

Nintex – DocuSign

Custom connector for sending document to a group.

Introduction

The documentation has been compiled and written according to my previous week research and also in such a long time I also implemented the things in practice as practicing actually make us know what we are working on.

DocuSign

It is a web based service using for sending and signing documents contained in envelopes. Besides providing us with user friendly web app (such as the sandbox) for sending envelopes, it also provides a **Rest API** for developers to integrate their app with DocuSign. This way we will create a nintex connector (can be thought of a web app) and make a call to DocuSign rest API, passing description of envelope (including group ID) in the body of http request.

Nintex Cloud Platform

This is again a web app for creating workflow and store and run it in cloud. It's the latest platform from nintex. Nintex supports sending document to single or multiple recipients through already provided DocuSign connector.

Nintex Connector [link](#)

Nintex Connector will require an open-API specification file (in json format) which can completely define a rest API. For example it defines operations (get, post etc. to specific path in a rest API is actually an operation). Since we are building a server less integration we try to specify most of things through open-API json file. So open-API spec. will allow nintex to understand the API very well including type and structure of requests and responses and thus allow it to use operations (actions) in workflow which it actually requires. I have predicted to use a minimum of two actions in *standard form*.

Oauth 2.0

Oauth 2.0 is an authorization protocol that gives an API client limited access to user data on a web server. For example you see when you have to sign up on a website or an app there is option to sign up using Google or Facebook or twitter etc. Same here the DocuSign will require user's consent before providing token to the nintex connector to connect to DocuSign rest API. Both Nintex and DocuSign follow Oauth 2.0 pattern and this way it's become easy to integrate authentication.

DocuSign Oauth 2.0 Specification

Referring to guide at <https://developers.docusign.com/esign-rest-api/guides/authentication>

Prerequisites

Your application has an Integration Key.

To generate an Integration key:

1. Log in to your DocuSign admin account

2. Under Integrations, select API and Keys
3. Under My Apps / Integration Keys, select Add App / Integration Key, then give it a name
4. Create a secret key.
5. No need to add redirect URI right now. We will add it later.
6. Copy the **secret key** and **Integration Key**.

Step 1: Create the connector.

1. Go to your nintex workflow cloud. For example mine is:
<https://inxane-149075.workflowcloud.com/>
2. Go to dashboard and from left navigation menu select Xtensions.
3. Click the plus button on right in Private connector list panel.
4. Add the open-API json file (we are just going to create in few moment).
5. The next “Configure security” step has actually determined that we are using Oauth 2.0 for authentication i.e this step will help to gain token which will be valid for a long time. In open-API json file we mentioned the Authentication Type as Oauth 2.0 and nintex determined it. Add **secret key (Client secret)** and **Integration Key (Client ID)**. Then copy the **Redirect URL**. Add this to integration key settings we went through earlier in **Prerequisites** section above. The other information had been supplied by open-API json file.
6. At the last select a description and icon for connector. The Name is supplied by open-API json file and you can still edit it. Click **publish**.

Private connector list



Add new connector


Configure security

Publish

Connector definition

Create your own connector in an OpenAPI specification. Upload or provide a URL to your specification to transform it into a connector.

OpenAPI specification file *

 Choose a file

Or

OpenAPI specification URL *

Next

[Add new connector](#) > [Configure security](#) > [Publish](#)

Security

Confirm the authentication type of your connector and complete any required fields.

Security *

Generic OAuth2

Redirect URL

https://us.nintex.io/connection/api/Token

Client ID *

Client secret *

Authorization URL *

https://account-d.docusign.com/oauth/auth

Token URL *

https://account-d.docusign.com/oauth/token

Publish

Enter a name, description and choose an icon for your connector.

Name *

Nintex-Docusign Connector

Description *

Select an icon *

Or

Choose a file

Min size 48 x 48 px

Previous

Publish

Step 2: Create the open-API specification.

1. Go to <https://editor.swagger.io>
2. Create the open-API json file. Also one can visualize on right panel a user friendly description.

Before creating the specification lets understand the things we actually want in it.

1. We need a server (account-d.docusign.com) and base URL (/ or /oauth/). The server for our need can be any of **account-d.docusign.com (dev.)** or **account.docusign.com (prod.)**.

