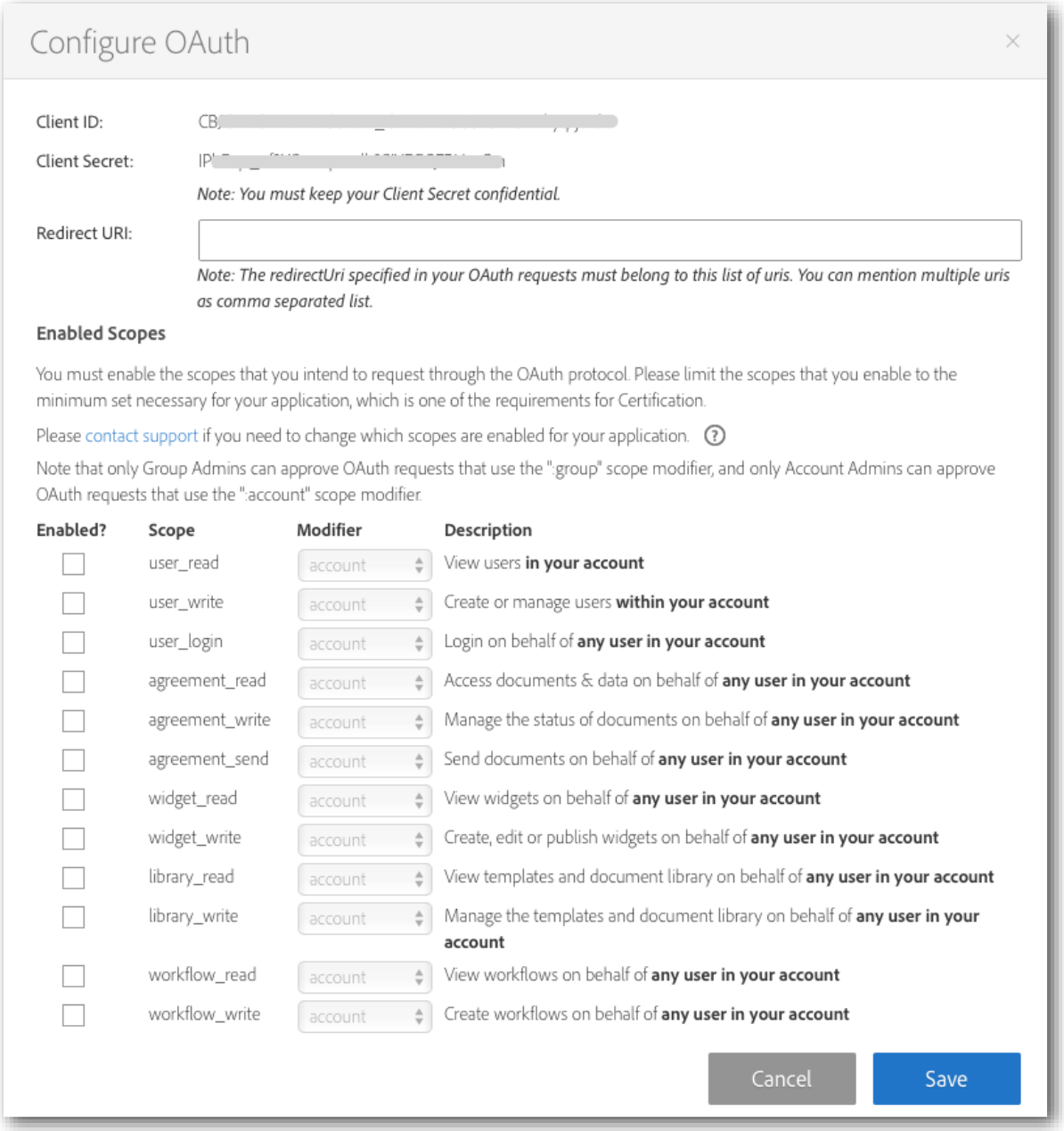


3. Click **Configure OAuth for Application**.





The *Configure OAuth* dialog displays.



Configure OAuth

Client ID: CB[redacted]

Client Secret: IP[redacted]

Note: You must keep your Client Secret confidential.

Redirect URI:

Note: The redirectUri specified in your OAuth requests must belong to this list of uris. You can mention multiple uris as comma separated list.

