Document Tagging Scenario - Example showing how to collect metadata for DMS archiving

The document tagging story v 21.16

You want to save your files and find them again straight away? With DMS tagging you can add metadata so that you can save your documents and envelopes with additional data and thus you can more easily assign and find them.

The DMS Tagging integration scenario explains how to integrate a custom tagging page (e.g. aligned with the data structures from your DMS / ECM) into eSignAnyWhere, and how to use this data for later upload (with tagging) to the DMS/ECM solution.
In the eSignAnyWhere Web UI, an additional (custom) page is added before sending the envelope. This page collects tagging information. The data are stored in the envelope's metadata section. A callback handler service implementation is notified after envelope completion, and will extract the tagging information from the metadata and perform the upload to the DMS.

This story includes the following:

- Use of a custom page implemented in an external service on integration layer, used to do tagging of the envelope for further use of the DMS
- Callback handler process completed envelopes and store the results in the DMS

We recommend to read the story "Retrieving the User API Token via OAuth" before continuing with this story.

Application Flow – Tagging

You can define the envelope as usual, with the eSignAnyWhere WebUI envelope creater. After the recipients page and the designer page, the sender will see the summary page of eSignAnyWhere. In the summary page, instead of the "SEND" button, the sender will see a "NEXT" button because a "before-send redirect url" was configured.

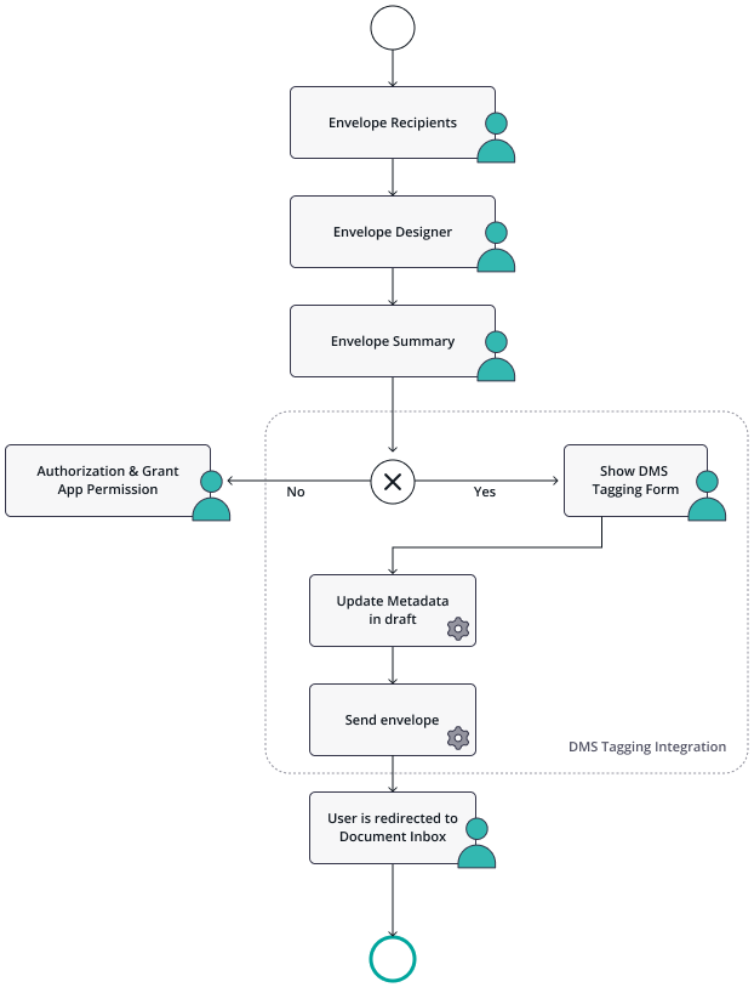
Following placeholders are available for the "before-send redirect url" :

- ##EnvelopeId##
- ##SenderUserId##
- ##OrganizationId##

When the sender clicks the "NEXT" button on the summary page, he/she is redirected to the configured before-send redirect page. This page, for example can allow to modify the envelope before sending. Therefore, the page is collecting the DMS Tagging metadata in a custom HTML form, and will set the data as metaDataXML into the envelope. On the authorization page, the OAuth 2.0 code grant flow is implemented, and therefore the user is asked, if not permissions have been granted for the application so far, to authenticate and grant permission for the custom web application to access the API. After successful authorization, the user is redirected back to the custom web application for the document tagging.