2. We need to define and configure authentication mechanism (Oauth 2.0).
3. We need to define paths and associated methods (get, post etc.) to form operations such as **/userinfo (get)**

Paths are relative to server + base URL so the absolute path will be:

`account-d.docusign.com/oauth/userinfo (get)`

This can be thought of an operation such as **Get User Info**.

4. Define request and response structures for operations. They will be recognized by nintex when we add the operation as an action in workflow and configure it.

A talk about authentication:

1. We will create a connection so that user's DocuSign account recognizes our app and grant it permission through user consent. This has to be done once. Connection will be stored and used for every operation we specify as to have authentication token to be included. We mention this in open-API spec. in each operation we want to.
2. The authentication and token management will all be done by nintex.

First proposed theory (standard one):

1. Now we need to know the base URL and account ID, form the URL and actually make request for example sending envelope.

`{base_uri} + "/restapi/v2/accounts/" + {account_id}`

For example to send an envelope we have to :

Authorization: Bearer eyJ0eX...MrhIddzBAQ

POST

`https://demo.docusign.net/restapi/v2/accounts/fe0a3-3b9b-46d4b7ab/envelopes`

```
{
  "status": "sent",
  "emailSubject": "Sent from the DocuSign REST API",
  "documents": [{
    "documentId": "1",
    "name": "contract.pdf",
    "documentBase64": "base64 document bytes...",
  }],
  "recipients": {
    "signers": [{
      "email": "jane.dough@example.com",
      "name": "Jane Dough",
      "recipientId": "1",
      "routingOrder": "1",
    }]
  }
}
```

2. So we add one operation for retrieving user info (json response as shown below) and obtain accounts information as a **collection** in nintex. The accounts property in json response is an array and we mention it in response structure in open API json file but it is mapped to collection in nintex. Then we **loop** through accounts in collection and store each account temporarily in text variable say **temp** and set a variable say **flag** to false initially. Based on condition that which account is default one we break the loop by setting the **flag** to true. This request could be made as:

Authorization: Bearer eyJ0eX...MrhIddzBAQ

GET https://account-d.docusign.com/oauth/userinfo (dev.)

Or

GET https://account.docusign.com/oauth/userinfo (prod.)

The json response would be

```
{
  ...
  ...
  "accounts": [
    {
      "account_id": "51f188fa-364d-4e9d-8d92-1cc0ee3db3df",
      "is_default": true,
      "account_name": "Geminid Systems",
      "base_uri": "https://demo.docusign.net",
      "organization": {
        "organization_id": "527aad24-2c80-4edf-9aa4-1cff3ee3e3a7",
        "links": [
          {
            "rel": "self",
            "href": "https://account-d.docusign.com/organizations/521cff"
          }
        ]
      }
    },
    {
      "account_id": "4210c64d-3ac4-47d5-8961-0b35ab7652c7",
      "is_default": false,
      "account_name": "Portola",
      "base_uri": "https://demo.docusign.net"
    }
  ]
}
```

The accounts node in the response is an array of account information entries. It is not uncommon for a user to be a member of multiple accounts. Depending on your use case, it may be best to use the default account, or your integration may enable the user to choose the account. The authenticated user's default account has the is_default property set to true. User can set the default account through which he wants to send envelope.

