Enabling live integration with an external CMS/DMS by implementing the AskDelphi Remote Content REST API

Author: Pieter-Bas IJdens
Version: 1.1
Date: 10 November 2022

Status: Final

Classification: Public

Revision history

Version	Date	By	Change
0.1	2020-11-25	Pieter-Bas IJdens	First draft
0.9	2021-01-26	Pieter-Bas IJdens	Updated API documentation, offering document for review
1.0	2021-02-12	Pieter-Bas IJdens	Updated for release
1.1	2022-11-10	Pieter-Bas IJdens	Returning valid and stable guids is mandatory now for topic-content objects. The adapter will as of now determine the topic guid for exported content.

Abstract

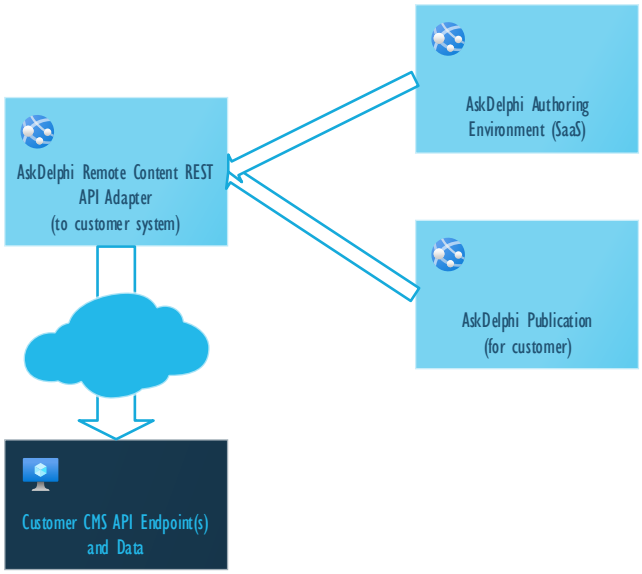
The AskDelphi system comes with a built-in CMS that's very rich in features, and can be used to model almost any content design. However, in organizations where AskDelphi is being deployed it's not unusual that large amounts of content are already available, and are managed in a LMS or CMS other than AskDelphi.

Of course AskDelphi offers many methods of importing this remote data into the system, but to an organization it may also make sense to keep using their existing CMS/LMS for authoring content. In those cases, the AskDelphi system offers the option of a live integration.

This document describes the system architecture, interfaces and user interaction related to this feature.

Overview

On a very high level, four components play a role, disregarding details such regarding networking, security and the AskDelphi internal service infrastructure.



The direction of the arrows indicates the direction of the control flow.

In this picture we see the following components:

- **The customer CMS API Endpoint(s) and Data**
  This represents the remote CMS/LMS system that is being integrated with. This may in reality be numerous systems with a great variety of API endpoints and data stores, but for purposes of simplicity we'll regard those as a whole.

- **AskDelphi Authoring Environment (SaaS)**
  This represents the SaaS environment in which customers edit their content for AskDelphi. In there. This is where for the customer's tenant options are unlocked for referring to remote data and documents.

- **AskDelphi Publication (for customer)**
  This represents the node in the SaaS publication environment that is used for presenting data to the end-user. This is where for the customer's tenant options are needed for obtaining the (structured) content or published documents, so they can be presented to the end-user.

- **AskDelphi Remote Content REST API Adapter (to customer system)**
  This service provides a standardized REST API interface that can be used for accessing remote content by the AskDelphi systems. Regardless of the underlying system, the interface offered here is the same, and is stable. Per integration one implementation of this API, specific to the remote system, will be deployed. This deployment can happen in the AskDelphi cloud, but could also be done on the customer's local network and made accessible to AskDelphi via a tunnel.

Terminology

Adapter: In software this is a component that implements an alternative interface on top of an existing system.

API: Application programmer's interface. Offers one or more endpoints to third-party systems to programmatically perform operations.

AskDelphi Authoring Environment: The AskDelphi web-application that can be used to edit content and configuration data. Also used to define and run publications.

AskDelphi Publication: The end-user-facing part of the AskDelphi software. Provides an online web-application for end-users to access the customer's content.