After he/she enters the data, the form action is calling the custom web application's back-end with the form data provided. The back-end will update the envelope (draft) data by adding the metadata, and will the send the envelope based on the draft. Afterwards, the user is redirected to the eSignAnyWhere documents inbox.

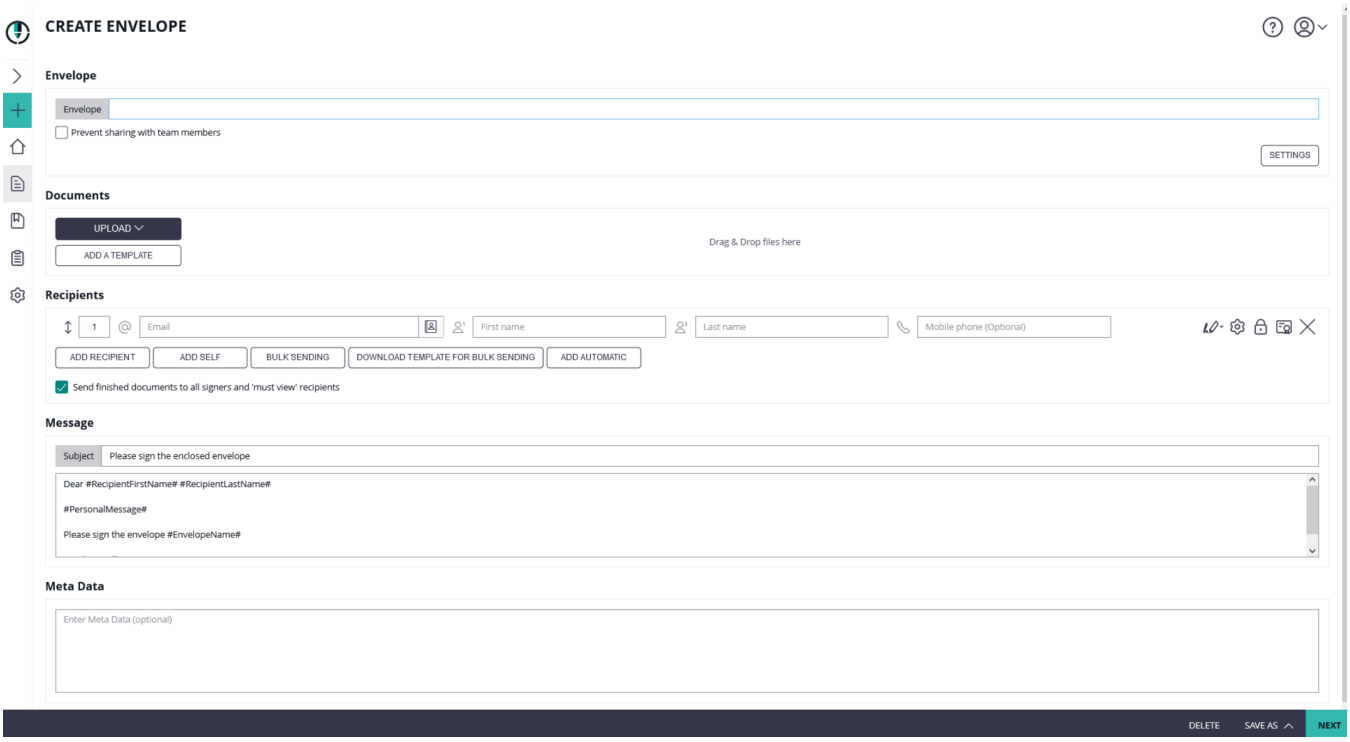
The next figure shows the process flow in detail:



This flow will be represented to the user in the following screens:

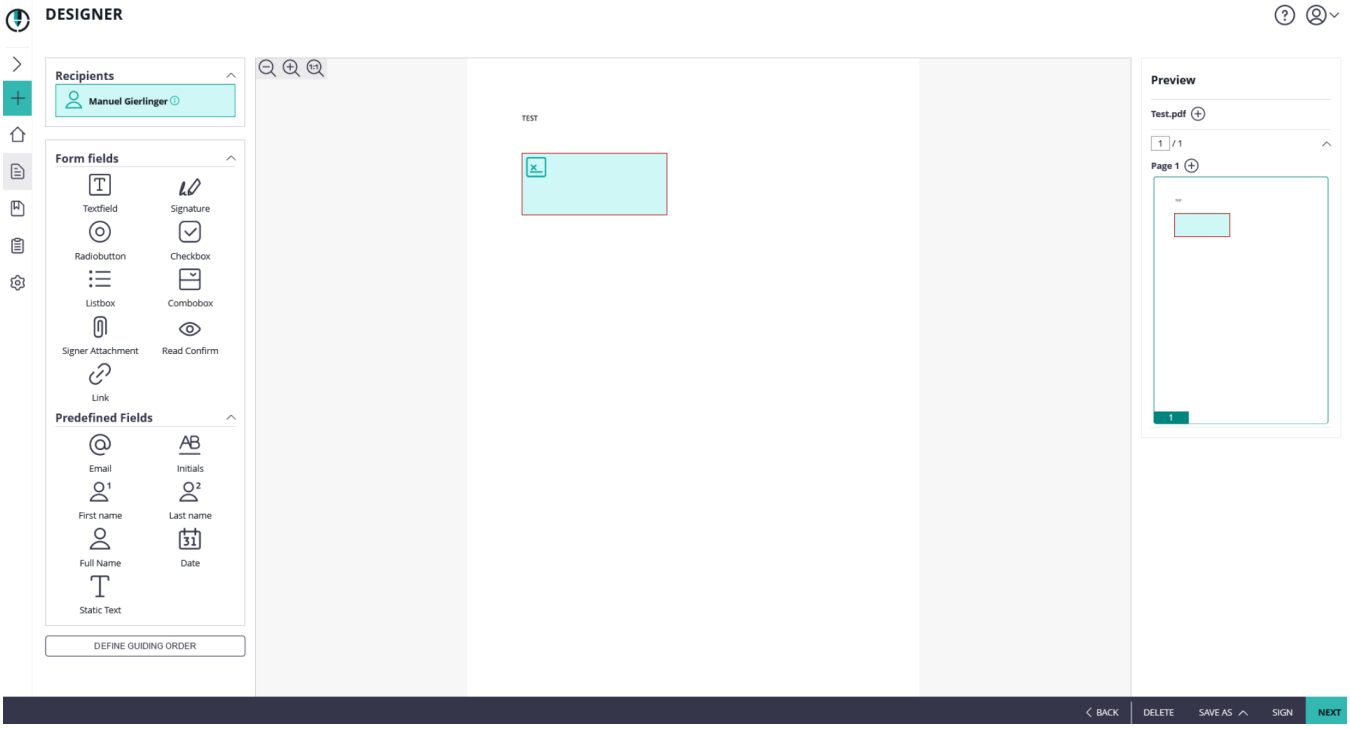
Recipients Page

In the recipients page, the sender selects documents to be signed, and defines the recipients and workflow. This is standard eSignAnyWhere functionality.



