Content Tagging Manual

Author: Tim van der Meijs
Version: 0.1
Date: 10 March 2022

Revision history

Version	Date	By	Change
0.1	2022-03-10	Tim van der Meijs	First draft of feature “Content Tagging Manual“

Abstract

This document contains all information to be able to fetch topics from AskDelphi’s API using context tags. It will explain how to authenticate users to retrieve JWT tokens, fetch content tags using the RulesAPI and how to get the corresponding topics that are linked to the content tags.

Authentication

User authentication with AskDelphi can be done in two different ways. The first one is by using the interactive login mechanism. A link to the login-page is obtained through the AskDelphi API, depending on the Publication used.

The second option is by using a session code that can be extracted from a QR code. Both methods will be explained in this chapter.

Method 1: Login Mechanism

To authenticate using the interactive login mechanism we use the following endpoint:

```
curl -X GET --header 'Accept: application/json' 'https://imola-prod-api-
2c3f1a34.azurewebsites.net/api/TokenAPI/tokenurl?publicationGuid=PublicationGuid'
```

Only one parameter is required to return a valid response i.e., the PublicationGuid (which can be retrieved through the AskDelphi CMS). This endpoint will return the login page for the parameter given by the request e.g.:

```
"https://example.azurewebsites.net/api/Token/Login"
```

This page should be opened in a webbrowser view that's controlled by the application that wants to log the user in.

This page will take the user through the regular SSO login process that's configured for the AskDelphi publication. Eventually this should lead to a redirect back to the Login page, but this time with an authenticated user.

When there is an authenticated user, a JWT token and refresh-token pair is generated that can be retrieved by the controlling application by looking for the following response headers. After these headers values are obtained, the web view can be closed again.

```
X-Authentication-RefreshToken
X-Authentication-Token
```

The X-Authentication-Token can then be used as an Authorization header for all other API calls in the following way:

```
Authorization: Bearer <X-Authentication-Token>
```

If the JWT token is expired the refresh endpoint can be called to retrieve a valid token:

```
curl -X GET --header 'Accept: application/json' 'https://imola-prod-api-
2c3f1a34.azurewebsites.net/api/TokenAPI/refresh?token=Token
&refreshToken=RefreshToken&publicationGuid=PublicationGuid'
```

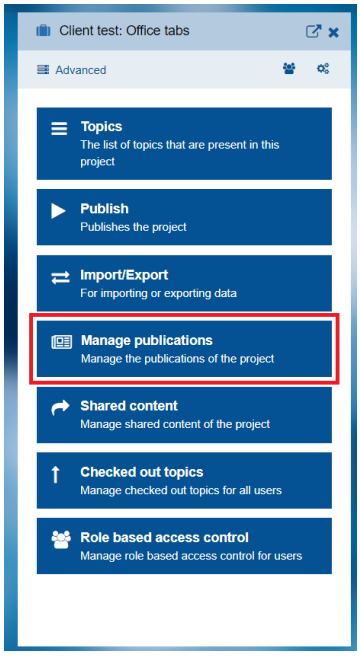
The PublicationGuid, Token and RefreshToken parameters are all mandatory. The response will look like this:

```
{
 “refresh”: “Refresh Token”,
 “token”: “Valid JWT Token”
}
```

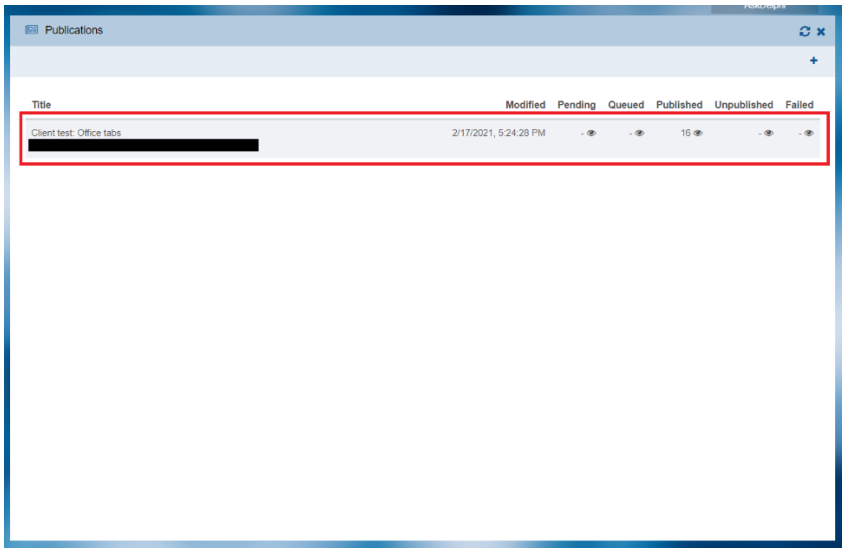
Method 2: Session Code

To authenticate using a session code we need the user to scan a QR-code which can be found the AskDelphi website. This feature needs to be enabled in the CMS by opening a project and following instructions below.