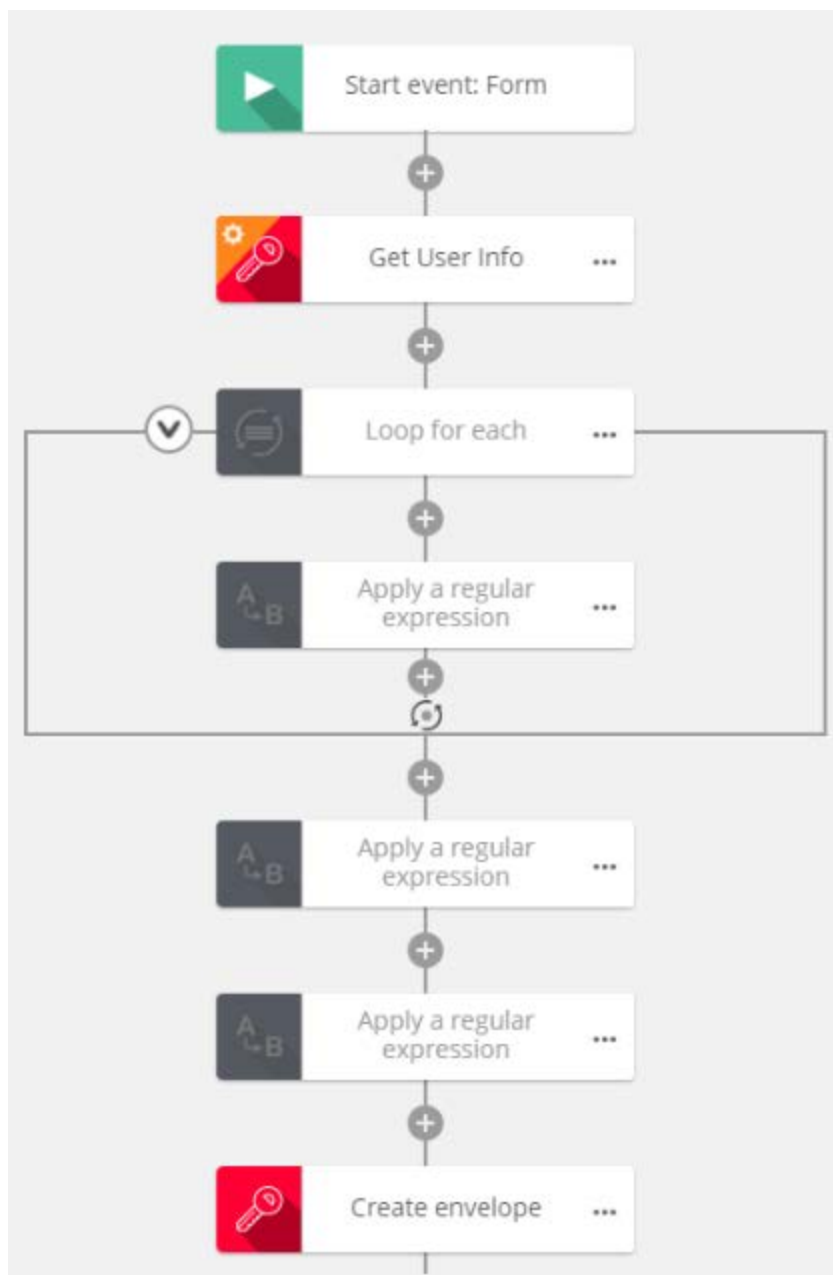
<https://support.docusign.com/en/guides/org-admin-guide-default-account>

3. When the default account is obtained. We will add logic to obtain to parse the particular account json text response in **temp** variable and obtain account ID and base URL.
4. Then we need to add second operation which will send the envelope given suitable information such as group ID, subject, document to sign etc.

If nintex was to support Open API v3.0.0 then the above proposed steps would be easy to do formulate in practice by making Open API v3.0.0 json File. This is because v3.0.0 support server templating meaning that server string becomes a variable and can be changed accordingly any time. While in v2.0 (swagger specification) the host (server) is fixed at start i.e. there is only one server per open API json file. We know that the base URL (server say <https://demo.docusign.net>) for an account can be different for each user account and is not the same as the one used for retrieving user information (account-d.docusign.com). So we need to have two server one for retrieving user information as seen above (account-d.docusign.com or account.docusign.com) and one specific to base URL (e.g. [www.docusign.net](https://demo.docusign.net), na2.docusign.net, eu.docusign.net, etc.) for making rest API calls ([link](#)).

For the above steps the proposed workflow could be like (tested):

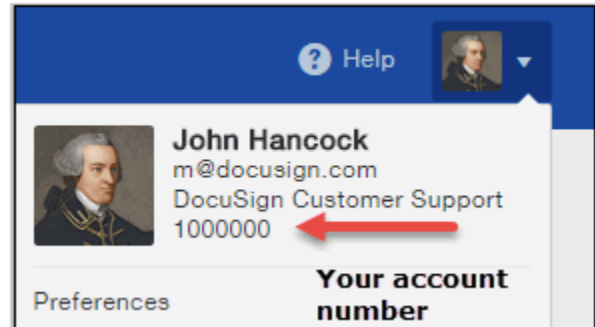
1. Get User Info action (1st operation, custom connector) stores the accounts node (array property) in json response as collection in variable.
2. Then the loop as I mentioned above. The first regular expression operation parse each account till it finds the default one and the loop breaks and account is stored as json text in variable.
3. The second regular exp. parses account ID from json text.
4. The second regular exp. parses base URL from json text.
5. The Create envelope action (2nd operation, custom connector) then uses base URL and account ID to form URL and send envelope to a group.