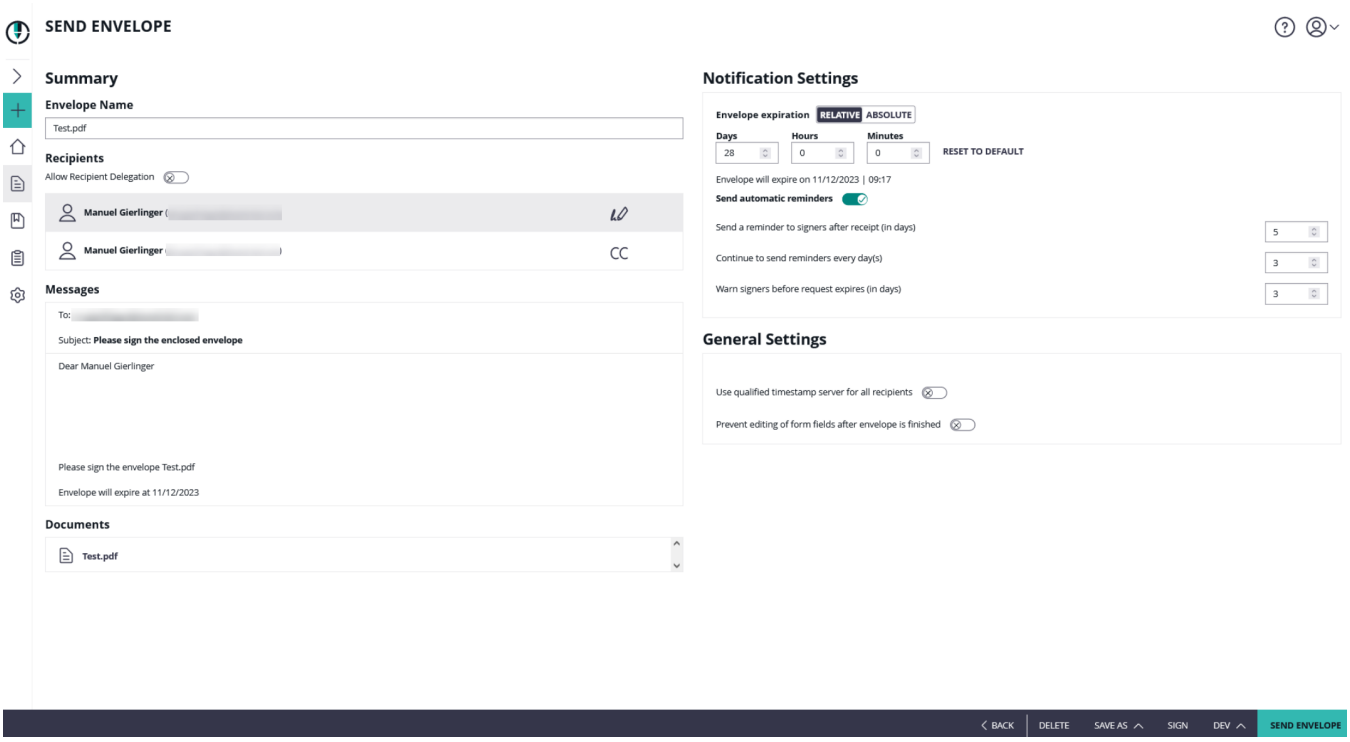
Designer Page

In the designer page, the sender may define or add form fields, signature fields and predefined annotation fields. This is standard eSignAnyWhere functionality.



