swissmedic

Technical document
swissdamed Machine-to-Machine REST API Documentation

Identification number:	BW630_40_810
Version:	4.0
Valid from:	02.09.2026

Table of Contents

1 Overview 3

2 Disclaimer 3

3 Terms of Use 3

4 Conceptual Guide 4

4.1 REST API and Technical Documentation 4

4.2 API Endpoints 4

4.3 Job Stories 4

5 Onboarding: Testing Machine-to-Machine on the Playground 7

6 Accessing Playground and Production Environment 7

7 Authentication 7

7.1 Obtaining client ID and client secret 8

7.1.1 Client ID for Actors 8

7.1.2 Client ID for Mandates 9

7.1.3 Client Secret for Actors or Mandates 11

7.1.4 Client Secret History 11

7.2 Obtaining a Token 12

8 Versioning 12

9 Backwards Compatibility 12

10 Rate Limiting 12

11 Troubleshooting 400 and 500 Series Error Codes 13

12 Frequently Asked Questions 16

1 Overview

This document is intended to provide guidance for the implementation of Machine-to-Machine (M2M) registration, update and market state management of devices in swissdamed, using a REST API.

Users must set up Machine-to-Machine management of devices in the playground environment prior to using this feature in the production environment.

2 Disclaimer

The availability of the Machine-to-Machine interface may be interrupted due to maintenance.

Processing times for UDI-DIs may vary.

Swissmedic reserves the right to implement a general or client specific rate-limit or temporarily block a client if excess traffic is generated, in order to protect swissdamed system availability for other clients and users. Clients must implement a backoff strategy to handle this type of event.

3 Terms of Use

The current swissdamed Privacy Notice and Terms of Use and Service Agreement apply.

4 Conceptual Guide

4.1 REST API and Technical Documentation

For the integration against the **playground** environment https://playground.swissdamed.ch.
The OpenAPI specification can be found here: https://playground.swissdamed.ch/v3/api-docs/udi-m2m-v1

The OpenAPI specification for the **production environment** can be found here:
https://swissdamed.ch/v3/api-docs/udi-m2m-v1

The business rules and data dictionary also required for the Machine-to-Machine integration are published on the Swissmedic website: Technical documentation.

4.2 API Endpoints

The REST API contains endpoints which enable the following actions in swissdamed:

- API endpoint 1 ***Submit UdiDi***:
  Used for submitting a UdiDi (incl. associated PackageUdiDis) with its related BasicUdi
  (POST /m2m/udi/data/*)

- API endpoint 2 ***GET Status***:
  Used for checking the status of one *Submit UdiDi* operation
  (POST /m2m/udi/data/udi-di-request-status)

- API endpoint 3 Set Market Status:
  Used for setting the market status of a UdiDi and/or PackageUdiDis
  (POST /m2m/udi/data/market-status)

- API endpoint 4 ***Get UdiDi***:
  Used for obtaining the data of a UdiDi (incl. associated PackageUdiDis) and its related BasicUdi, and the current market status.
  (GET /m2m/udi/data/*)

4.3 Job Stories

The following job stories describe how the endpoints listed in Section 4.2 are used to manage BasicUdis and UdiDis, including PackageUdiDis in swissdamed.

Job Story 1 – Register and set market status of new UdiDi with or without PackageUdiDi
Registering a new UdiDi (associated to an existing or new BasicUdi) with or without PackageUdiDis and setting the market status requires the following steps:

- First step - API endpoint 1:
  Use ***Submit UdiDi*** to register a UdiDi (incl. its PackageUdiDi(s)) with its related BasicUdi.

Second step - API endpoint 2:

Use ***GET Status*** to check if the processing of one *Submit UdiDi* operation has been completed. This step can be performed after submitting one UdiDi, or a bulk status request can be done after multiple *Submit UdiDi* operations.

- Third step - API endpoint 3:
  Use ***Set Market Status*** to set the market status of UdiDis and/or PackageUdiDis. This step can be performed after submitting one UdiDi, or a bulk market status set can be done after multiple *Submit UdiDi* operations (see job story 5).