AskDelphi Remote Content REST API: An API definition that can be implemented by or on top of a customer CMS.

CMS: Content management system; a system intended for managing content in some structured manner. Can refer to either the AskDelphi CMS or the Customer CMS/DMS.

Content: Any data normally stored in topic form in the AskDelphi CMS.

Content design: Definition of the structure of the content and the relations between content elements in the AskDelphi CMS.

Customer CMS: The CMS or DMS of the customer, either privately hosted inside the customer's network or publicly accessible

DMS: Document management system; a system to store and retrieve documents. Typically offers features for workflow, version control and access control for sets of documents.

Endpoint: A single function in an API.

Published asset: A PDF-document, video, SCORM package, HTML package or image that originates from a remote system and that represents the published version of some content in that system.

Resource: Any binary data such as Image, Video or File resources. Resources are part of the content but do not play any role in the content design.

Seamless integration: We consider integration of content seamless when it's not obvious to the end-user of the published content that parts of the content actually originate from a remote system.

Taxonomy: Hierarchy.

Topic namespace: Topic namespaces identify the structure of the information item. Based on the topic's namespace a certain structure is expected for a topic in the backend.

Topic type: Entry in the content design identifying a topic type. A topic type always uses exactly one namespace, but multiple topic types may use the same namespace.

Virtual folder: An organization into folders of assets in the customer CMS, where these folders have parent-child relations. In a virtual folder structure, the folder hierarchy may considerably differ from the actual folder hierarchy in the customer CMS, for example to combine assets from multiple CMS instances into a single folder structure.

Functional description for the publication

From an end-user point of view, deploying this API unlocks the following functions in the AskDelphi publication:

Use case: As a publication-user I want to read the latest released version of content from an external system as an integral part of an AskDelphi publication

In this use case content from the customer CMS is made available through the AskDelphi publication. From a business point of view the requirements are:

- Content from a customer CMS is included in the AskDelphi publication
- This content is seamlessly integrated with the AskDelphi publication
- It's possible to navigate to specific child-content via the sidebar and relations
- It's possible to present images, video's and embedded documents as part of this content
- Seamlessly integrated remote content should be available in the same target languages as the original content is
- When a different version of this content gets 'published' status in the customer CMS, then this newer version of the content is available in the AskDelphi publication almost immediately after publication

Use case: As a publication user I want to include references to published assets from the customer CMS in my content

In this use case, references to published assets from the customer CMS are included in the AskDelphi publication by means of hyperlink or direct download. Requirements are:

- Side-panel and inline navigation is offered to these published assets
- It's possible to offer these assets for download
- It's possible to hyperlink to these assets, for opening them in a new tab
- It's possible to use these assets as resources in AskDelphi, e.g. to provide a SCORM package, packaged HTML or a look and feel definition
- When a different version of this content gets 'published' status in the customer CMS, then this newer version of the content is available in the AskDelphi publication almost immediately after publication

Use case: As a publication user I want to embed published assets from the customer CMS in my content

In this use case the content from the customer CMS is embedded in the AskDelphi publication. The content could be displayed like an embedded PDF, but might as well be a full HTML package that's rendered inside an embed box in the AskDelphi publication. Requirements are:

- Content from the customer CMS can be rendered inside an embed-box in the publication
- Features such as zoom are available as always
- It's possible to provide embedded SCORM packages, embedded packaged HTML and embedded documents and images this way.
- When a different version of this content gets 'published' status in the customer CMS, then this newer version of the content is available in the AskDelphi publication almost immediately after publication

As a publication user I want to use AskDelphi search to find all content that's included from the customer CMS the same way I find local content

This use case deals with search. A large part of the publication users will use search rather than navigation to find their content. For those users, it should be possible to find the content from the customer CMS via search, very much like they can also find local content. For seamlessly integrated content, also the search results should be seamlessly integrated.