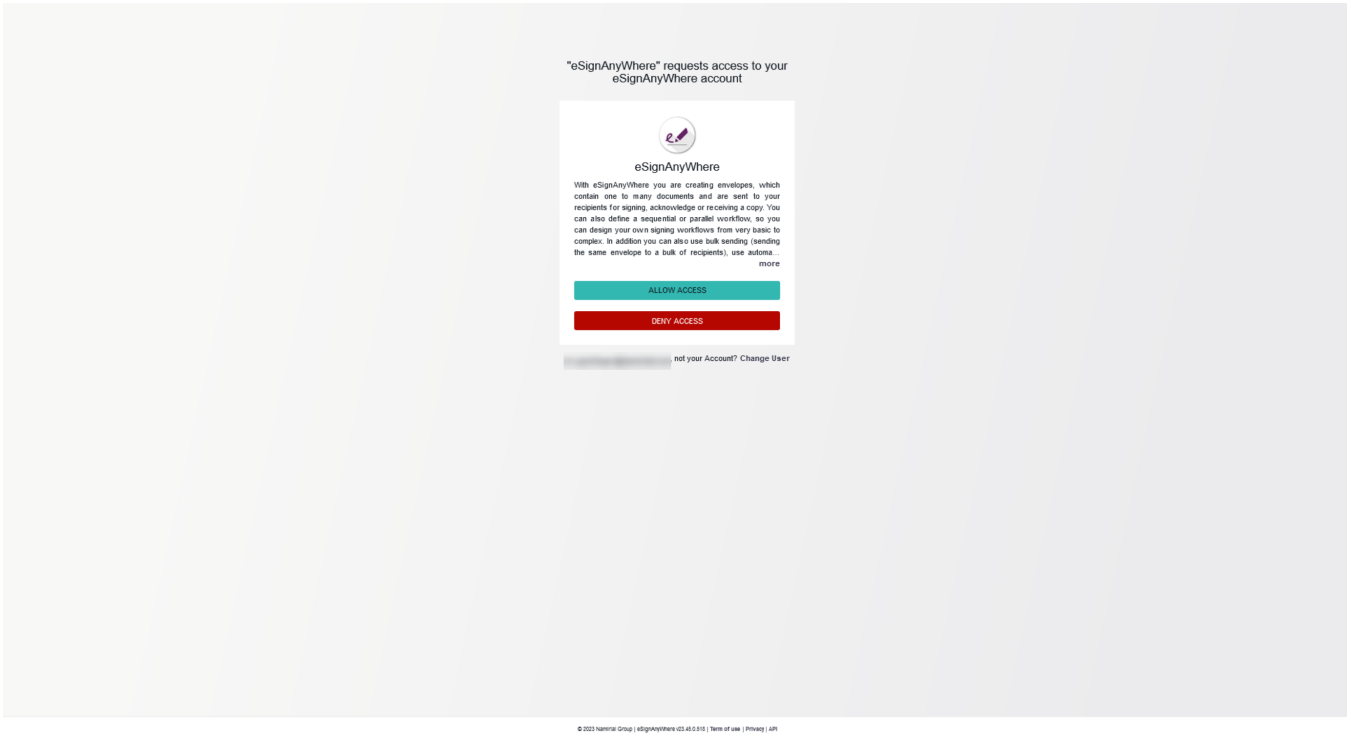
Summary Page

In the summary page, the sender may adjust some envelope settings such as notifications/reminders.