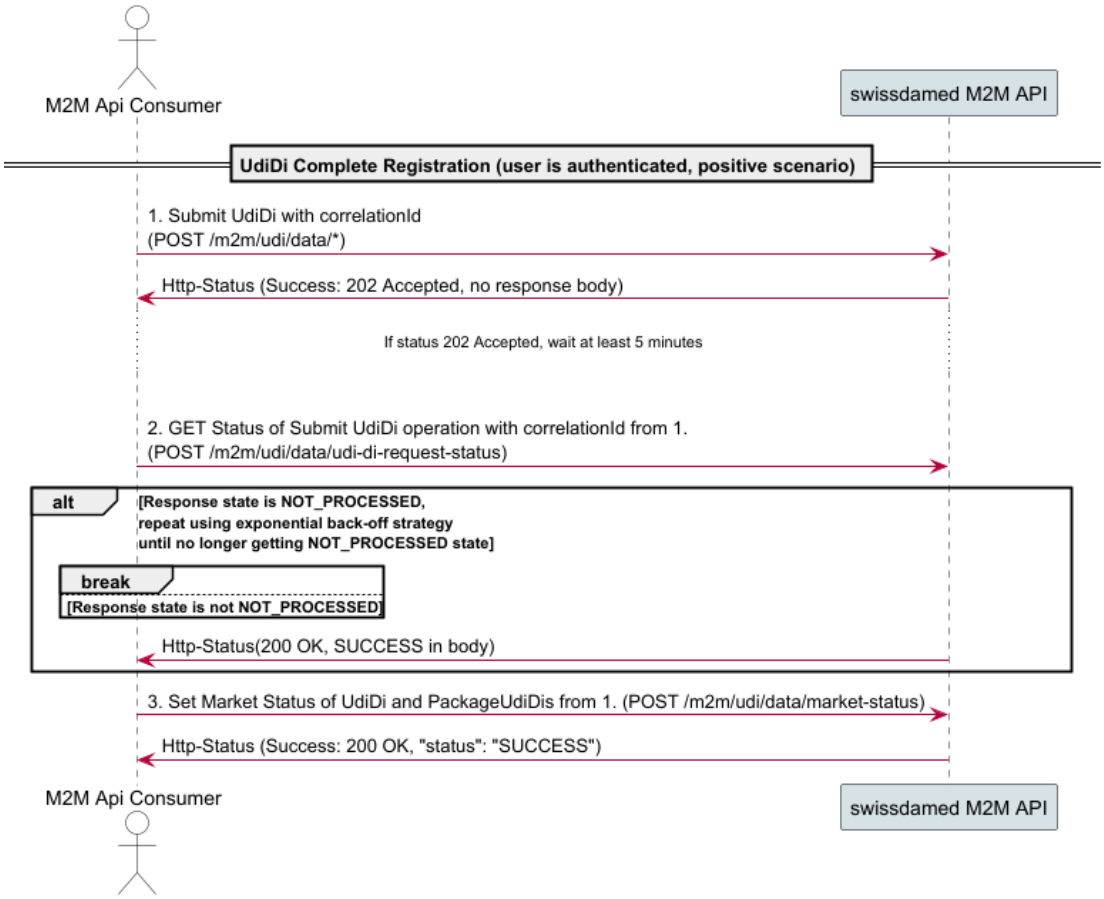


Figure 1: Overview endpoints for job story 1

Job Story 2 – Update a BasicUdi

Updating BasicUdi data in swissdamed requires the following steps:

- First step - API endpoint 1:
  Use ***Submit UdiDi*** to change BasicUdi data. The entire BasicUdi, containing all changed and unchanged data, plus one entire UdiDi (incl. its PackageUdiDis) associated with that BasicUdi, must be submitted**.**

- Second step - API endpoint 2:
  Use ***GET Status*** to check if the processing of one *Submit UdiDi* operation has been completed. This step can be performed after submitting one updated BasicUdi, or a bulk status request can be done after multiple *Submit UdiDi* operations.

Job Story 3 – Update a UdiDi
Updating UdiDi data in swissdamed, without changing the market status, requires the following steps:

- First step - API endpoint 1:
  Use ***Submit UdiDi*** to change UdiDi data. The entire UdiDi (incl. PackageUdiDis), containing changed *and* unchanged data must be submitted, as well as the entire associated BasicUdi**.**

- Second step - API endpoint 2:
  Use ***GET Status*** to check if the processing of one *Submit UdiDi* operation has been completed. This step can be performed after submitting one updated UdiDi, or a bulk status request can be done after multiple *Submit UdiDi* operations.

Job Story 4 – Add new PackageUdiDis to an existing UdiDi
Adding one or more new PackageUdiDis to an existing UdiDi and setting the market status of the PackageUdiDi(s) requires the following steps:

- First step - API endpoint 1:
  Use ***Submit UdiDi*** to add one or more PackageUdiDis to an existing UdiDi. The new PackageUdiDi(s) must be submitted with the entire associated UdiDi (including already existing PackageUdiDis, if applicable), as well as with the entire BasicUdi.

- Second step - API endpoint 2:
  Use ***GET Status*** to check if the processing of one *Submit UdiDi* operation has been completed. This step can be performed after submitting one PackageUdiDi, or a bulk status request can be done after multiple *Submit UdiDi* operations.

- Third step - API endpoint 3:
  Use ***Set Market Status*** to set the market status of the PackageUdiDi. This step can be performed after submitting one UdiDi, or a bulk market status set can be done after multiple *Submit UdiDi* operations (see job story 5).

Job Story 5 – (Bulk) market status change of a UdiDi(s) and/or PackageUdiDi(s)
Changing the market status for one or more UdiDis and/or one or more PackageUdiDis requires the following step:

- API endpoint 3:
  Use ***Set Market Status*** to set the market status of one or more UdiDis and/or PackageUdiDis to ‘On the market’ or ‘No longer placed on the market’.