Requirements:

- That content from the customer CSM that's included in the publication should be found via search.
- There is no hard requirement to offer translated content via the AskDelphi search mechanism

Functional description for the authoring environment

From an author's point of view, deploying this API unlocks the following functions in the AskDelphi publication:

Use case: As an author, I can browse for content in the customer CMS just like I can browse for local content topics

When in the AskDelphi Authoring Environment a user is allowed to insert a reference to local content topics, they can also choose to include a reference to a dynamically created topic from a remote system instead.

They can however not create new content in the remote system on-the-fly, like they would be able to do in the local system.

Requirements:

- When a topic chooser is presented to the author, they get an option to browse for remote content instead.
- The remote content chooser looks very much like the AskDelphi chooser, including the presence of the filter bar.
- In addition, the topic chooser offers a 'folder' browser, allowing the user to navigate to the remote content by means of (virtual) folders.
- The author can search for remote content

Use case: As an author, I can create resource topics that refer to published assets from the customer CMS

In AskDelphi there are special resource topics for blobs, images and video's that act as a container surrounding the actual resources. For these topics special alternatives will be available where the resource is a reference to the resource in the customer CMS instead.

Requirements:

- When the author creates a remote resource, they can browse a folder structure in the customer CMS and select a file resource there.
- The author can search for resources.
- The author can't mix local and remote assets in a single resource topic.

AskDelphi Remote Content REST API definition V1.0

In order to offer the functionality for the authoring environment, we require the customer CMS to offer (by means of an adapter) the following endpoints.

Protocol

The implementing system should offer this API on HTTPS only. AskDelphi will not connect to plain HTTP endpoints or HTTPS endpoints that do not at least support TLS 1.2.

Request versioning

All API methods must require a query parameter as input:

- api-version
  If this version of the specification is followed the value must be 1

Standardized API responses and response versioning

Upon success, all API methods will always return a JSON-structure like this:

```
{
  "success": "true",
  "version": "1",
  … response data …
}
```

The **version** field must indicate which API version specification has been used to create the response objects. This version number must not exceed the version number specified in api-version on the input query string.

Upon failure, all API methods will return the following structure:

```
{
  "success": "false",
  "version": "1",
  "code": "<error code string>",
  "message": "<error message string>",
  "id": "<string>",
}
```

The **code**-field should uniquely identify the cause of the error, in a way that makes sense to the implementing system and can be used to track down the underlying issue, for example in the service logs.

The **message** field may contain some additional information about the problem. It must never return any implementation details such as stack-traces and key or token values.

The **id** field should contain a unique operation ID.

Authorization

The authorization with the AskDelphi Remote Content REST API will be based mostly on using an API key. The API will work in two modes. One mode in which a system account is used, and no user identification is provided.

To allow the AskDelphi Remote Content REST API to customize the returned content for the logged-in user, the system can be configured

Logging in

Method	POST
Endpoint	/api/auth/login
Headers	Authorization: Bearer <api-key>
Input	Body parameters - **claims *(optional)*** JSON-encoded array of tuples { "type": <string>, "value": <string> } containing the claims of the currently logged-in user. The adapter that implements this function may use this to determine the user's identity on the customer CMS. The caller of this API will always include a special claim of type http://tempuri.org/askdelphi/remote-system-id with a string value. This special claim provides the system with an identification of itself inside the AskDelphi environment. If the content APIs need to return automatically generated remote content objects, they can include the value of this claim as an ID. If the claim is not present, they can use an empty string.

Output	Content-type: application/json
	{
	"success": "true",
	"version": "1",
	"token": "<jwt-token>",
	(optional) "refresh": "<refresh-token>"
	}

Description	This method can be used to 'log in' to the adapter using the secret API key that should only be known to systems that have access to the API. If claims are specified, the API may encode some details of the logged-in user in the claims part of the returned JWT token. Having those details in there could allow other calls to this API to customize results based on the logged-in user's authentication claims. A refresh token may be returned to allow an implementation to extend the lifetime of a JWT token. Supporting this functionality is entirely optional. Functionally there is no need to support refresh tokens, as the caller should be able to use the login endpoint again to get a new token. The lifetime of the returned JWT token should be limited and should not exceed 60 minutes. JWT Tokens must comply with [RFC7519] and must include "aud", "nbf" and "exp" fields.

