LANTMÄTERIET

DOCUMENT VERSION: 1.0 2020-07-03

The API-portal guide

Table of contents

Introduction

Lantmäteriet is using WSO2 API-Manager to expose some of its API:s. Eventually, in the long term, this is the main platform for exposing the API:s that Lantmäteriet is offering.

Through the API portal, users themselves can list and subscribe to API:s that are of interest. They can also get documentation and test an API for a better understanding and get statistics regarding their usage of API:s.

To make an API call an Access Token (also called API key) is used, which is the only authentication method. To use Lantmäteriet's API portal you'll need an account which you order from Lantmäteriet.

The user account and its permissions also control which API:s each user can view and subscribe to in the API portal.

Documentation for the platform:

- WSO2 API-Manager KeyConcepts
- WSO2 API-Manager Wiki

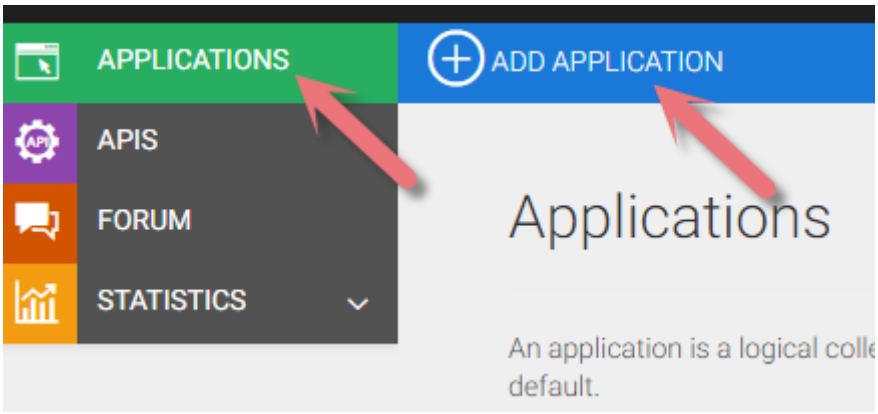
The API portal is available in our two environments:

- Verification environment (VER)
- Production environment (PRD)

Concepts

Applications

Figure 1. Screenshot of how to add an "Application".



Under "Applications" your applications are displayed. Applications let you group a collection of API:s and use the same *Access Token* for all API:s in that grouping.

Your keys and the API:s you subscribe to are grouped by application. There is always a *Default Application* created that you can use but it is appropriate to create and name your own, new application.

Observe: If you, as a user have the applications APP-X, APP-Y and APP-Z which use API:s (one API can be included in several applications) you should create the corresponding logical applications in the API portal i.e. then the keys and subscriptions will be handled separately for each application X, Y, Z.

Keys

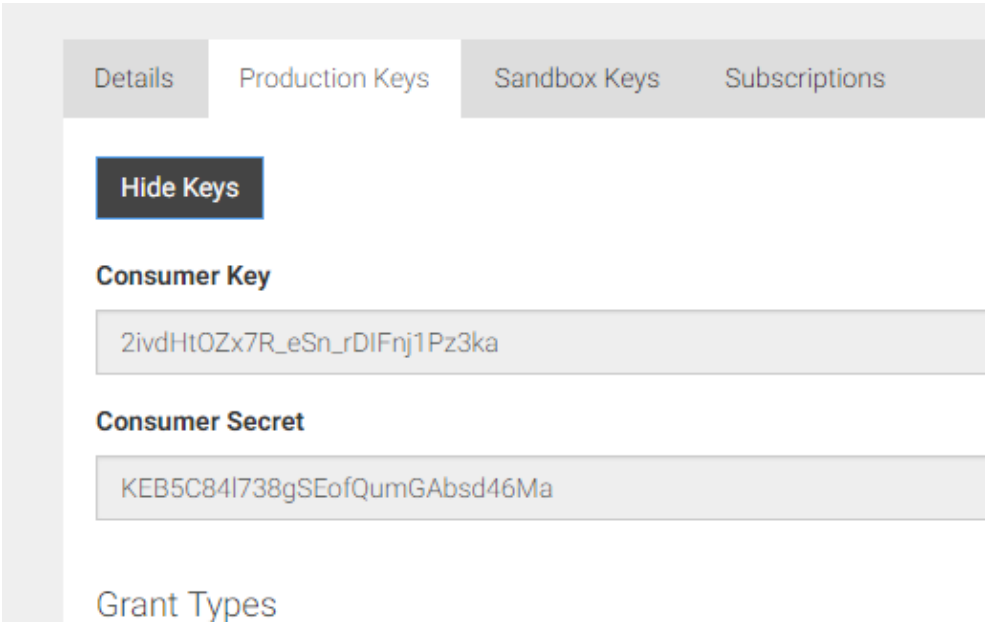
For each logical application (see above) a key pair, *Consumer key* and *Consumer secret*, are generated. These are used to generate an *Access Token* for the application.