Job Story 6 – Comparing UdiDi and/or PackageUdiDi data to user’s system
If discrepancies between the data in the user’s system and swissdamed are suspected, the current BasicUdi, UdiDi and/or PackageUdiDi data and market status can be obtained from swissdamed, which requires the following step:

- API endpoint 4:
  Use ***Get UdiDi*** to obtain the UdiDi and/or PackageUdiDi data or market status from swissdamed. This endpoint *should not be used regularly*, as there is no need to verify data and market status after using any of the other API endpoints.

5 Onboarding: Testing Machine-to-Machine on the Playground

To ensure correct implementation of the Machine-to-Machine functionality, it is important for users to test Machine-to-Machine integrations on the swissdamed playground **before** implementing Machine-to-Machine in the swissdamed production environment. This onboarding activity ensures the accuracy of the mapped data elements and prevents costly corrections of device data on the production environment.

The extent of testing should take into account the variability of users' device data and the actors applicable to them.

The following elements should be considered when creating a test plan:

- job stories described in this document
- applicable swissdamed actor types
- applicable devices or device variations (i.e. devices with different data combinations)

The tests should confirm that:

- the UDI data of each device or device variation is correctly uploaded onto the playground environment, i.e. that the data mapping was executed correctly.
- the job stories are executed correctly for all actors and applicable devices or device variations.

There is no formal onboarding approval given by Swissmedic. It is in the responsibility of the user to determine their readiness and their ability to submit data correctly, before making the decision to implement the Machine-to-Machine functionality on the production environment of swissdamed.

6 Accessing Playground and Production Environment

Machine-to-Machine providers needing an account for accessing the playground environment can request this access by opening a ticket on the Swissmedic website under Support.

In the production environment, Machine-to-Machine providers must be onboarded via the respective actors or mandates for whom they are submitting data to swissdamed.

7 Authentication

The swissdamed API uses a OAuth2 Client-Credential-Flow, with a client ID and client secret. When providing data for different actors and/or mandates, a different client ID and client secret is needed for each actor and/or mandate. (Note: The client ID is not the same as the Swiss single registration number CHRN.)

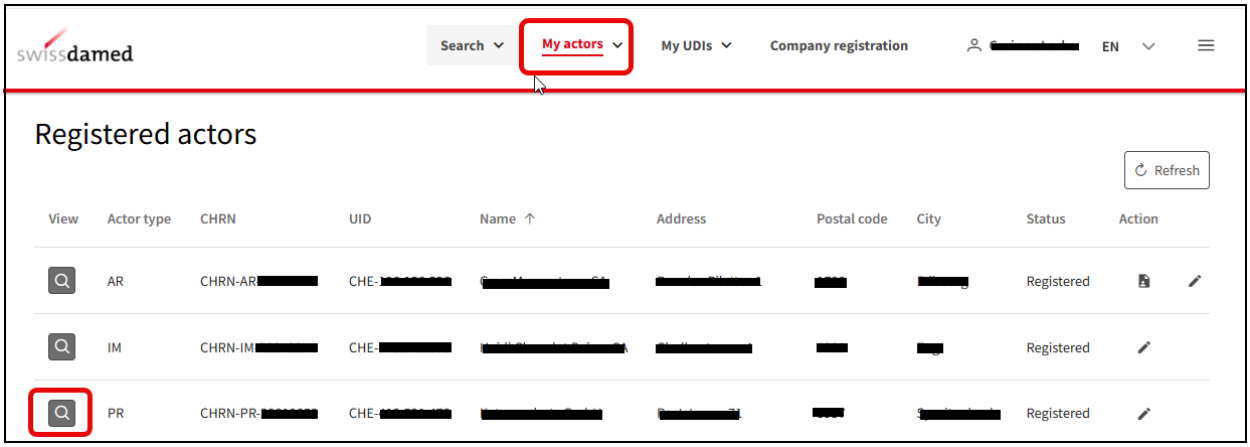
The system does not have an access point configuration preventing a token to be used by multiple users; the API can be used for a given actor or mandate by whomever knowing its credentials. However, it is recommended that the Machine-to-Machine API only be used by one user per actor / mandate to avoid overwriting of data.

7.1 Obtaining client ID and client secret

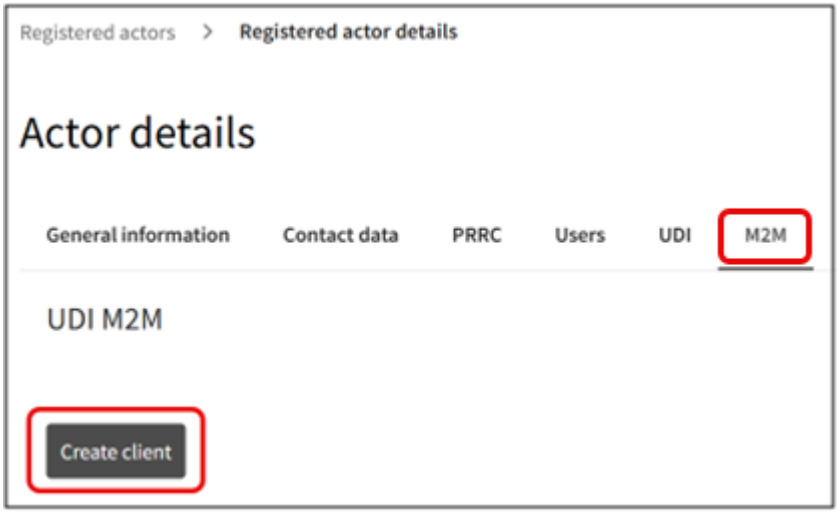
Client IDs and client secrets for actors and mandates can be ordered in swissdamed. Each actor and mandate must order their own client ID and client secret. Different client IDs and client secrets are used for playground and production environments of swissdamed.