Logging out

Method	GET
Endpoint	/api/auth/logout
Headers	Authorization: Bearer <jwt-token>
Input	

Output	Content-type: application/json
	{
	"success": "true",
	"version": "1"
	}

Description	This method should invalidate the supplied JWT token with the adapter. It should also invalidate any and all associated refresh tokens, meaning that any and all requests places to the implementing system using those tokens will be refused in the future. This method must always return success

Extending the token lifetime

Method	GET
Endpoint	/api/auth/refresh
Headers	Authorization: Bearer <jwt-token>
Input	Query parameter: - **refresh: <refresh-token>** The refresh token that was obtained earlier using one of the login methods.

Output	Content-type: application/json
	{
	"success": "true",
	"version": "1",
	"token": "<jwt-token>"
	}

Description	This method may return an updated copy of the supplied token, with an extended life-span. Implementations may choose to not implement this method, in which case the calling system should simply request a new token upon expiration of the token.

Accessing resources

The following section of this document deals with accessing resources. These are single files containing some form of bundled content. Examples are images, video's PDF documents, SCORM packages or packaged HTML.

Listing resources in virtual folders

Method	GET
Endpoint	/api/resources/list
Headers	Authorization: Bearer <jwt-token>
Input	Query parameter: - ***(optional)* folderId: string** Uniquely identifies the folder of which the contents should be listed. If null , empty or not specified, the request is for the root folder of the virtual directory structure.
Output	Content-type: application/json { "success": "true", "version": "1", "folders": [<folder-descriptor>],

	"resources": [<resource-descriptor>] }
Description	Returns the contents of a single virtual folder of resources in the customer CMS system. How folders are organized is left up to the adapter that implements this interface. There is no need for this virtual folder structure to match 1:1 with actual folder structures in the customer CMS. A single interface implementation could even combine multiple sources of resources into a single virtual directory structure this way. The resource identifier and folder identifiers are strings of potentially unlimited length. Implementing systems should use these to encode the all required information to find the resource on the remote system. The list of resources and folders will be presented to the end-user in the order it is returned by this endpoint. It's strongly suggested to return these lists ordered by name. When using serialized forms of complex structures as identifiers, it's suggested to base64 encode the structures.

Searching for resources

Method	GET
Endpoint	/api/resources/search
Headers	Authorization: Bearer <jwt-token>
Input	Query parameter: - **query: string** Query string to be used as input to the search mechanism for this content-source. - ***(Optional)* page: number** Implementations can page their results, this parameter can be used to specify the page number for the search result. If not specified, implementation should return the first page of results. Page numbers start at zero. - ***(Optional)* size: number** If specified, contains the number of (additional) results the system would like to receive per page. If specified, the result must not contain more than this many items. - ***(Optional)* continuationToken: string** If an implementing system supports paging, then the calling system will provide the continuationToken as input instead of the search query.
Output	Content-type: application/json { "success": "true", "version": "1", "totalCount": number, "page": number, "continuationToken": string,

	"resources": [<resource- descriptor>] } - **resources** the array of resources may contain more or less items than requested. It's up to the implementation to decide if it wants to use exact or approximate page sizes. The only indication the end of the search is reached is if the result contains zero items. - **totalCount** may be -1 in case it's not supported, otherwise it's the number of potential search results for the query, allowing the caller to provide an indication of the number of pages of results that are available. If it's -1 then the only indication for having reached the end of the search results is when an empty page is returned.
Description	Searches the customer CMS for resources that best match the specified query. It's up to the implementing system to decide how this query is interpreted and which fields and metadata are involved in searching. *Background: This search API is intended to be used by authors only, and is not part of the 'integrated' search options in the end-user facing publication. For version 1 of this API, it's left up to the implementing system how to search, what to search and in which order to return the results.*

Obtaining metadata for a single resource

Method	GET
Endpoint	/api/resources/metadata
Headers	Authorization: Bearer <jwt-token>
Input	Query parameter: - **resourceId: string** Uniquely identifies the resource for which the metadata is retrieved. This parameter is required and may not be null or empty.