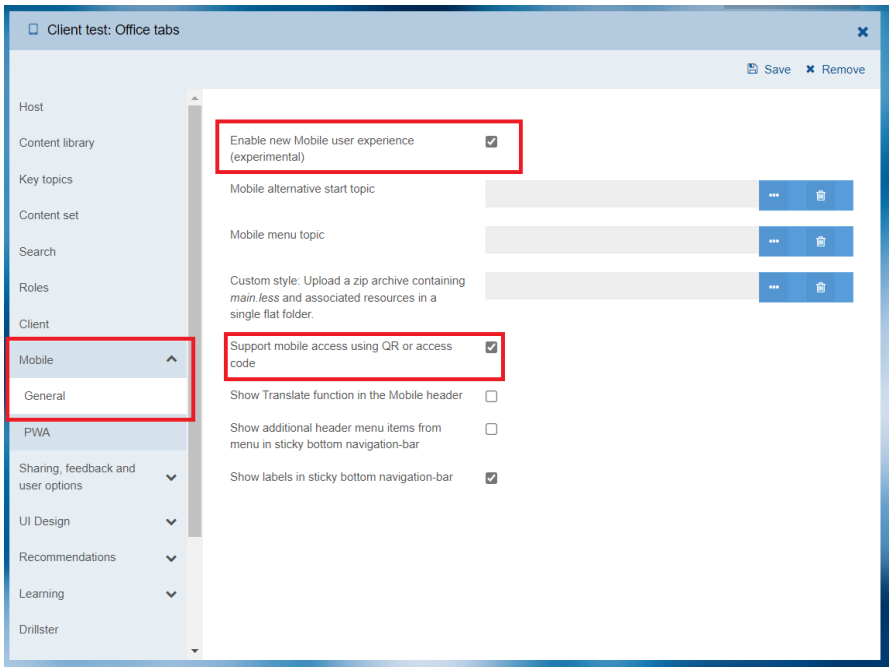
Enabling QR-codes



Step 1: Click Manage publications



Step 2: Click the desired publication



Step 3: Click Mobile > General

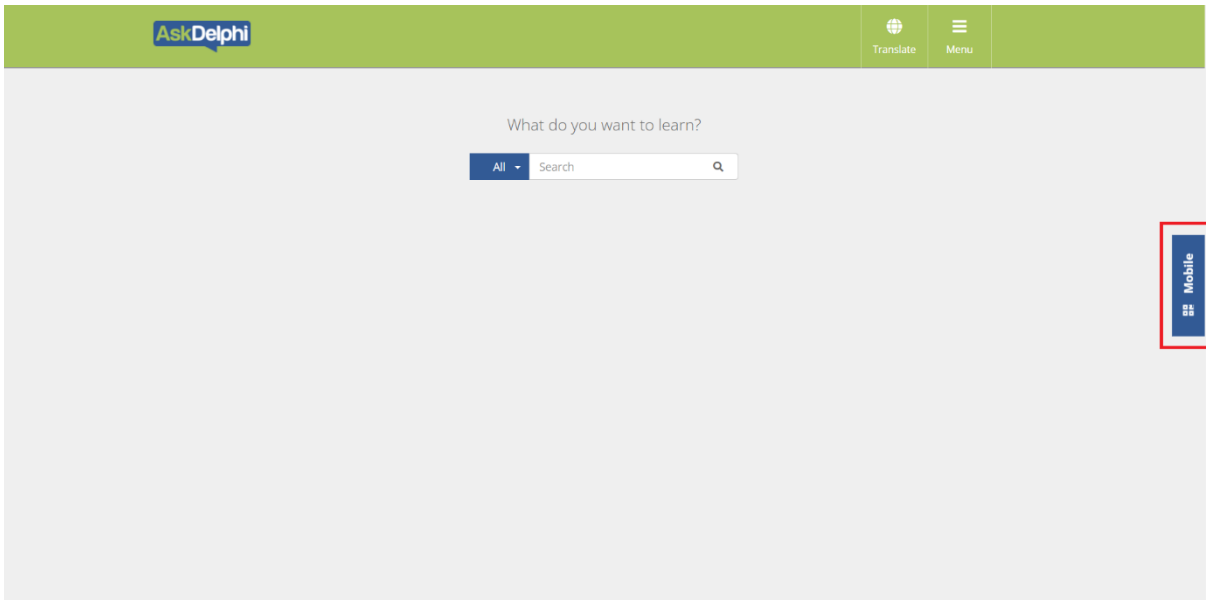
Step 4: Check 'Enable new Mobile user experience (experimental)'

Step 5: Check 'Support mobile access using QR or access code'

After this you can republish the project and the QR code will be available in AskDelphi.