7.1.1 Client ID for Actors

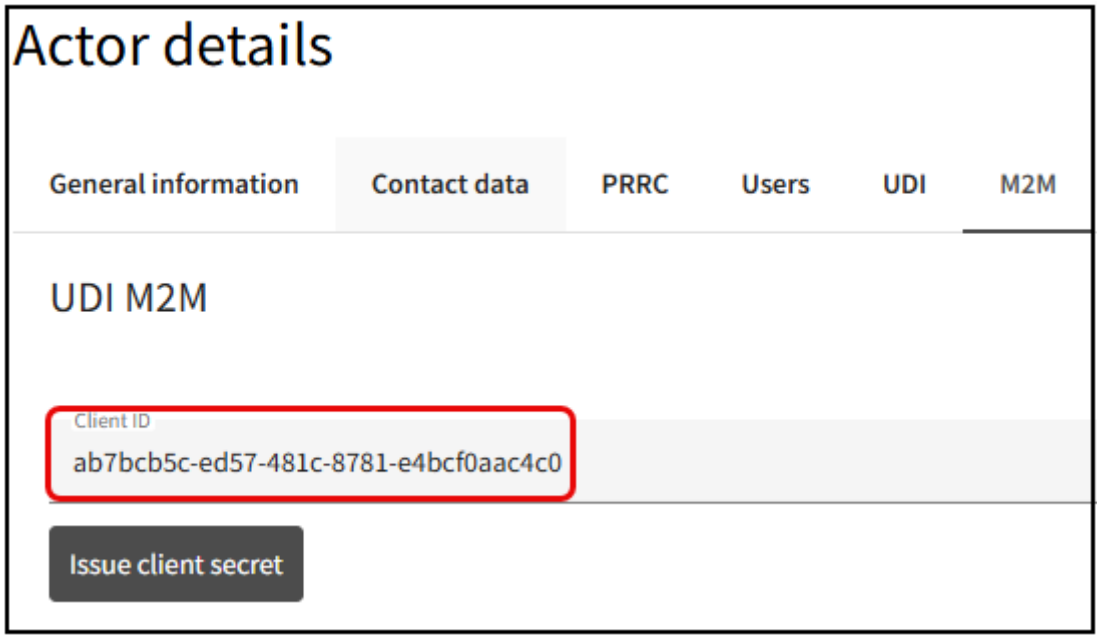
In swissdamed, navigate to “My actors” > “Registered actors”. Click on the magnifying glass symbol of the actor for whom the client ID needs to be created. This opens the “Actor details” section of the actor.



In the “Actor details” section, open the “M2M” tab. In the “M2M” tab click the “Create client” button to create a client ID for the actor.

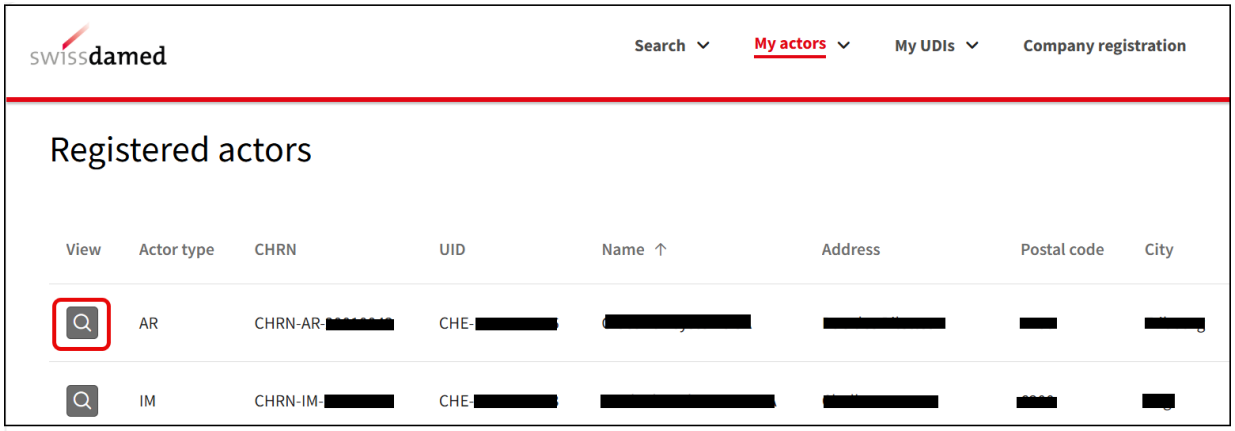


The creation of a new client may take up to 2 minutes. During this time no client ID is showing and no client secret can be ordered.
Once the client ID is created, it is displayed in the “M2M” tab.