Output	Content-type: application/json
	{
	"success": "true",
	"version": "1",
	"meta": <resource-metadata>
	}

Description	Returns metadata for a single resource. When this method is invoked without any user-claims, the calling system is requesting this metadata for indexing purposes from a background process.

Obtaining content of a single resource

Methods	GET HEAD
Endpoint	/api/resources/content
Headers	Authorization: Bearer <jwt-token>

	It's suggested that the range-request headers are supported by the system, especially when serving larger resources. See https://developer.mozilla.org/en-US/docs/Web/HTTP/Range_requests for more information about implementing range requests.
Input	Query parameter: - **resourceId: string** Uniquely identifies the resource for which the contents are retrieved. This parameter is required and may not be null or empty.
Output	Valid HTTP response for returning the binary resource. When using ranges requests be sure to follow the standard.
Description	This method is the primary method to retrieve the binary contents of a resource. It's also allowed for this method to redirect specific requests to a different resource URL, for example when redirecting directly to a SAS URI for a remotely hosted public resource, but the resource that's redirected to should always be available to the calling system. The following HTTP codes are expected responses for this endpoint: 200 OK 206 Partial Content 307 Temporary Redirect 416 Request Range Not Satisfiable Note that next to GET-requests also HEAD-requests must be supported by the endpoint, and that the HEAD-request response headers must include either 'Accept-Ranges: none' in case ranged requests are not supported, or 'Accept-Ranges: bytes' in combination with a valid Content-Length header, in case ranged requests are supported.

Accessing content

The content-part of this API deals with how structured content is returned. Here implementing systems will need to make decisions on how to return content and may want to transform content from the original format into a format that makes sense to the receiving system

How this translation happens depends entirely on the requested content type. There are over 50 different types of content that can be processed by AskDelphi. The data structures of those content types are defined in the "AskDelphi Data Structures for Imola Publications" document which is available upon request.

Listing sub-folders in virtual folders

Method	GET
Endpoint	/api/content/folders
Headers	Authorization: Bearer <jwt-token>
Input	Query parameter: - ***(optional)* folderId: string** Uniquely identifies the folder of which the sub-folders should be listed. If null , empty or not specified, the request is for the root folder of the virtual directory structure.
Output	Content-type: application/json

	{
	"success": "true",
	"version": "1",
	"folders": [<folder-descriptor>]
	}

Description	Returns the contents of a single virtual folder of topics in the customer CMS system. How folders are organized is left up to the adapter that implements this interface. There is no need for this virtual folder structure to match 1:1 with actual folder structures in the customer CMS. A single interface implementation could even combine multiple sources of resources into a single virtual directory structure this way.

Listing and searching for content

Method	POST
Endpoint	/api/content/search
Headers	Authorization: Bearer <jwt-token>
Input	Body parameter: - **request: <search-request>** The search request information

Output	Content-type: application/json
	{
	"success": "true",
	"version": "1",
	"totalCount": <number>,
	"continuationToken": <string>,
	"topics": [<topic-descriptor>]
	}

Description	Returns the topic-contents of a single virtual folder of topics in the customer CMS system. How folders are organized is left up to the adapter that implements this interface. There is no need for this virtual folder structure to match 1:1 with actual folder structures in the customer CMS. A single interface implementation could even combine multiple sources of resources into a single virtual directory structure this way. The list of resources and folders will be presented to the end-user in the order it is returned by this endpoint. It's strongly suggested to return these lists ordered by name. When using serialized forms of complex structures as identifiers, it's suggested to base64 encode the structures.

Obtaining metadata for a single content object

Method	GET
Endpoint	/api/content/metadata
Headers	Authorization: Bearer <jwt-token>

Input	Query parameter: - **topicId: string** Uniquely identifies the topic for which the metadata is retrieved. This parameter is required and may not be null or empty.

Output	Content-type: application/json
	{
	"success": "true",
	"version": "1",
	"meta": <topic-metadata>
	}

Description	Returns metadata for a single topic. When this method is invoked without any user-claims, the calling system is requesting this metadata for indexing purposes from a background process.