Enabled Scopes

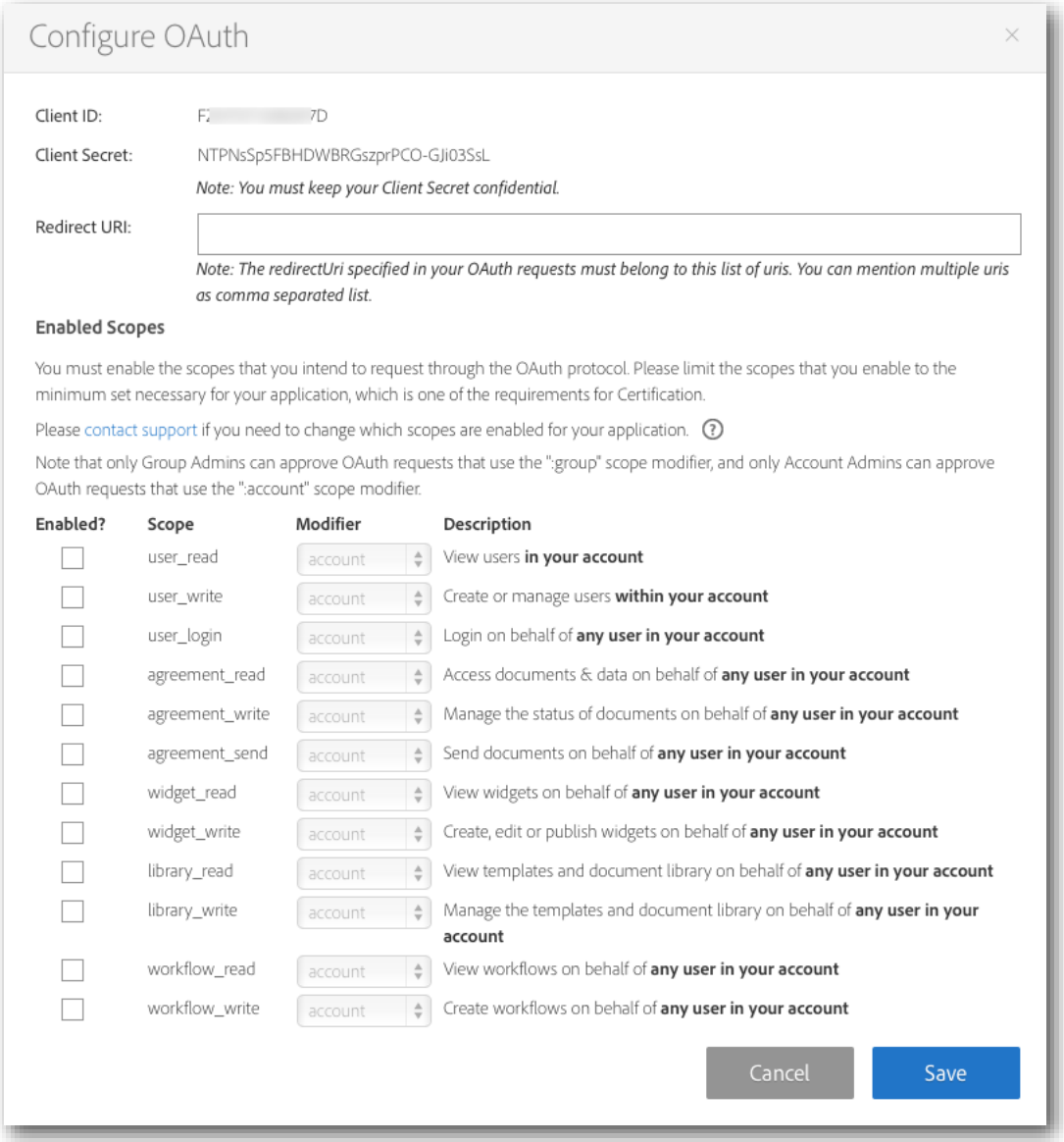
You must enable the scopes that you intend to request through the OAuth protocol. Please limit the scopes that you enable to the minimum set necessary for your application, which is one of the requirements for Certification.

Please contact support if you need to change which scopes are enabled for your application.