Scanning QR-code

To obtain the session code from the user, the user needs to scan the QR-code in AskDelphi from which we get a link where we can extract the session code to make the corresponding API call to obtain JWT tokens. Below is explained how the user can obtain the QR-code.



Step 1: Click Mobile

Step 2: User captures the QR-code with their camera

The QR-code will contain a link that is built up in the following way:

```
https://portal.askdelphi.com/session/<sessioncode>
```

To use this mechanism for logging in a user in an application, only the **sessioncode** part of the URL is required.

Once the session-code is parsed from the URL the following endpoint should be called to retrieve the JWT token and refresh token:

```
curl -X GET --header 'Accept: application/json'
'https://portal.askdelphi.com/api/session/registration?sessionCode=SessionCode'
```

The endpoint will then return the following JSON response body:

```
{
 “id”: “Session ID”,
 “accessToken”: “Valid JWT Token”,
 “refreshToken”: “Refresh Token”
 “url”: “Publication URL”
}
```

From here the accessToken and refreshToken can be extracted and used as Authorization header in upcoming call, and can be refreshed using the /api/TokenAPI/refresh endpoint.

```
Authorization: Bearer <AccessToken>
```

Storing tokens

Regardless of how the user is authenticated, the application will have obtained a Bearer token (JWT identity token) and a Refresh-token for the user session. Both tokens should be stored by the application and used afterwards. There should not be a need for the user to log in again until their refresh token is either expired or withdrawn.

A word of warning here: After a bearer token is refreshed once it is no longer valid. Be careful with automatic refreshing when calls fail. If two calls are executed simultaneously and both try to refresh the same token, then only one will succeed. It’s best to serialize the refresh process.

Fetching relevant content

Fetching topics for context-sensitive support from the AskDelphi API is done in two steps, firstly an XML document describing the application state is crafted and pushed through a rules-engine. This yields the possible contexts that the user could be in. These contexts have a title for display purposes and a tag to allow searching for related content topics in a next step.

This chapter will explain how to make these calls.

Fetching matching Context Tags using the rules-engine

To get a collection of context tags from a publication we need to put the application's state in an XML document, so the API can process it. For example, for Microsoft Office, the XML document describing the application state could look like this:

```
<context>
        <BasicWindowProperties window-class="OpusApp"></BasicWindowProperties>
        <WordRibbon>
                <Tab name="Home" active="true"></Tab>
        </WordRibbon>
</context>
```

This could for example represent the Home ribbon being the active ribbon in the user's Word application.

Applications should define and document their own XML structure used. This should contain enough information for the content-authors to identify what the user is currently doing. The rules engines uses XPath expressions to do this.

Depending on the rules and how they are created in the CMS the way rules are formatted can look differently. Once the state is defined, the matching context identification tags can be obtained using the following endpoint:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
application/json' -d '{ \
   "contextXML": "<<Your XML document as a string>>", \
   "publicationGuid": "<<Your publicationguid>>", \
   "state": {}, \
   "tenantGuid": "<<Optional>>" \
 }' 'https://imola-prod-api-2c3f1a34.azurewebsites.net/api/RulesAPI/match-
es'
```

Only the contextXML and publicationGuid are required parameters, the other parameters need not be specified. Once this call successfully is made the following reply will be sent back:

```
{
 "totalcount": 0,
 "matchingRules": [
        "description": "string"
        "score": 0,
        "tag": "string",
        "publicationGuid": "string"
 ],
 "state": {}
}
```

From the JSON body you will be able to extract the tags corresponding to the XML application state. These tags can then be used to retrieve all information topics using the call described below.

Note that this document does not describe how to define the mapping rules and/or how to map the tags to specific topics. This is documented in the user documentation.

Searching Topic Details

The previous call returns the names (tags) for all contexts that the user could be in according to the context detection rules. Quite often there will be a very specific context (e.g. the user is entering a sales order for an amount greater than a certain threshold), and some more generic contexts (e.g. the user is entering a sales order).

Your application should pick the lowest priority context (the one returned first) as it typically is most specific and should show the list of supporting content topics for that to the user. The application can then show the other contexts as “alternative” contexts to the user.

The list of content topics relevant to the tags can be obtained using:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ \
   "tag": "<<your tag>>", \
   "publicationGuid": "<<your publicationguid>>"
}' 'https://imola-prod-api-2c3f1a34.azurewebsites.net/api/SearchAPI/searchbytag'
```