Obtaining the contents of a single topic

Method	GET
Endpoint	/api/content/content
Headers	Authorization: Bearer <jwt-token>
Input	Query parameter: - **topicId: string** Uniquely identifies the topic for which the content is retrieved. This parameter is required and may not be null or empty.

Output	Content-type: application/json
	{
	"success": "true",
	"version": "1",
	"contents": [<topic-content>] // array
	}

Description	Returns all topic contents required for rendering for a single topic. This will have to convert the customer CMS-data into a format that's usable in AskDelphi, as defined in the content specification. Contents must be structured according to the rules for the namespace that's reported as part of the topic list data and the topic metadata. If the topic itself consists of multiple additional 'topic' objects, then these objects may be returned here also, even when these objects are not returned at all by the search function (ever). As an example, suppose we're creating a document with three embedded images and a video. Then the result could be an array of: • The generated primary topic • A "remote-image" topic identifying the image on this system The system's ID can be included in these topics from the system ID that was provided in the claims during log in. See 'Logging in' for more information. • A "remote-video" topic identifying the video resource on this system

	The system's ID can be included in these topics from the system ID that was provided in the claims during log in. See 'Logging in' for more information.

Data types

In the API the following data types are used:

About resource-, folder- and content-identifiers

Resources, folders and content identifiers are always encoded as a string. There are no limits to the contents of this string. Implementing systems should put all information in this ID that they need to find the resource again later.

Although any string is permitted, it makes sense to make this a Base64-encoded string containing some JSON-encoded data structure that makes sense as an identifier to the implementing system.

Note that resource and content metadata does not include a reference to the containing folder anymore. If this information is important for retrieving the resource, this should be encoded in the resource or content identifier by the implementing system.

<api-key>

The login endpoint receives a special authorization token, which is an API key. This should still be prefixed in the Authorization header with the “Bearer” keyword. The API key must be structured as follows: <purpose>:<base64-encoded-value>.

It’s up to the implementing system to decide how it wants to generate API keys and what the values of <purpose> and/or the <base-64-encoded-value> are, as long as <purpose> is a string that does not contain the literal ‘:’, and the <base64-encoded-value> is indeed that, some value after base64 encoding.

<jwt-token>: JSON Web Tokens

A structured string of which the contents comply with the JWT specification as defined in IETF RFC 7519 (https://tools.ietf.org/html/rfc7519)

<refresh-token>

Any string-encode identifier that can be used to allows a user to refresh their JWT token. Refresh tokens should be unique and to be secure they should be handed out randomly from a sufficiently big pool. Using short refresh tokens or using a predictable mechanism to generate refresh tokens poses a security risk because users could guess those tokens.

<iso8601-utc-timestamp>

String containing the a date and time in UTC, expressed according to ISO 8601 using the Zulu time zone, i.e. YYYY-MM-DD**T**hh:mm:ss**Z**. Even though ISO8601 allows for alternative formats, the suggested format should preferably be used.

<folder-descriptor>

An object describing a folder. It is structured as follows:

{	

"folderId":	<string> Should provide enough information for the implementing system to uniquely identify the folder.
"name":	<string> The display name of the folder, as it's shown to the user.
"lastModified":	<iso8601-utc-timestamp> String containing the last modification date and time of the folder.
}	

<resource- descriptor>

An object describing a resource. It is structured as follows:

{	
"resourceId":	<string> Should provide enough information for the implementing system to uniquely identify the resource.
"filename":	<string> The display name of the file, as it's shown to the user.
"lastModified":	<iso8601-utc-timestamp> String containing the last modification date and time of the folder.
"mimeType":	<string> A string defining the mime type of the resource according to [MIMETYPES]
"contentLength":	<number> The size of the referenced document, in bytes. This can be a long integer, i.e. 64-bits, unsigned.
"status":	<string> A short string identifying the status of this document. This is shown in the AskDelphi authoring environment as additional information to help the user select the resource they want.
}	

<resource-metadata>

An object describing additional metadata of a resource. Mainly used for indexing purposes.