These three keys can be viewed in the API-portal. Normally only "Production Keys" are used, but it's also possible to generate "Sandbox Keys" which are used when an API is linked to two backend services where "Sandbox" may point to a technical environment.

That is, it is then possible to control which environment you use based on which *Access Token* is used, provided that an API has a "Sandbox" environment.

CONSUMER KEY OCH CONSUMER SECRET

Figure 2. Screenshot of the "Production Keys" fields.



These keys are unique to each application and are used to generate a new *Access Token*. Read more in section below called Example – Acces Token generation.

Curl-example for generating an *Access Token* using *Consumer key* and *Consumer secret*.

```
curl -k -d "grant_type=client_credentials" -H "Authorization: Basic
<Base64(consumer-key:consumer-secret)>" https://api-ver.lmv.lm.se/token
```

```
{"access_token":"5f996af4-afe6-3438-8e1e-7823018f6898","scope":"am_ap-
plication_scope default","token_type":"Bearer","expires_in":1373}
```

Curl-example of subsequent API call using the generated *Access Token*.

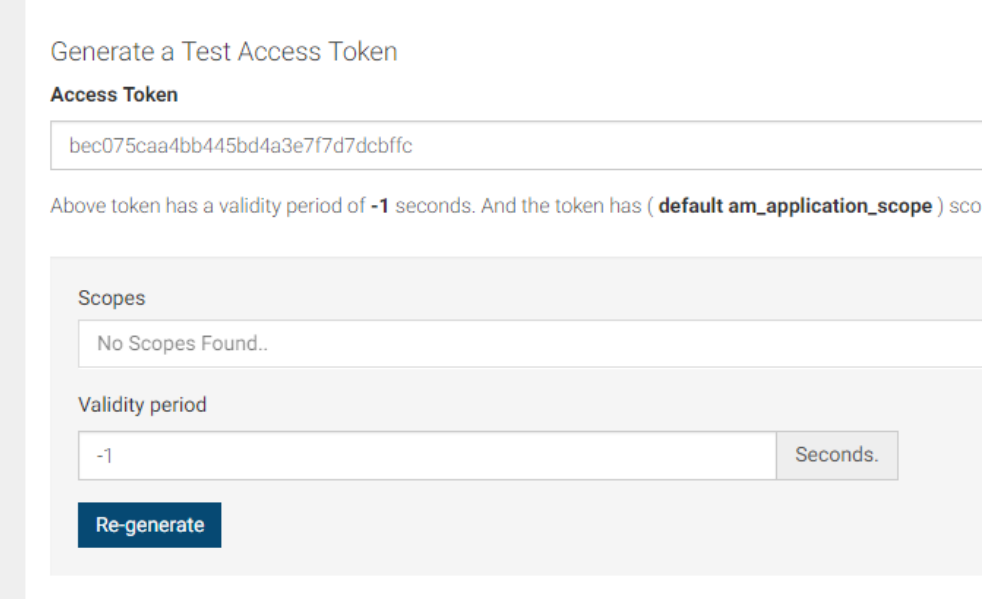
```
curl -X GET --header 'Accept: application/json' --header 'Authoriza-
tion: Bearer 5f996af4-afe6-3438-8e1e-7823018f6898' 'https://api-
ver.lantmateriet.se/hello/2.0/open/test'
```

Curl-example to revoke an *Access Token*.

```
curl -k -d "token=5f996af4-afe6-3438-8e1e-7823018f6898" -H "Authoriza-
tion: Basic Base64Encoded(Consumer key:consumer secret)" https://api-
ver.lmv.lm.se/revoke
```

ACCESS TOKEN

Figure 3. Screenshot of how to generate a "Test Access Token".



This is the key used to make an API call.

Default lifetime of a generated *Access Token* is 3600 seconds (1 hour).