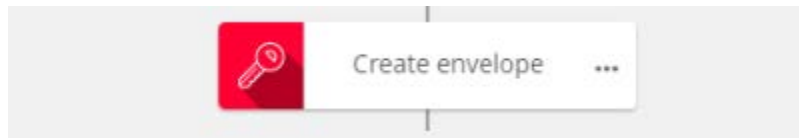
Note: the above regular expressions (ruby style) are not required and an easy operation “Query JSON” is available in nintex action menu. It uses json query a very easy syntax and also it will look as a standard practice to use it.

A Work Around (second proposal):

Since support for Open API v3.0.0 in nintex is coming soon (we don't know yet) so we have to work around with swagger specification (Open API v2.0). Then we will require base URL somehow without using nintex connector. Now for any sandbox (development) environment it is fixed (<https://demo.docusign.net>) but may vary for production. So it has to be noted down somehow. The account ID is not a trouble. We can instead use the account no. as shown in image.



So if Open API v2.0 is the option then the workflow reduces to:



I have created both specification.

Although the second operation is not yet designed for sending envelope to a group. Only an example is implemented for second operation.

Collection – Array Mapping Problem

(<https://learn.nintex.com/nintex-workflow-cloud-collection-operations>)

A collection is a container that can hold a list of values (or a single value). Each value in a collection can be a 'hard-coded' value or a value stored in a variable. For example, a collection can store the number 7, the text 'Hello', and a Date/Time variable for a specific date. Example:

[7, 'Hello', dtOpeningDay]

- 7 has an index of 0.
- 'Hello' has an index of 1.
- dtOpeningDay has an index of 2.
- No item exists with an index of 3.

A collection accepts many different types of variables. They may come from the Workflow as you create them, or they may originate from the Start event or the workflow Context. You cannot nest a collection within a collection.

In our request json object we have two arrays which does not have primitive items () but have items (a json object) which cannot map to items (string, integer or number) in collection. If the values were an integer or number the then mapping is possible.

```
{
  "status": "sent",
  "emailSubject": "Sent from the DocuSign REST API",
  "documents": [{
    "documentId": "1",
    "name": "contract.pdf",
    "documentBase64": "base64 document bytes...",
  }],
  "recipients": {
    "signers": [{
      "email": "jane.dough@example.com",
      "name": "Jane Dough",
      "recipientId": "1",
      "routingOrder": "1",
    }]
  }
}
```

Above “**documents**” and “**signers**” are arrays of object. That object cannot be mapped to values in collection. There was a workaround when **response** json object contains an array, that array object items becomes string in collection during mapping process and we can then use “Regular Expression or Query Json” Actions in nintex. I used it to get the account ID and base URL.

There is no workaround possible for creating **request** json object which has properties such as **complex array** (array of non-primitive values).

Last Step

See Request Token

<https://developers.docusign.com/esign-rest-api/guides/post-go-live>

See changing to Production:

<https://developers.docusign.com/esign-rest-api/guides/post-go-live>

References

<https://developers.docusign.com/esign-rest-api/reference/>

<https://developers.docusign.com/esign-rest-api/guides/concepts/envelopes>

<https://editor.swagger.io>

<https://developers.docusign.com/esign-rest-api/guides/authentication/user-info-endpoints>

https://help.nintex.com/sdk/Content/04_Reference/REF_KnownIssues.htm?tocpath=Reference%7C_____4

<https://apiexplorer.docusign.com/#/esign/restapi>

<https://mermade.org.uk/openapi-converter>

<https://goessner.net/articles/JsonPath/>

<https://developers.docusign.com/esign-rest-api/guides/post-go-live>