Note that only Group Admins can approve OAuth requests that use the ":group" scope modifier, and only Account Admins can approve OAuth requests that use the ":account" scope modifier.

Enabled?	Scope	Modifier	Description
☐	user_read	account	View users **in your account**
☐	user_write	account	Create or manage users **within your account**
☐	user_login	account	Login on behalf of **any user in your account**
☐	agreement_read	account	Access documents & data on behalf of **any user in your account**
☐	agreement_write	account	Manage the status of documents on behalf of **any user in your account**
☐	agreement_send	account	Send documents on behalf of **any user in your account**
☐	widget_read	account	View widgets on behalf of **any user in your account**
☐	widget_write	account	Create, edit or publish widgets on behalf of **any user in your account**
☐	library_read	account	View templates and document library on behalf of **any user in your account**
☐	library_write	account	Manage the templates and document library on behalf of **any user in your account**
☐	workflow_read	account	View workflows on behalf of **any user in your account**
☐	workflow_write	account	Create workflows on behalf of **any user in your account**

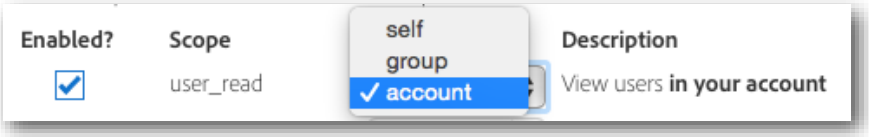
Cancel Save



4. In the *Configure OAuth* dialog, do the following:

   a. Enter a single Redirect URI or multiple URIs separated by commas. The Redirect URI is the page on your website that users will be returned to after the OAuth flow.

   b. Enable each of the required scopes as required then select a modifier (self, group, account) for each.



   c. Click **Save**.

Adobe Sign Scenarios

The following scenarios describe the process of building integrations between applications and Adobe Sign using the REST and SOAP-based APIs.

Scenario 1: Sending & Tracking from an Application with REST-based API

This common scenario involves a 3rd party application (e.g., a CRM system or a document management system) sending document(s) for signature either automatically or due to user-initiated actions. The status of the document and the audit trail need to be exposed in the sending application and when the document is signed, a PDF copy of the signed agreement is retrieved and stored in the application.

This integration scenario can be accomplished as follows:

1. **Sending a document for Signature:** To send a document out for signature through the Adobe Sign REST-based API, you must first call */transientDocuments, POST* to upload the document. This is a multipart request consisting of filename, mime type, and the file stream. The returned transientDocument ID is to be used to refer to the document in the agreement creation call (*/agreements, POST*). The application will specify the recipients and other sending options required for the transaction. The application can specify a callback URL that will be used by Adobe Sign to notify the external application when an event occurs or deliver the signed and completed document to the calling system when the signature process is complete.

   Adobe Sign returns a unique Agreement Id for each request. This Agreement Id can be used to retrieve up-to-date status of the agreement either by polling or when Adobe Sign notifies the calling application of change of status for the document or for retrieving the signed copy of the agreement.

2. **Checking the status of an agreement:** You can get the most current status of an agreement by calling */agreement/{agreementId} GET*. This method takes your OAuth token in the header and Agreement Id as a parameter. Adobe Sign will return the current status of the agreement and a complete history of events that have happened thus far on the particular agreement.

   Adobe Sign supports two mechanisms for an external application to reflect the most current or up-to-date status for an agreement sent for signature. The simplest mechanism is for your application to provide a callback URL when sending the document for signature. Adobe Sign will then ping your service whenever the status of the agreement changes. Upon receiving a callback, your application can then call Adobe Sign to get the latest status on the agreement. The callback URL included in the request must be accessible to Adobe Sign (i.e., must be Internet facing).

   By default, the callback URL is called whenever an event involving a particular transaction occurs in Adobe Sign. The callback includes the Document Key of the agreement whose status has changed, the current status of the agreement, and information on the event that resulted in the callback. Your application logic can evaluate the received status and decide whether to perform an action in the calling system. The callback request looks something like:

   ```
   https://<yourURL>? documentKey=<docKey>&status=<documentStatus>&eventType=<event>
   ```

   In addition to HTTP GET, Adobe Sign also alternatively supports HTTP PUT for receiving events about the signature process, included in the request will be the completed signed PDF. Adobe Sign uses an HTTP

PUT request to return the signed PDF. Please ensure that your application can correctly handle such a request. Please contact Adobe Support or your assigned Client Success Manager to get your account configured to receive HTTP PUT events.