In the user interface it is possible to generate a new *Access Token* which can be fine at a test stage but it is preferable if it's automatically generated from your application.

If you want to use a static key you can set *Access token validity period = -1*, however, we don't recommend this but it may be fine from a test point of view and possibly in legacy applications where there are limited ways in which to dynamically generate keys.

SCOPE

In the curl example below on how to generate *Access Token* the *scope* parameter can be specified. It doesn't have to be but if the API:s resources are protected by a *scope* (read more in Protected with extra authentication/roles) you have to request an *Access Token* for the current *scope*. In the response you will see the *scope* for which the *Access Token* is valid.

```
curl -k -d "grant_type=client_credentials&scope=write" -H "Authoriza-
tion: Basic <Base64(consumer-key:consumer-secret)>" https://api-
ver.lmv.lm.se/token


{"access_token":"5f996af4-afe6-3438-8e1e-
7823018f6898","scope":"write","token_type":"Bearer","expires_in":1373}
```

Scope may also be used to differentiate between various client instances using the same credentials (*Consumer key* and *Consumer secret*). This is sometimes needed as problems may occur while generating *Access Tokens* from different instances simultaneously.

Note! If you use *Scope* to distinguish between clients where they get different *Access Token* though they use the same credentials, the *scope* must have the prefix: **device_** e.g. device_instance1.

It is possible to specify more than one *scope*, to do that you separate them using a blank space. E.g.

```
curl -k -d "grant_type=client_credentials&scope=write device_instance1"
-H "Authorization: Basic <Base64(consumer-key:consumer-secret)>"
https://api-ver.lmv.lm.se/token
```

Example - create application and subscription

1. Log in to the API-portal with your user account.
2. Create an application to group your API:s, read more under section Applications.
3. Click on the API you want to subscribe to and select "subscribe", it is important to specify which application the subscription refers to.

Figure 4. Screenshot of how to subscribe to an "Application".

4. Go to your subscriptions under "Applications" and select the application for which you want to list your keys and API:s. If *Access token* is not generated for your application you can generate it in the user interface. The *Access Token* itself can be copied and used directly but preferably only for testing.
   It is strongly recommended that *Access token* is generated from your code and from your application keys, *Consumer key* and *Consumer secret*, read more about *Keys*.
5. You can also test an API via the "API Console" tab. More about it can be found on WSO2's wiki under Invoke an API using the Integrated API Console.

General information about subscribing to an API can be found on WSO2's wiki under Subscribing to an API.

Example – Access Token generation

When calling an API, an *Access Token* must be used and preferably it should be dynamically generated from your application. Used for this is the OAuth2 specification.

According to OAuth2 there are several ways to do this and below are two descriptions when the ClientCredentials flow is used in Java-based clients.

Addresses used to generate a new Access Token:

- VER: https://api-ver.lantmateriet.se/token
- PRD: https://api.lantmateriet.se/token

Example 1 - generate Access Token with Spring OAuth2

Spring has developed a library that supports, among other things OAuth2 clients. If you use it, *Access Token* will be automatically generated based on the life span of the token.

```
<!-- https://mvnrepository.com/artifact/org.springframework.secu-
rity.oauth/spring-security-oauth2 -->
<dependency>
   <groupId>org.springframework.security.oauth</groupId>
   <artifactId>spring-security-oauth2</artifactId>
   <version>2.1.0.RELEASE</version>
</dependency>
```

When making a machine-to-machine API call it's common to use an OAuth2-flow called *ClientCredentials*.

1. Create a Spring-bean of type *OAuth2RestTemplate* with a bean of type *ClientCredentialsResourceDetails*. Enter your applications Consumer key and Consumer secret as Credentials, read more about it under Keys.

```
@Bean
public OAuth2RestTemplate createRestTemplate(ClientCredentialsResource-
Details resource) {
   return new OAuth2RestTemplate(resource);
}

@Bean
public ClientCredentialsResourceDetails createClientCredentialsRe-
sourceDetails() {
   ClientCredentialsResourceDetails resource = new ClientCredentialsRe-
   sourceDetails();
   resource.setClientId(clientId);
   resource.setClientSecret(clientSecret);
   resource.setAccessTokenUri(accessTokenUri);
   resource.setScope(scope); // Om scope används
   return resource;
}
```