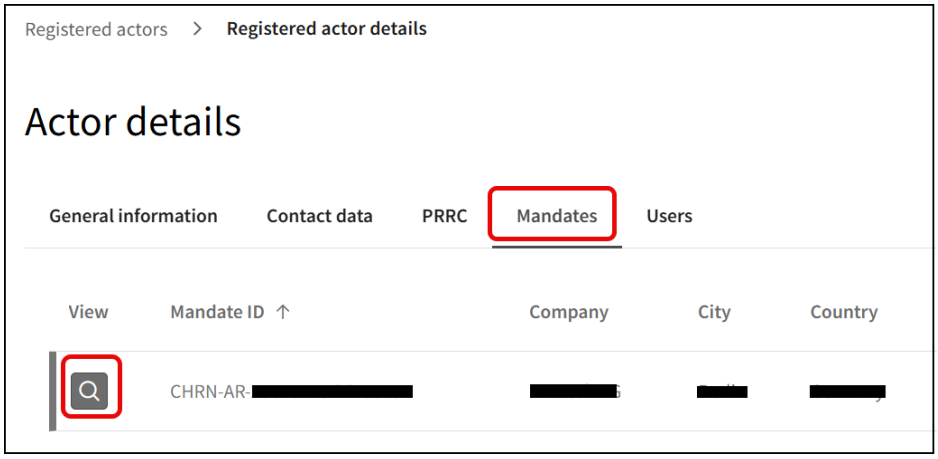


7.1.2 Client ID for Mandates

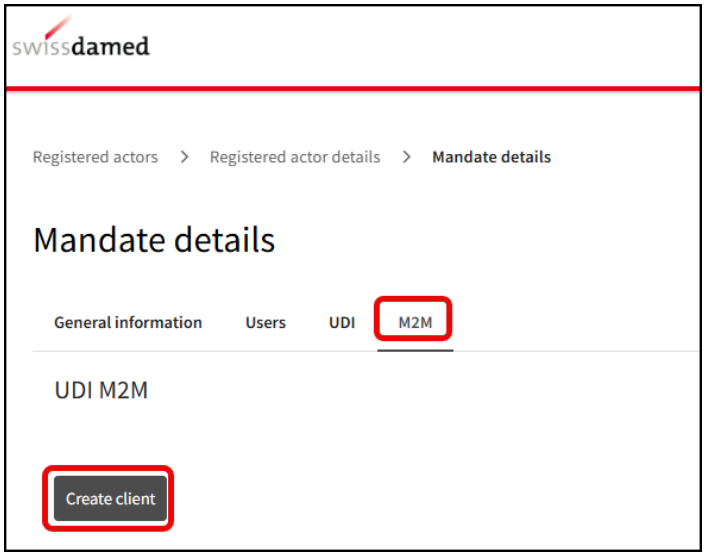
In swissdamed, navigate to “My actors” > “Registered actors”. Click on the magnifying glass symbol of the authorised representative (AR), to whom the mandate is associated.



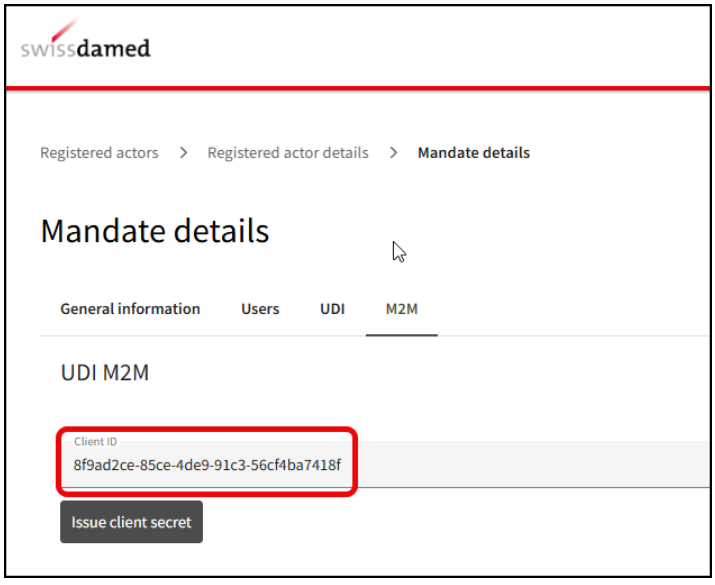
This opens the “Actor details” section of the authorised representative (AR). In the “Actor details” of the AR, open the “Mandate” tab. In the “Mandate” tab click the magnifying glass symbol of the mandate for whom the client ID needs to be created. This opens the “Mandate details” section of the mandate.