The second mechanism to reflect the most current or up-to-date status of an agreement sent for signature is for your application to periodically poll Adobe Sign regarding the agreement's status. The upside of polling is that it can be used in cases where your calling application is behind your firewall and not accessible from the Internet thus enabling Adobe Sign to complete a callback. The down side of polling is that you have to create a scheduling mechanism within your application to periodically query the status of all documents that were not yet signed, check whether the document's status has changed, and update your system accordingly. If you choose to use polling, we recommend you have different policies based on document "age" In other words, you would reduce the frequency of polling for documents not signed after X days.

3. **Retrieving the signed PDF:** Once an agreement is signed, your application can retrieve the signed copy of the PDF and store that within your application. The signed agreement can be retrieved by calling */agreements/{agreementId}/combinedDocument GET*. This will return a single combined PDF document for the documents associated with the agreement. To retrieve any supporting document, you can call */agreements/{agreementId}/documents GET*. This will return the IDs of all the main and supporting documents of an agreement. The returned document ID can be used in the */agreements/{agreementId}/documents/{documentId} GET* call to retrieve the file stream of a document of the agreement.

   Depending on your application, you can also retrieve the form field data that your signer may have filled in to the document when signing the document by calling */agreements/{agreementId}/formData GET*. The data can be used to update your calling application with the information provided by the signer during signing.

Scenario 2: Sending & Tracking from an Application with SOAP-based API

This common scenario involves a 3rd party application (e.g., a CRM system or a document management system) sending document(s) for signature either automatically or due to user-initiated actions. The status of the document and the audit trail need to be exposed in the sending application and when the document is signed, a PDF copy of the signed agreement is retrieved and stored in the application.

This integration scenario can be accomplished as follows:

1. **Sending a document for Signature:** To send a document out for signature through the Adobe Sign SOAP-based API, call the *sendDocument* method. The application will specify the recipients, files and other sending options required for the transaction. The application will specify a callback URL that will be used by Adobe Sign to notify the external application when an event occurs or deliver the signed and completed document to the calling system when the signature process is complete.

   Adobe Sign returns a unique Document Key for each request. This Document Key can be used to retrieve up-to-date status of the agreement either by polling or when Adobe Sign notifies the calling application of change of status for the document or for retrieving the signed copy of the agreement.

2. **Checking the status of a document:** You can get the most current status of a document by using the *getDocumentInfo* method. This method takes your OAuth token and Document Key parameters. Adobe Sign will return the current status of the agreement and a complete history of events that have happened thus far on the particular document.

   Adobe Sign supports two mechanisms for an external application to reflect the most current or up-to-date status for an agreement sent for signature. The simplest mechanism is for your application to provide a callback URL when sending the document for signature. Adobe Sign will then ping your service whenever the status of the agreement changes. Upon receiving a callback your application can then call Adobe Sign to get the latest status on the agreement. The callback URL included in the request must be accessible to Adobe Sign (i.e., must be Internet facing).

   By default, the callback URL is called whenever an event involving a particular transaction occurs in Adobe Sign. The callback includes the Document Key of the agreement whose status has changed, the current status of the agreement, and information on the event that resulted in the callback. Your application logic can evaluate the received status and decide whether to perform an action in the calling system. The callback request looks something like:

   ```
   https://<yourURL>? documentKey=<docKey>&status=<documentStatus>&eventType=<event>
   ```

   In addition to HTTP GET, Adobe Sign also alternatively supports HTTP POST for receiving events about the signature process, included in the request will be the completed signed PDF. Adobe Sign uses an HTTP POST request to return the signed PDF. Please ensure that your application can correctly handle such a request. Please contact Adobe Support or your assigned Client Success Manager to get your account configured to receive HTTP POST events.