This call will return all topics that are linked to the tag retrieved in the previous call. The JSON response body will look as the example below, where for all matching topics the title and URL are returned. In addition, also outgoing relations for the topic are returned, which might be useful in some cases.

```
{
  "count": 0,
  "offset": 0,
  "totalcount": 0,
  "results": [
    {
      "score": 0,
      "relations": [
        {
          "relationtypeid": "00000000-0000-0000-0000-000000000000",
          "tag": "string",
          "sequencenumber": 0,
          "description": "string",
          "publicationid": "00000000-0000-0000-0000-000000000000",
          "topictypeid": "00000000-0000-0000-0000-000000000000",
          "version": {
            "MajorVersion": 0,
            "MinorVersion": 0
          },
          "id": "00000000-0000-0000-0000-000000000000",
          "title": "string",
          "url": "string"
        }
      ],
      "description": "string",
      "publicationid": "00000000-0000-0000-0000-000000000000",
      "topictypeid": "00000000-0000-0000-0000-000000000000",
      "version": {
        "MajorVersion": 0,
        "MinorVersion": 0
      },
      "id": "00000000-0000-0000-0000-000000000000",
      "title": "string",
      "url": "string"
    }
  ],
  "facets": [
    {
      "count": 0,
      "hierarchy": "00000000-0000-0000-0000-000000000000",
      "id": "string",
```

From this response body you should, extract the title and the URL to each of the content topics. That should be enough to display the list of matching content to the user. The content of each of these topics can be obtained for further processing using the following call:

```
curl -X GET --header 'Accept: application/json' 'https://imola-prod-api-
2c3f1a34.azurewebsites.net/api/ContentAPI/topic?publicationGuid=Publica-
tionGuid&topicGuid=TopicGuid'
```

The TopicGuid can be extracted from the JSON body from the last call, the parameter is called id. The call will return the following JSON body with the XML body attached:

```
{
 "bodyXml": "string",
 "relations": [
  {
   "relationtypeid": "00000000-0000-0000-0000-000000000000",
   "tag": "string",
   "sequencenumber": 0,
   "description": "string",
   "publicationid": "00000000-0000-0000-0000-000000000000",
   "topictypeid": "00000000-0000-0000-0000-000000000000",
   "version": {
    "MajorVersion": 0,
    "MinorVersion": 0
   },
   "id": "00000000-0000-0000-0000-000000000000",
   "title": "string",
   "url": "string"
  }
 ],
 "description": "string",
 "publicationid": "00000000-0000-0000-0000-000000000000",
 "topictypeid": "00000000-0000-0000-0000-000000000000",
 "version": {
  "MajorVersion": 0,
  "MinorVersion": 0
 },
 "id": "00000000-0000-0000-0000-000000000000",
 "title": "string",
 "url": "string"
}
```

Summary

Above the full API flow is described from how to log in until fetching and displaying the contents of topics.

The flow:

1. Authorize the user in an interactive session
   Obtain the X-Authentication-Token and X-Authentication-RefreshToken response headers
2. Refresh the JWT token using /api/tokenapi/refresh when it expires (you can check for expiry by decoding the JWT token, or by responding to 40x responses)

3. Construct XML describing your application's state according to a format agreed with the content team
4. Pass XML to the **/api/rulesapi/**matches endpoint to get "tags"
5. Pass a tag to the **/api/searchapi/searchbytag** endpoint to get a list of titles and URLs of matching content topics

6. Optionally, fetch more details on these content topics using the **/api/contentapi/topic** endpoint.

About context

A context XML document should represent your application's state and should contain information on what the user is doing.

If your application has named screens, and named fields you could create a context XML document that looks like this:

```
<context>
        <ActiveScreen module="abc123" tab="tab3"></ActiveScreen>
        <Fields>
                <Field id="product">Some product ID</Field>
                <Field id="amount">9999.99</Field>
        </Fields>
</context>
```

The author creating the rules could then use an xpath like /context/ActiveScreen[@module='abc123']/Fields[@id='amount' and value() > 1000] to identify that the current user is in this module and has entered an amount exceeding 1000 units.