In the “Mandate details” section, open the “M2M” tab. In the “M2M” tab click the “Create client” button to create a client ID for the actor.

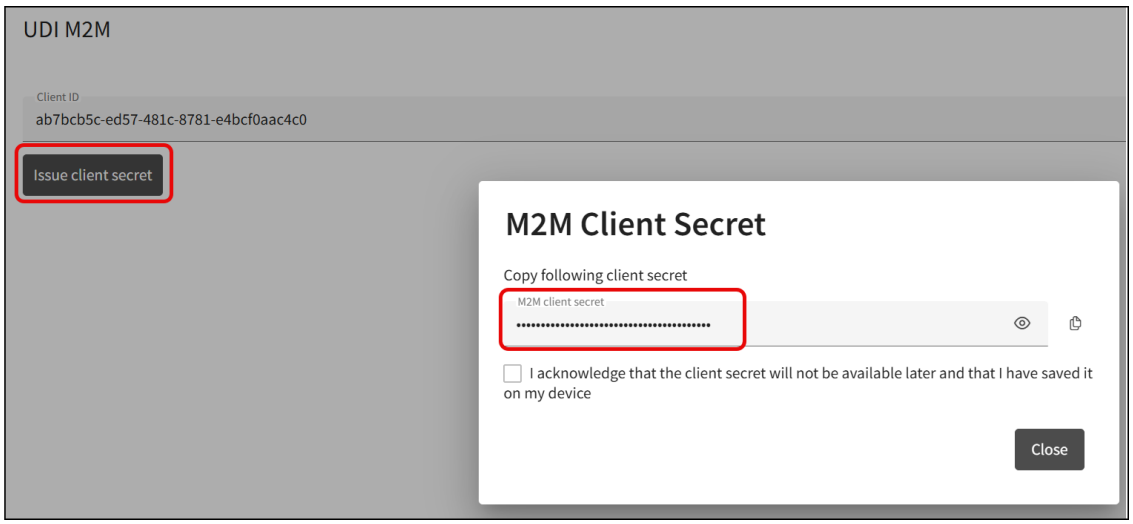


The creation of a new client may take up to 2 minutes. During this time no client ID is showing and no client secret can be ordered. Once the client ID is created, it is displayed in the “M2M” tab.



7.1.3 Client Secret for Actors or Mandates

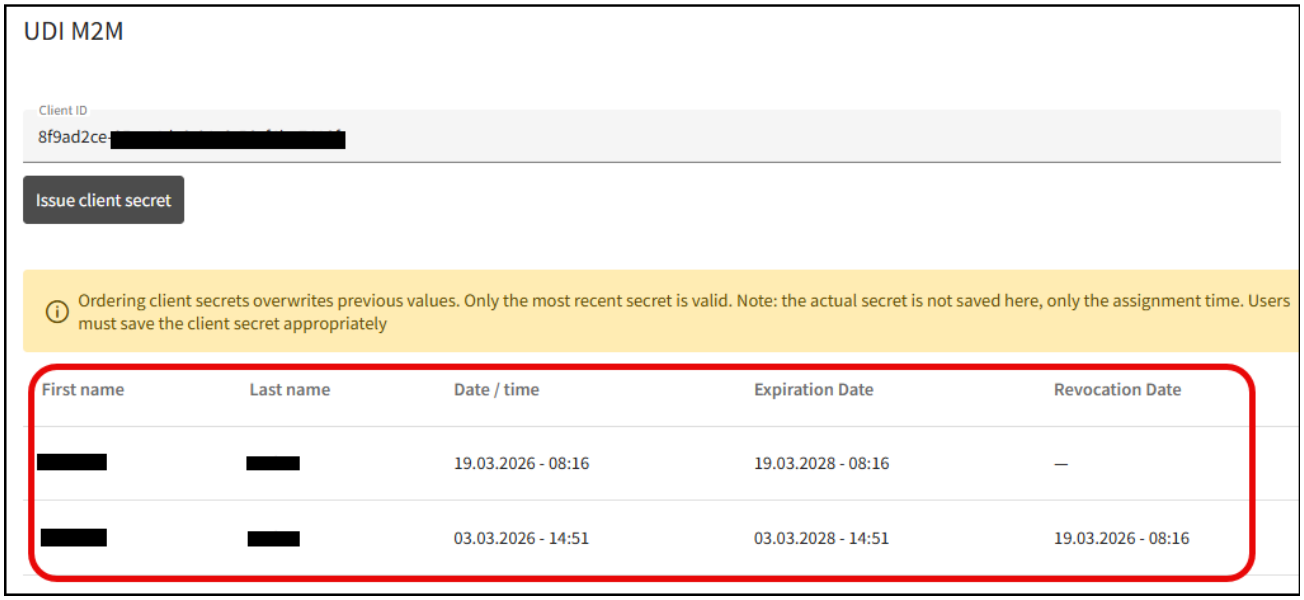
After the client ID is issued, a client secret can be obtained by clicking the “Issue client secret” button below the client ID.
Important note: Make sure to copy and save the client secret, as it will only be displayed at the time it is issued and cannot be viewed at a later time.