   The second mechanism to reflect the most current or up-to-date status of an agreement sent for signature is for your application to periodically poll Adobe Sign regarding the agreement's status. The upside of polling is that it can be used in cases where your calling application is behind your firewall and not accessible from the Internet thus enabling Adobe Sign to complete a callback. The down side of polling is that you have to create a scheduling mechanism within your application to periodically query the status of all documents that were not yet signed, check whether the document's status has changed, and update your system accordingly. If you choose to use polling, we recommend you have different policies based on document "age". In other words, you would reduce the frequency of polling for documents not signed after X days.

3. **Retrieving the signed PDF:** Once an agreement is signed, your application can retrieve the signed copy of the PDF and store that within your application. The signed agreement can be retrieved using the *getDocuments* method in the API. This API method supports several arguments to allow retrieving the signed documents separately or allow retrieving any supporting documents that the signer may have uploaded during signing, etc.

   Depending on your application, you can also retrieve the form field data that your signer may have filled in to the document when signing the document using the *getFormData* method. The data can be used to update your calling application with the information provided by the signer during signing.

Scenario 3: Embedding Adobe Sign eSigning in An Application

Another common scenario involves an application where users need to sign documents within the application as part of a process. For example, a partner portal for onboarding new partners that requires them to sign an NDA or an e-commerce application that requires users to sign a purchase agreement. In these cases, the document is not sent to recipients for signature, but is presented to them within your application.

For this type of integration, Adobe Sign supports creating a **Widget** through the Adobe Sign APIs. A widget is like a reusable template that can be presented to users multiple times and signed multiple times. Each time a widget gets signed, the signed document becomes a separate instance of the document. A good way to think about the relationship of the widget and the documents signed through it is a parent-child relationship.

A widget can be presented either to an anonymous signer, in which case Adobe Sign can validate the signer's identity as part of the signing process, or to a signer whose identity can be specified through the API by the hosting application.

For a simple example using Widgets, go to http://www.formerator.com.

The **Widget** integration scenario can be accomplished as follows using the Adobe Sign REST-based API:

1. **Creating a Widget:** To create a widget through the API, you must first call */transientDocuments*, *POST* to upload the document. This is a multipart request consisting of filename, mime type, and the file stream. The returned transientDocument ID is to be used to refer to the document in the widget creation call *(/widgets, POST*). The API end-point, in addition to the widget key, returns an embed-code, which can be used for embedding the widget within your application as well as a URL at which the widget gets hosted. The URL can be posted within your application for users to navigate to for signing a document. If the identity of the person signing the widget is known a priori, the widget can also be personalized with the signer's information using the provided personalization method *PUT /widgets/{widgetId}/personalize*.

   When creating the widget your application may also specify the address of the Web page that users will be redirected to when they successfully complete signing the widget.

2. **Checking the status of documents signed through a widget:** As mentioned earlier, each time a widget is signed a separate instance of a document gets created.

   To get the agreements created using the widget, call */widgets/{widgetID}/agreements GET* where *widgetID* is the key returned by the service while creating the widget.

   To retrieve the data filled by the users at the time of signing the widget, call */widgets/{widgetID}/formData GET*. The service returns data in comma-separated value (CSV) file format.

   The first line includes the column header names and each row represents a distinct instance of the widget. The document keys of all child widgets will be in the first column, under "EchoSign transaction number". See example below:

```
EchoSign transaction number, Agreement name, signed, email
12ABC3D456E7F,test widget,2/5/10 09:21,email@domain.com
98ZYX7W654V3U,test widget,2/6/10 11:56, email2@domain.com
```

If the child document is signed by two signers, there will be two rows in the CSV with the same document key. See example below:

```
EchoSign transaction number, Agreement name, signed, email
12ABC3D456E7F,test widget,2/5/10 09:21,email@domain.com
98ZYX7W654V3U,test widget,2/6/10 11:56,email2@domain.com
12ABC3D456E7F,test widget,2/6/10 13:37,email3@domain.com
```

Exposing Additional Adobe Sign Actions

In addition to sending documents for signature and tracking the status of the document, your application can also expose additional actions to its users allowing them to cancel an agreement when it's still out for signature or send a reminder to the current signer while the document is waiting for signature. These additional actions allow users to interact with the Adobe Sign functionality entirely from within your application.

See the Getting Started document for a list of available documentation resources.