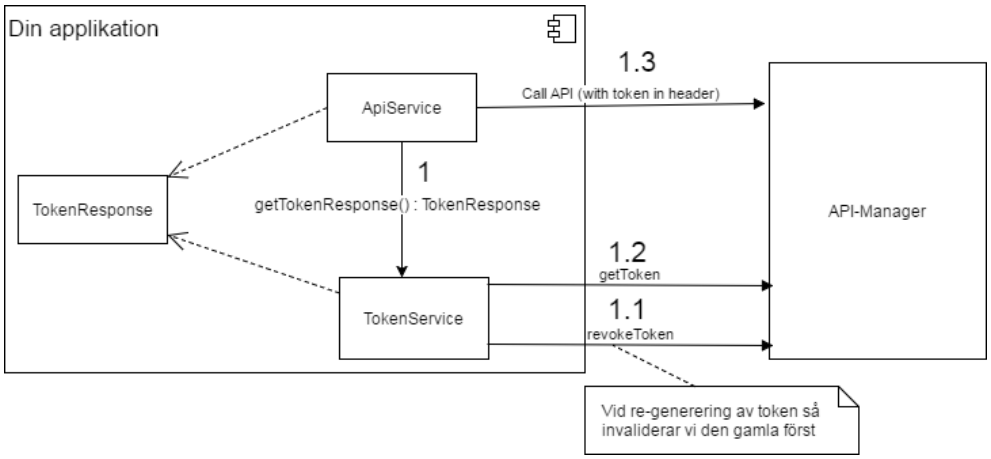
2. Autowire the *OAuth2RestTemplate* you just created and start calling the API:s, *Access Token* are regenerated automatically when it's life span ends.

```
@Autowired
private OAuth2RestTemplate restTemplate;

public void callAPI() {
   try {
      apiCall();
   }
   // Fallback if unauthorized token, should only happen if accesstoken
   is
   // by misstake manually re-generated from other places e.g API-Store
   GUI
   catch (HttpClientErrorException e) {
      if (HttpStatus.UNAUTHORIZED.equals(e.getStatusCode())) {
         logger.warn("Unauthorized, genererar ny nyckel");
         restTemplate.getOAuth2ClientContext().setAccessToken(null);
         apiCall();
      }
   }
}

private void apiCall() {
   logger.info("AccessToken: " + restTemplate.getAccessToken());
   logger.info("AccessToken scope: " + restTemplate.getAc-
   cessToken().getScope());
   logger.info("AccessToken expires at: " + restTemplate.getAc-
   cessToken().getExpiration());

   String result = restTemplate.getForObject(API_URL, String.class);
   logger.info("Svar från API-anrop: " + result);
}
```

Example 2 - generate Access Token DIY

The following example shows how to dynamically generate an *Access Token* from your application (server-side) with an OAuth2 flow called Client-Credentials.

Figure 5. Flow chart of how to dynamically generate an "Access Token".



The example is based on Java and Spring Framework.

- We have a class *Token Service* that handles regeneration of an *Access Token* using *Consumer key*, *Consumer secret* (these should be kept protected in the application).
- In *Token Service* the *Consumer key*, *Consumer secret* is injected from a properties-file.
- A help object, *Token Response* is used as an information carrier.
- *Api Service* makes the call to any API with *Access Token* from *Token Service* and *Token Response*.

TOKEN RESPONSE

The information carrier of *Access Token* information

- *access_token* - the API-key
- *expires_in* - specifies in seconds how long *access_token* is valid
- *expireAsDate* - specifies in date-format how long the key is valid

```
public class TokenResponse {
   private Long expires_in;
   private String access_token;
   private Calendar expiresAsDate;

   public void setExpiresAsDate() {
      expiresAsDate = Calendar.getInstance();
      // Om livstid är satt till oändligt, lägg på 50 år...
      if (expires_in > Integer.MAX_VALUE) {
         expiresAsDate.add(Calendar.YEAR, 50);
      } else {
         expiresAsDate.add(Calendar.SECOND, expires_in.intValue());
      }
   }
...
   // Dra bort 3 sek för att få lite marginal
   public void setExpires_in(Long expires_in) {
      this.expires_in = expires_in - 3;
   }
   public Calendar getExpiresAsDate(){
      return expiresAsDate;
   }
...
}
```

TOKEN SERVICE

Generates a new *Token Response* via REST calls to the API-manager. *Token Response* is cached and generated only if it is timed out. Before regenerating a new *Token Response* we invalidate the old *Access Token*.