If a new client secret is issued, the previous client secret will be invalidated. Client secrets will expire after two years and will then need to be re-generated.

7.1.4 Client Secret History

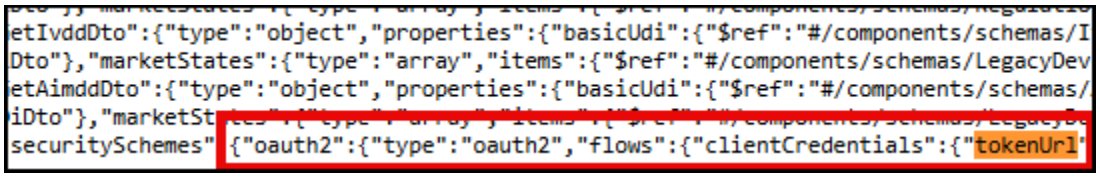
For actors and mandates, a client secret history table is displayed in the “M2M” tab. The table displays the names of the users who obtained the client secrets, the dates when they were issued, and the dates their validity was revoked due to a new client secret being issued. This table does not display the client secret values themselves, as those must be saved by the users upon issuance.



7.2 Obtaining a Token

swissdamed uses tokens to automatically assign actions to the correct actor or mandate. Therefore no additional reference to an actor or mandate is required in the payload.

After obtaining a client ID and client secret, a token can be obtained at the token URL indicated in the OpenAPI specification. The token is generated by the API-user from the client ID and client secret using the client-credential flow.



8 Versioning

The swissdamed Machine-to-Machine API is versioned. If a breaking API change becomes necessary, a new version is published whilst leaving the existing version accepted and backwards compatible until a publicly announced deprecation date, to allow users time to switch to the latest API.

Versioning in this context refers to the version of the Machine-to-Machine interface itself.
For versioning, the URI version approach is used.

9 Backwards Compatibility

The API guarantees to be backwards compatible for each version (for a description of the versioning, refer to Section 8).

Users must allow the following changes on existing versions, without breaking their implementation:

- for new optional fields/elements to be added.
- for fields/elements to not be present or null in case no value is available and the fields/elements are declared optional.

10 Rate Limiting

Users must implement a backoff strategy to handle rate limits as they might change at any time and might depend on activity from other users of the API.

11 Troubleshooting 400 and 500 Series Error Codes

400: malformed request or missing required parameters

A 400 error can be caused by missing fields or unknown field names. Details regarding which field name caused the error are indicated in the “detail” and “raw detail” sections of the of the error’s response body. In order to address the error, it must be ensured that all required fields are present and the field names correspond with the API specification.

Example for an error message indicating a missing field in the payload (in this example, the field “basicUdi.active” was left out):


```
{
    "detail": "Validation failed for argument [0] in public default ResponseEntity<Void> ch.swissmedic.
        swissdamed.infra.udim2UdiMachineToMachineApi.submitMdr(ch.swissmedic.swissdamed.infra.udim2MdrDto):
        [Field error in object 'mdrDto' on field 'basicUdi.active': rejected value [null]; codes [NotNull.
        mdrDto.basicUdi.active,NotNull.basicUdi.active,NotNull.active,NotNBoolean,NotNull]; arguments
        [DefaultMessageSourceResolvable: codes [mdrDto.basicUdi.active,basicUdi.active]; arguments [];
        default message [basicUdi.active]]; default message [must not be null]] ",
    "instance": "/v1/m2m/udi/data/mdr",
    "properties": {
        "errorCode": "InputValidationError",
        "params": []
    },
    "status": 400,
    "title": "Bad Request",
    "type": "about:blank"
}
```