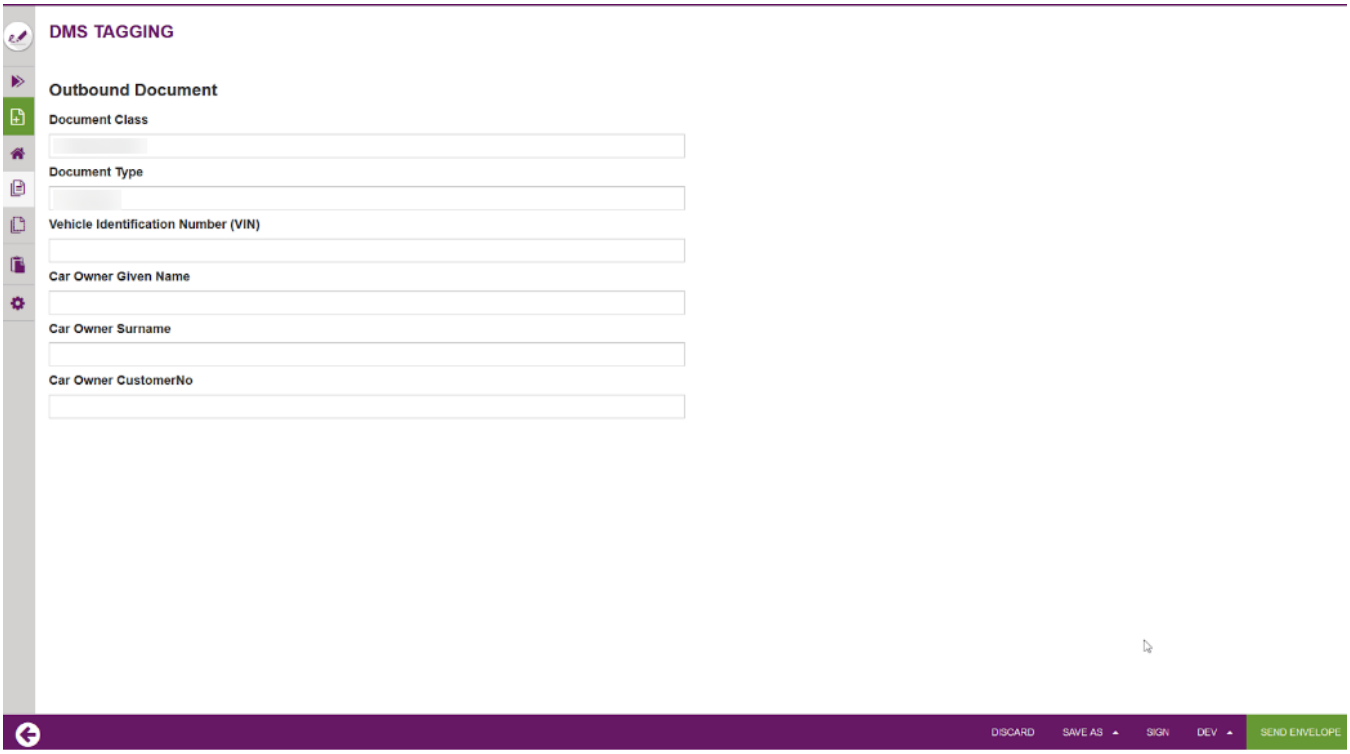
Authorization Request

Then the user is redirected to the configured redirect page. Before the tagging page is shown, the user may be asked to login (which is usually skipped because the user's session is already active), and on first usage the app asks for permissions to grant API access. Details of this authentication are well described in the story "Retrieving the User API Token via OAuth".

