ADOBE® ECHOSIGN®

EchoSign API Guide

Version 2.0

Last Updated: April, 2013

Table of Contents

Overview

Adobe EchoSign's eSignature platform allows developers to securely send documents for signature, embed the Adobe EchoSign signing experience into their website or application, track document status in real time and retrieve signed documents and all related data from EchoSign. Developers can also use Adobe EchoSign's platform to manage accounts, users and documents.

The Adobe EchoSign eSignature Platform utilizes Web Services API for communication with external systems. Web Services is a standard based, secure and scalable method of establishing communications between systems over the Web.

The Adobe EchoSign API requires every developer to have a unique API key allowing them to integrate their solution to EchoSign while applying EchoSign's permission model to the data they can create or access.

This document provides an overview of how to use the EchoSign API in common integration scenarios. The EchoSign APIs are SOAP-based Web Services that allow applications to communicate with the EchoSign service for building integrations between these external applications and EchoSign.

Getting Started

Developers can sign up online for a free unlimited Adobe EchoSign Developer account at https://secure.echosign.com/public/upgrade?type=developer. As part of the registration process developers will receive an API key allowing them to use the eSignature Platform along with code samples and other resources. Note the API Key is **CASE SENSITIVE** and should be used exactly in the form provided by EchoSign within your integration.

Administrators for existing Enterprise or Global EchoSign account can get an API Key for their account by going to their EchoSign API section under the Account tab through the Web Browser.

The API Key is specific to your account and should be kept a secret.

Key developer resources:

- EchoSign API WSDL: https://secure.echosign.com/redirect/latestApiWsdl
- EchoSign API Documentation: https://secure.echosign.com/redirect/latestApiMethods
- EchoSign API Development SDK: https://secure.echosign.com/redirect/latestApiDevelopersKit

Working with the EchoSign APIs

The EchoSign APIs can be used to build a variety of different integrations from external applications. These integrations can enable users working in an external application to initiate transactions within EchoSign to get documents signed entirely from within the external application. The external application can also receive status updates for initiated transactions and retrieve a copy of the signed agreements to storage in the external system.

Lets look at a couple of integration scenarios between an external system and EchoSign and walk through the details of building the integration through the API.

Scenario 1: Sending & tracking from an external application

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 detailed audit trail need to be exposed into the sending application and when the document is signed, a PDF copy of the signed agreement is retrieved and stored in the application.

This high-level integration scenario can be accomplished as follows:

1. **Sending a document for Signature**: To send a document out for signature through the EchoSign 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 EchoSign 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.

   EchoSign 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 EchoSign 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 API Key and Document Key parameters. EchoSign will return the current status of the agreement and a complete history of events that have happened thus far on the particular document.

   EchoSign 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, EchoSign will then ping your service whenever the status of the agreement changes. Upon receiving a callback your application can then call EchoSign to get the latest status of the agreement. The callback URL included in the request must be accessible to EchoSign (i.e. must be Internet facing).

   By default the callback URL gets called when the signature process completed and included in the request will be the completed signed PDF. EchoSign uses an HTTP POST request to return the signed PDF, please ensure that your application can correctly handle such request. Your EchoSign account can also be configured such that the service can ping your application for every change in the status of the agreement. The callback will include the Document Key of the agreement whose status has changed and the current status. Your application logic can evaluate the received status and decide whether to perform an action in the calling system. Please contact EchoSign support or your assigned Client Success representative to get your account configured to receive ping at every document event.

The second mechanism is for your application to check the status of the agreement by periodically polling the service and checking the status of the agreement. The upside of polling is that it can be used in case your calling application is behind your firewall and not accessible from the Internet for EchoSign to callback. On the flip side, to use polling 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. If you choose to use polling, we recommend you have different policies based on document "age" - ie. reducing the frequency of polling if the document was 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 2: Embedding EchoSign signing in another application

Another common scenario involves an application where users need to sign documents within the application as part of a process. For e.g. a partner portal for onboarding new partners requiring them to sign an NDA or an online ecommerce application requiring users to sign a purchase agreement, etc. In this case the document is not sent for the recipient for signing but is presented to them within your application.

For this type of integration EchoSign supports creating a Widget through the API. 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 EchoSign 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.

This integration scenario can be accomplished as follows:

1. **Creating a Widget**: EchoSign APIs provide a mechanism for creating Widgets. EchoSign API provides several options for creating Widgets: *createEmbeddedWidget* allows you to create a widget and receive an embed-code which can be used for embedding within your application, *createUrlWidget* allows you to create a widget and receive 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 methods.

   When creating the widget your application can specify the files and other options. The 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, this separate instance has its own Document Key i.e. child Document Key distinct from the Document Key of the parent widget.

   There are two ways you can find the child Document Keys: by using the *getFormData* method, or by using the completion URL.
   a. Finding the child Document Keys using the *getFormData* method: Your application can call the *getFormData* method and pass it the Document Key of the parent widget. The output will include the data in comma-separated value (CSV) 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
```

   b. Finding the child Document Keys using the completion URL: When setting up the Widget you can tell EchoSign where to send the user after they've completed signing the widget. The child Document Key will be appended as a parameter to that URL if the sender is the same as the API key user.

This child Document Key, you can call the *getDocumentInfo* method to get the latest status of the document and retrieve the signed copy of the agreement using the *getDocuments* method.

Exposing additional EchoSign 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 EchoSign functionality entirely from within your application.