We inject properties and create *client Credentials*, a base64-string based on *consumer Key* and *consumer Secret*.

```
@Service
public class TokenService  {
   @Value("${host}")
   private String host;

   @Value("${consumerKey}")
   private String consumerKey;

   @Value("${consumerSecret}")
   private String consumerSecret;

   @Autowired
   protected RestTemplate restTemplate;

   private String clientCredentials;
   private TokenResponse tokenResponse = null;

   @PostConstruct
   public void initCredentials() throws Exception {
      clientCredentials = "Basic " + Base64Utils.encodeToString((con-
      sumerKey + ":" + consumerSecret).getBytes());
   }

   public AccessToken getAccessTokenObject() {
      if (accessToken != null && Calendar.getInstance().after(ac-
      cessToken.getExpiresAsDate())) {
         revokeToken(accessToken.getAccess_token());
         accessToken = getTokenByClientCredentials();
      } else if (accessToken == null) {
         accessToken = getTokenByClientCredentials();
      }
      return accessToken;
   }

   /**
```

```
* curl -k -d "grant_type=client_credentials" -H "Authorization:
Basic Base64Encoded(Consumer key:consumer secret)"
https://api.lmv.lm.se/token

*/

private TokenResponse getTokenByClientCredentials() {

   // Set header Authorization = "ConsumerKey:ConsumerSecret" as
   base64

   HttpHeaders headers = new HttpHeaders();

   headers.add("Authorization", clientCredentials);

   // Set params "grant_type=client_credentials"

   MultiValueMap<String, String> vars = new LinkedMulti-
   ValueMap<String, String>();

   vars.add("grant_type", "client_credentials");

   HttpEntity<MultiValueMap<String, String>> request = new
   HttpEntity<MultiValueMap<String, String>>(vars, headers);

   // Call endpoint

   ResponseEntity<TokenResponse> response = restTem-
   plate.postForEntity(host + "/token", request, Token-
   Response.class);

   return response.getBody();

}


/**

* curl -k -d "token=<ACCESS_TOKEN_TO_BE_REVOKED>" -H "Authorization:
Basic Base64Encoded(Consumer key:consumer secret)"
https://api.lmv.lm.se/revoke

*/

public boolean revokeToken(String tokenToBeRevoked) {

   // Set header Authorization = "ConsumerKey:ConsumerSecret" as
   base64

   HttpHeaders headers = new HttpHeaders();

   headers.add("Authorization", clientCredentials);

   // Set params "token=<ACCESS_TOKEN_TO_BE_REVOKED>"

   MultiValueMap<String, String> vars = new LinkedMulti-
   ValueMap<String, String>();

   vars.add("token", tokenToBeRevoked);

   HttpEntity<MultiValueMap<String, String>> request = new
   HttpEntity<MultiValueMap<String, String>>(vars, headers);

   // Call endpoint

   ResponseEntity<String> response = restTemplate.postForEntity(host
   + "/revoke", request, String.class);

   if (response.getStatusCode().is2xxSuccessful()) {

      log.debug("revoke of accessToken: {} was successful", tokenTo-
      BeRevoked);

      return true;
```

```
        }
        log.warn("revoke of accessToken: {} was NOT successful", tokenTo-
        BeRevoked);
        return false;
    }
}
```