{	
"description":	<string> A plain-text description of the resource's contents.
"tags":	Array [<taxonomy-values>, …] Identifies the taxonomies and nodes to which the resource is tagged. It's possible to link these nodes and taxonomies to AskDelphi-defined resources, provided the correct identifiers are used.
"content":	<string> A plain text representation of the resource. Used for indexing purposes only, never displayed to the end-user.
}	

<taxonomy-values>

An object describing a number of tags from a single taxonomy of tags. Ideally the adapter maps the

{	
"taxonomyId":	<string> Unique ID of the taxonomy. If this taxonomy should be linked to the AskDelphi system, this should be a topic Guid of a Hierarchy-type topic in AskDelphi.
"name":	<string> The name of the taxonomy that the nodes belong to.
"values":	Array [<tag-value>, …] List of tag-values representing nodes in the taxonomy to which the item is tagged.
}	

<tag-value>

An object describing a number of tags from a single taxonomy of tags. Ideally the adapter maps the

{	
"id":	<string> An identifier of the tag. If specified must match the identifier of a node in a taxonomy as defined in the AskDelphi authoring environment.
"name":	<string> Display name for the node.
}	

<topic-descriptor>

An object describing a topic list entry. It is structured as follows:

{	
"topicId":	<string> Should provide enough information for the implementing system to uniquely identify the topic.
"title":	<string> The display name of the folder, as it's shown to the user.
"status":	<string> A short string identifying the status of this document. This is shown in the AskDelphi authoring environment as additional information to help the user select the resource they want.
"namespace":	<string> Namespace according to the namespace lookup table.
"type":	<string> Topic type programmatic title, as used in content design of the target project.
"version":	<string> Version string in the form "<major>.<minor>" representing the version of the object as reported earlier in the metadata.
}	

<topic-metadata>

An object describing additional metadata of a topic. Mainly used for indexing purposes.

{	
"description":	<string> A plain-text description of the topic's contents. Used in search results.
"tags":	Array [<taxonomy-values>, …] Identifies the taxonomies and nodes to which the topic is tagged. It's possible to link these nodes and taxonomies to AskDelphi-defined hierarchies, provided the correct identifiers are used.
"indexContents":	<string> A plain text representation of the resource. Used for indexing purposes only, never displayed to the end-user.
}	

<search-request>

{	
"folderId":	<string> *(optional)* Uniquely identifies the folder of which the contents should be listed. If null , empty or not specified, the request is for the root folder of the virtual directory structure. Should not be used in combination with a continuationToken.
"query":	<string> *(optional)* Search query. If specified it's up to the implementing system to filter the returned topics to include only those topics that are considered to match the query. How to match is up to the implementing system. Should not be used in combination with a continuationToken.
"topicTypes":	Array of <string> *(optional)* If specified, the results are filtered to contain only these topic types. Which topic types are supported is defined in the project's content design. Should not be used in combination with a continuationToken.
"tags":	Array of <taxonomy-values> *(optional)* If specified, the results are filtered to contain only topics that are tagged to all of the supplied taxonomy nodes. Should not be used in combination with a continuationToken.
"page":	<number> *(optional)* Page number for which items should be returned. Page numbers start at zero.
"size":	<number> *(optional)* Number of items that's requested per page.
"continuationToken":	<string> *(optional)* If specified the should be the only field in the request, except for page. All other search parameters, including size, are implicit from the continuation token.
}	

<topic-content>

An object describing a resource. It is structured as follows:

{	
"guid":	<guid>

	The AskDelphi topic guid of the topic: this guid will be used inside the AskDelphi software to uniquely identify this content item. The Guid for all topic-content structures returned must be valid and for the lifetime of the content object identified with the topic id string, this guid must remain constant. If two topic content objects have the same guid, they are expected to be the same object, and both are expected to identify the same remote content. For the same content, the guid should remain stable over time. Implementations should guarantee this. There currently is no API call to obtain a topic by its guid; given the 1:1 link betwene a topic ID and the Guid, it is possible such a call will be created later.
"basicData":	<topic-basic-data> Basic information about the topic.
"relations":	<topic-relation-data> Information about the topic's outgoing relations to other content.
"topicId":	<string> ID in case the topic should be fetched as stand-alone data. Mostly relevant when multiple topics are returned. If empty, the topic can't be fetched stand-alone.
"content":	<string> The actual content of the topic, the structure of this data is defined in the Imola Data Format Specification for Topic Namespaces [IDFSTN] document. This should be a JSON-serialized version of that object.
}	