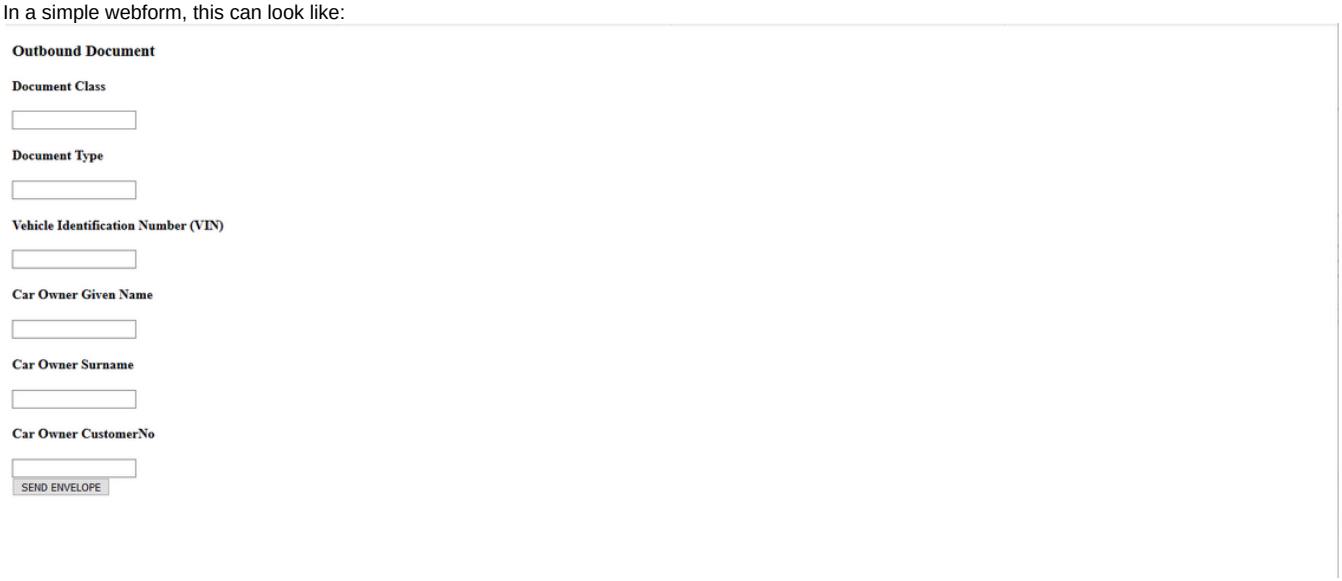


Custom Tagging Page

This page is fully implemented in the custom web application for DMS tagging. The fact that it looks like the eSignAnyWhere Web UI is just because it was implemented that in a sample code. This is purely part of the custom DMS tagging sample.



In a simple webform, this can look like:



Serverside handling of Form Data

After clicking the "Send Envelope" button in the custom tagging page, notify the backend of the tagging page implementation (also called "BeforeSendRedirectPage") and let the backend do the following with the retrieved form data:

Update Metadata in Draft

Set the data collected via the form e.g. in draft metadata section, to make these data available for the later callback processing
This will make the data available for pushing the signed documents to the document management system (DMS)

POST /v6/draft/update

POST body

```
{
  "DraftId": "...",
  "MetaData": "..."
}
```

Send Envelope

While being on the BeforeSendRedirectPage, we are still working on a draft. There was no envelope sent yet. To send the envelope, we use an API method for the transition from draft to envelope:

POST /v6/draft/send

POST body

```
{
  "DraftId": "..."
}
```

Document Inbox

After the envelope is sent, the custom web app should redirect to the eSignAnyWhere document inbox. This is standard eSignAnyWhere functionality; the custom web application just needs to redirect to that page once done.

Note if you want to cancel the settings:

Disable the beforeSendRedirectUrl in Settings-Organization. After all pending envelopes (drafts) have been sent, disable the OAuth Application (prevent log-in via this OAuth App). When disabling the OAuth App, existing drafts still contain the beforeSendRedirectUrl, so the OAuth Application should not be blocked before those drafts have been sent.

Application Flow – Callback Handler

After the callback for finished envelope was received, the callback handler should check if it was a completed action. The handler has to retrieve the envelope ID from the HTTP GET parameters. Then, the callback handler should note down the envelope ID in a persisted storage for further processing and return with an HTTP 200 (success) to indicate the archiving request was noted down.