APISERVICE

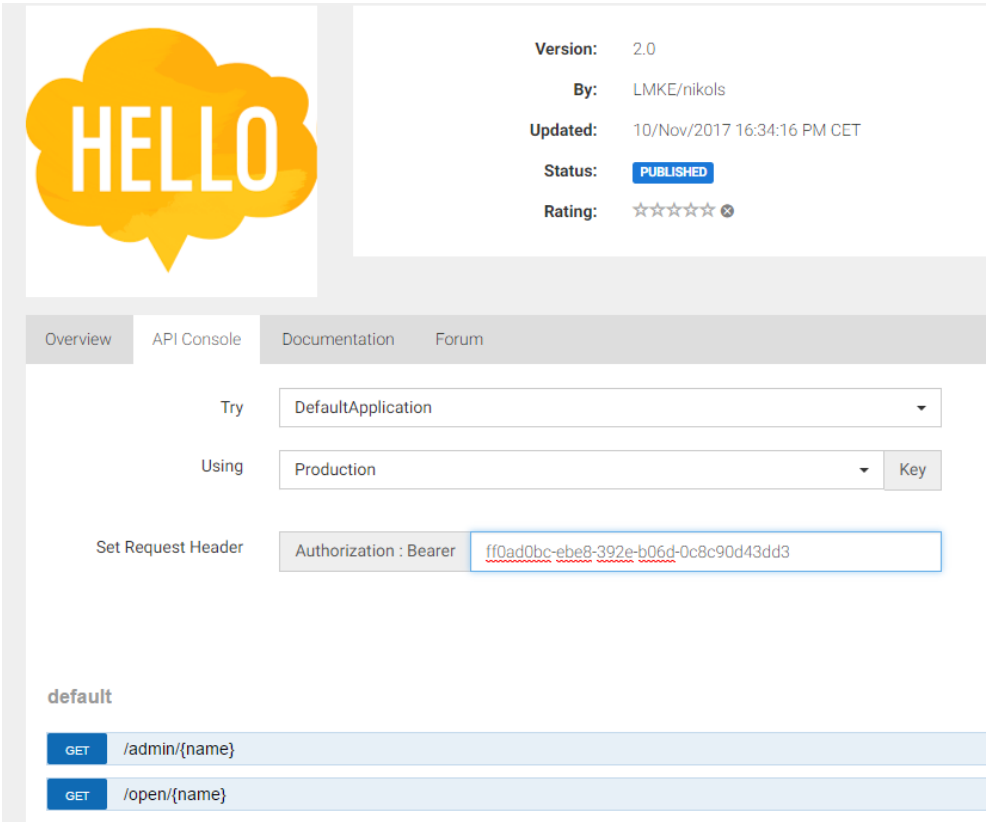
The service that calls the current API with Token from *Token Service*.

```
@Service
public class ApiService {
    private static final Logger log = LoggerFactory.getLog-
    ger(ApiService.class);
    @Value("${host}")
    private String host;

    @Autowired
    protected RestTemplate restTemplate;

    @Autowired
    private TokenService tokenService;

    public void callApi() {
        ResponseEntity<String> apiResponse = null;

        // Get accessToken
        String accessToken = tokenService.getAccessTokenObject().getAc-
        cess_token();

        try {
            // Call API
            apiResponse = callApi(accessToken);
        }
        // Fallback if unauthorized token, should only happen if ac-
        cesstoken is
        // by misstake manually re-generated from e.g API-Store GUI
        catch (HttpClientErrorException e) {
            if (HttpStatus.UNAUTHORIZED.equals(e.getStatusCode())) {
                log.warn("Unauthorized, genererar ny nyckel");
                accessTokenUtils.revokeToken(accessToken);
                accessToken = accessTokenUtils.getAccessTokenString();
                apiResponse = callApi(accessToken);
```

```
            }
        }

        log.info(apiResponse.getBody());
    }

    private ResponseEntity<String> callApi(String accessToken) {
        HttpHeaders headers = new HttpHeaders();
        headers.add("Authorization", "Bearer " + accessToken);
        HttpEntity<String> entity = new HttpEntity<String>(headers);
        return restTemplate.exchange(host + "/hello/2.0/open/Kalle",
        HttpMethod.GET, entity, String.class);
    }
}
```

Demo API – Hello

Figure 6. Screenshot of a public Demo API.



There is a public Demo API (Hello) published which is available for laboration.

Try creating a subscription to it and use the API Console to make test calls to the API or try calling it from your application and try to generate keys from your application first to see if your key-generation works.

Note that the resource /*admin* is protected with a so-called *scope*, i.e. to make a call to that resource an *Access Token* must have been generated for the current *scope*. This means that you, with *scope*, can control extra access on a resource level.

Curl-example for generating *Access Token* using *Consumer key*, *Consumer secret* and *scope*.

```
curl -k -d "grant_type=client_credentials&scope=hello_admin" -H "Au-
thorization: Basic <Base64(consumer-key:consumer-secret)>" https://api-
ver.lmv.lm.se/token
```

```
{"access_token":"5f996af4-afe6-3438-8e1e-
7823018f6898","scope":"hello_admin","token_type":"Bearer","ex-
pires_in":1373}
```

Example for setting a *scope* in the user interface when generating *Access Token*.

Figure 7. Screenshot of scope.