<topic-basic-data>

{	
"topicTitle":	<string> *(required)* Plain text string for the title, without any markup.
"topicTitleMarkup":	<string> *(optional)* Optional version of the topic title, with mark-up. Limited to newlines.
"description":	<string> *(required)* Plain text description of the topic.
"modificationDate":	<iso8601-utc-timestamp> Defines the last modification date of the topic.
"version":	<string> *(required)* Version number in the form "<major>.<minor>" where both major and minor are integers. Other types of versions are not allowed.
"topicType":	<string> *(required)* Title of the topic type, as it is in the content design.
"metricsTags":	Array-of-<string> *(required)*

	Array of names of those tags from the project's metrics hierarchy that are relevant for this topic.
"enabled":	<boolean> *(required)* Set to false to disallow navigating to this topic. Set to true otherwise (should always be true).
"isPublished":	<boolean> *(required)* Set to true, always.
"namespace":	<string> *(required)* Namespace for the topic type of this topic. Must match the topicType's configured namespace in the content design.
"isEmpty":	<boolean> *(optional)* Set to true if the body text for this topic is empty. Normally false. Setting this to true causes the topic to be rendered collapsed in lists.
"isDescriptionCalculated":	<boolean> *(optional)* Set to false.
}	

<topic-relation-data>

{	
"keyVisualGuid":	<guid> *(optional)*
"keyVisualVisualization":	<string> *(optional)*
"thumbnailTopicGuid":	<guid> *(optional)*
"references":	<topic-relation-reference> *(required)*
}	

<topic-relation-reference>

{	
"relationTypeKey":	<guid> *(required)* Relation type from the project's content design used to determine at which "pyramid level" the item is, or to see if the target is potentially downloadable] [could be replaced with "Pyramid level" and "isDownloadable", and upon 'import' be replaced with the appropriate values]
"metadata":	Array-of-<key-value-pair> *(optional)*
"sequenceNumber":	<number> *(required)*
"view":	<string> *(required)* Set this to "Default".
"use":	<string> *(optional)* When specifying tags by means of relations, set this to "Tag". In all other cases set this to "Relation"
"targetTopicType":	<topic-type-reference>
"targetTopicNamespaceUri":	<string> (required)
"targetTopicTitle":	
"targetTopicGuid":	
}	

<key-value-pair>

{	
"key":	<string> *(required)*
"value":	<string> *(required)*
}	

Annex 1: Options for deployment

A given in the solution architecture is that the AskDelphi system is a SaaS system that's hosted in the public Azure cloud. This holds for both the AskDelphi Authoring Environment and the AskDelphi Publication, and is regardless of which SSO solution is used for accessing either.

In addition to these two end-user facing systems, a backend is deployed in a private part of the AskDelphi Azure cloud environment. This backend is responsible for background tasks such as publication and/or indexing of content for search purposes.

These systems need to access the AskDelphi Remote Content REST API directly.

Annex 2: References

[MIMETYPES]

N. Freed, A. Melnikov, M Kucherawy – Media types. Published at https://www.iana.org/assignments/media-types/media-types.xhtml.

[RFC7519]

M. Jones, J. Bradley, N. Sakimura: JSON Web Token (JWT). Published at https://tools.ietf.org/html/rfc7519. Updated by RFC7797 (https://tools.ietf.org/html/rfc7797), RFC8725 (https://tools.ietf.org/html/rfc8725). May 2015.

[RFC7233]

R. Fielding, Y. Lafon, J. Reschke - Hypertext Transfer Protocol (HTTP/1.1): Range Requests. Published at https://tools.ietf.org/html/rfc7233. June 2014.

[MDN: HTTP Range Requests]

MDN (mozilla.org): HTTP range requests - HTTP. Published at https://developer.mozilla.org/en-US/docs/Web/HTTP/Range_requests

[IDFSTN]

Imola Data Format Specification for Topic Namespaces. Available upon request from AskDelphi.