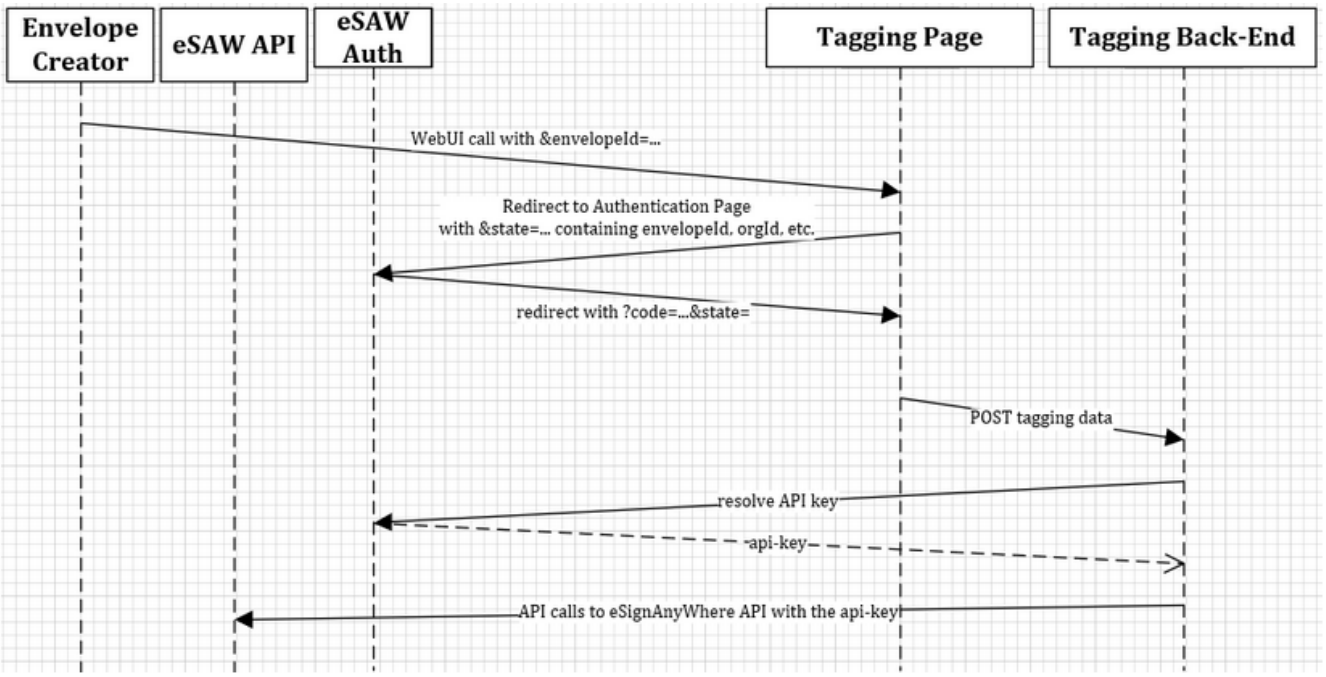
For more information about server-side post-processing using callback handlers please see this story.

Sample Implementation of the OAuth2 Code Grant Flow for the Tagging Scenario

A common situation is that the eSignAnyWhere user is redirected to a custom web application which should deal with the envelope sent before via eSignAnyWhere Web UI. To access the envelope via API, the custom web application must authenticate to the API with a user that has the permission to access the envelope.

While this was difficult in the past and often required to keep records of a mapping of envelopes to the creator, this got much simpler since eSignAnyWhere supports the OAuth2 Code Grant authentication flow. The user of the custom web application is asked to authenticate, via OAuth2, with the eSignAnyWhere user credentials. As a result, a code sharing between eSignAnyWhere and the custom web application is triggered and as a result the custom web application gets API credentials to access the envelope. During authentication, the user may choose the wrong user account on eSignAnyWhere – therefore it is required to check the permissions for the envelope, and redirect to the authentication again if the authentication provided does not have the required permissions.

Sample sequence diagram of a sample application we implemented for DMS tagging:



Note that the invocation of the tagging page cannot contain the "envelopeId" as HTTP GET parameter, as GET parameters are not allowed in the OAuth 2.0 redirect_uri. This is because the OAuth application configuration in eSignAnyWhere AdminWeb must already be whitelist the full URL including all parameters. Therefore, the sample application we implemented is encoding all GET arguments from first call into the “state” parameter of the authentication call. The redirect_uri is invoked with the state parameter, as specified in RFC 6749. So the state is used in that case to transport all the GET parameters.

The callback handler implementation also requires to know the API Token (Bearer Token) of the sender to retrieve signed documents, audit trail etc. – therefore the DMS Tagging Page implementation is already storing a mapping between envelopeId and senderUser and also a bearertoken of the senderUser in its persistent storage.