Adobe Sign Events

The functionality of Adobe Sign can be incorporated into external applications by directly embedding the Adobe Sign user interface (UI) within these applications. Adobe Sign also supports sending events (status updates) to the third party application pages so that the external application is aware of the actions that the user is performing with the Adobe Sign UI. These events are passed between the controller window and a receiver window running on different domains for event communication. This section provides a guide to all the events supported by Adobe Sign.

Event System Requirements

Using the event framework within Adobe Sign requires the user of a browser, which supports postMessage. See https://developer.mozilla.org/en-US/docs/Web/API/Window.postMessage for support browsers.

List of Supported Events

The following table lists the Adobe Sign supported UI events that can be embedded and presented within the UI of external applications.

Event Type	Data	Description
WORKFLOW EVENTS		
'ESIGN'	NONE	This event gets fired after a user has successfully signed an agreement.
'REJECT'	NONE	This event is fired after a user rejects an agreement.
'PREFILL'	NONE	This event is fired after a user completes prefilling an agreement and sends it.
PAGE LOAD EVENTS		
'PAGE_LOAD'	pageName: 'POST_SEND' apiAgreementId: '<agreement capability>'	This event fires when an agreement has been successfully sent and the post send page has been loaded.
'PAGE_LOAD'	pageName: 'DIGSIG_DOWNLOAD'	This is a special event that is fired for documents requiring Digital Signatures. This event fires when a user has completed all the required fields in a document and page to download the document for Digital Signature gets loaded.
'PAGE_LOAD'	pageName: 'AUTHORING'	This event fires when the form-field authoring page loads for an agreement.
'PAGE_LOAD'	pageName: 'DELEGATION'	This event fires when the page from which an agreement can be delegated gets loaded. The

Event Type	Data	Description
		loading of the page does guarantee that delegation has or will actually occur.
'PAGE_LOAD'	pageName: 'MANAGE'	This event fires when the manage page loads.
'PAGE_LOAD'	pageName: 'LOGIN'	This event fires when the login page loads.
SESSION EVENTS		
'SESSION_TIMEOUT'	message:'PRE_SESSION_TIMEOUT' warningTimeMinutes: <float> expirationTimeMinutes: <float>	This event is triggered two seconds before session timeout dialogue is displayed to the user. The UI shows "Your session is about to expire" message to the user. The warningTimeMinutes and expirationTimeMinutes values correspond to the warning & session timeout times in minutes.
'SESSION_TIMEOUT'	message:'POST_SESSION_TIMEOUT' warningTimeMinutes: <float> expirationTimeMinutes: <float>	This event is triggered when the user's session times out and the user is notified.
'ERROR'	message: <varies>	This event fires when an error dialog or an error page is displayed to the user. System Error: 500, 503 is returned General user error message: document processing or conversion errors
USER ACTION EVENTS		
'CLICK'	pageName: 'POST_SEND' or 'POST_SIGN' target: 'MANAGE_LINK' url: '<full URL with agreementId>' apiAgreementId: '<agreement capability>'	This event fires when the user clicks on the "Manage this document" button in the post-send page. The URL data contains the full URL needed to bring up the manage page with the particular agreement selected. The apiAgreementId is the DocumentKey, used by client application making the API calls.
'CLICK'	pageName: 'POST_SEND' or 'POST_SIGN' target: 'SEND_ANOTHER_LINK'	This event fires when the user clicks on the "Send another document" button.

Using Adobe Sign Events

Adding an Event Handler on the Parent Page

In order to use the events fired by Adobe Sign, the external application should include a callback handler in the parent page that embeds the Adobe Sign application UI.

Below is an example event handler that can be placed in the parent page:

```
function eventHandler(e) {
   if (e.origin == "https://secure.echosign.com") {
      console.log("Event from EchoSign!", JSON.parse(e.data));   }
   else {
      console.log("Do not process this!");
   }
}
if (!window.addEventListener) {
   window.attachEvent('onmessage', eventHandler);
}
else {
   window.addEventListener('message', eventHandler, false);
}
```

Embedding the Adobe Sign UI in an iFrame

Adobe Sign provides APIs that allow embedding the Adobe Sign UI into an external application. The API call returns a URL which can directly be embedded into a child iFrame of the parent which includes the event listener.

```
<script type='text/javascript' language='JavaScript' src='ECHOSIGN URL'></script>
```