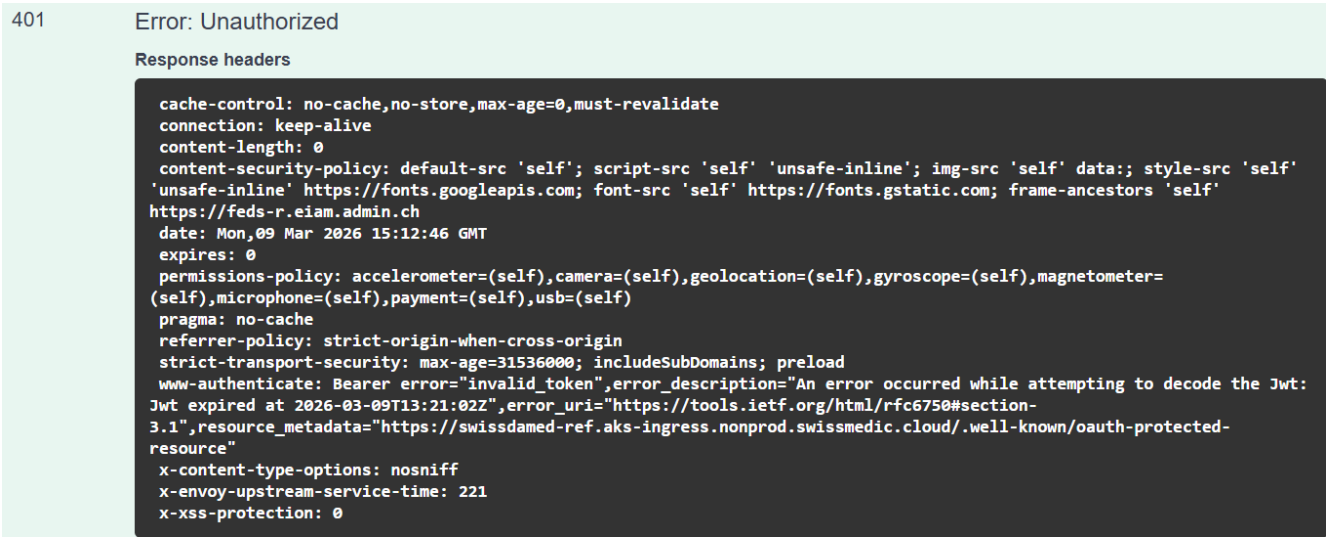

Example for an error message due to an unknown/misspelled field name (in this example a field name was misspelled as “asicUdi” instead of “basicUdi”):


```
{
    "detail": "JSON parse error: Unrecognized property \"asicUdi\" (class ch.swissmedic.swissdamed.infra.
        udim2MdrDto), not marked as ignorable",
    "instance": "/v1/m2m/udi/data/mdr",
    "properties": {
        "errorCode": "MessageParsingError",
        "params": []
    },
    "status": 400,
    "title": "Bad Request",
    "type": "about:blank"
}
```

401: invalid or missing authentication credentials

A 401 error can be caused by invalid credentials, such as a token that has timed out. To address this problem, valid credentials mus be used.

Example for an error message due to a token which is no longer valid:



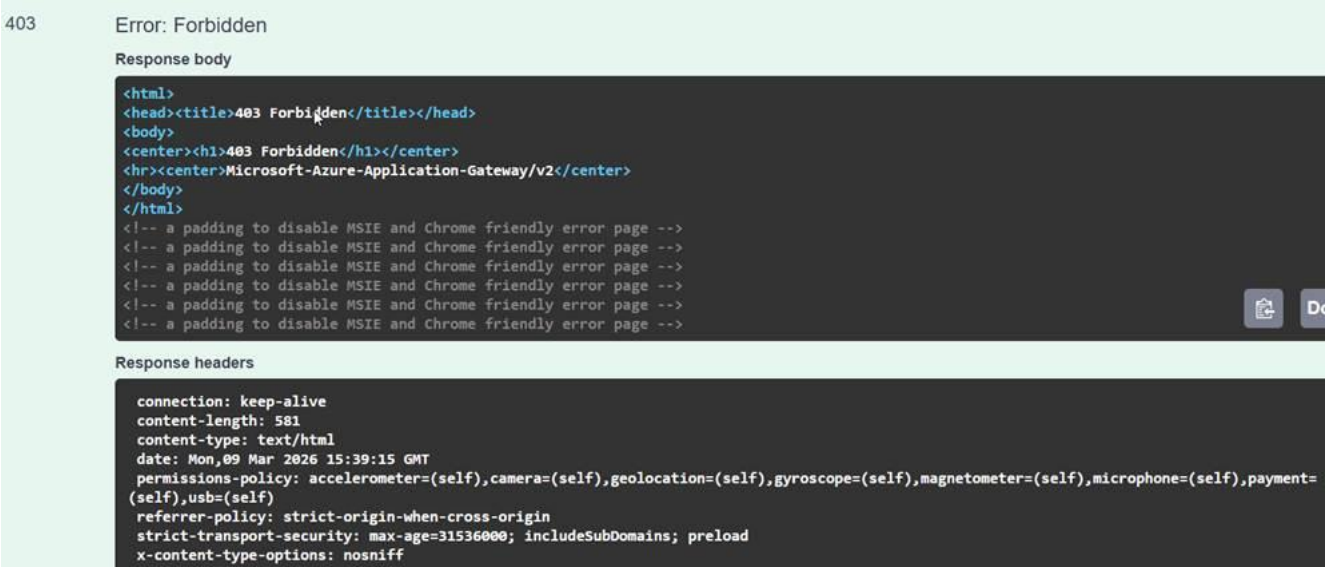
403: access forbidden due to insufficient permissions

A 403 error can be caused by a missing token. To address this problem, a token must be entered.
Example for an error message due to a missing token:

Other causes for 403 errors include:

- The actor/mandate being inactivated
- The mandate was transferred to another authorized representative
- Invalid JSON format like missing brackets
- Detected malicious content in payload
    - In case of false-positives: please open a ticket at Support and describe the actions you took before the error occurred as well as the JSON payload which led to the error

Example for an error message due to incorrect syntax (in this example a comma was placed incorrectly or is missing):



403 errors will not include details about its cause.

409: conflict

A 409 error may occur at various endpoints:

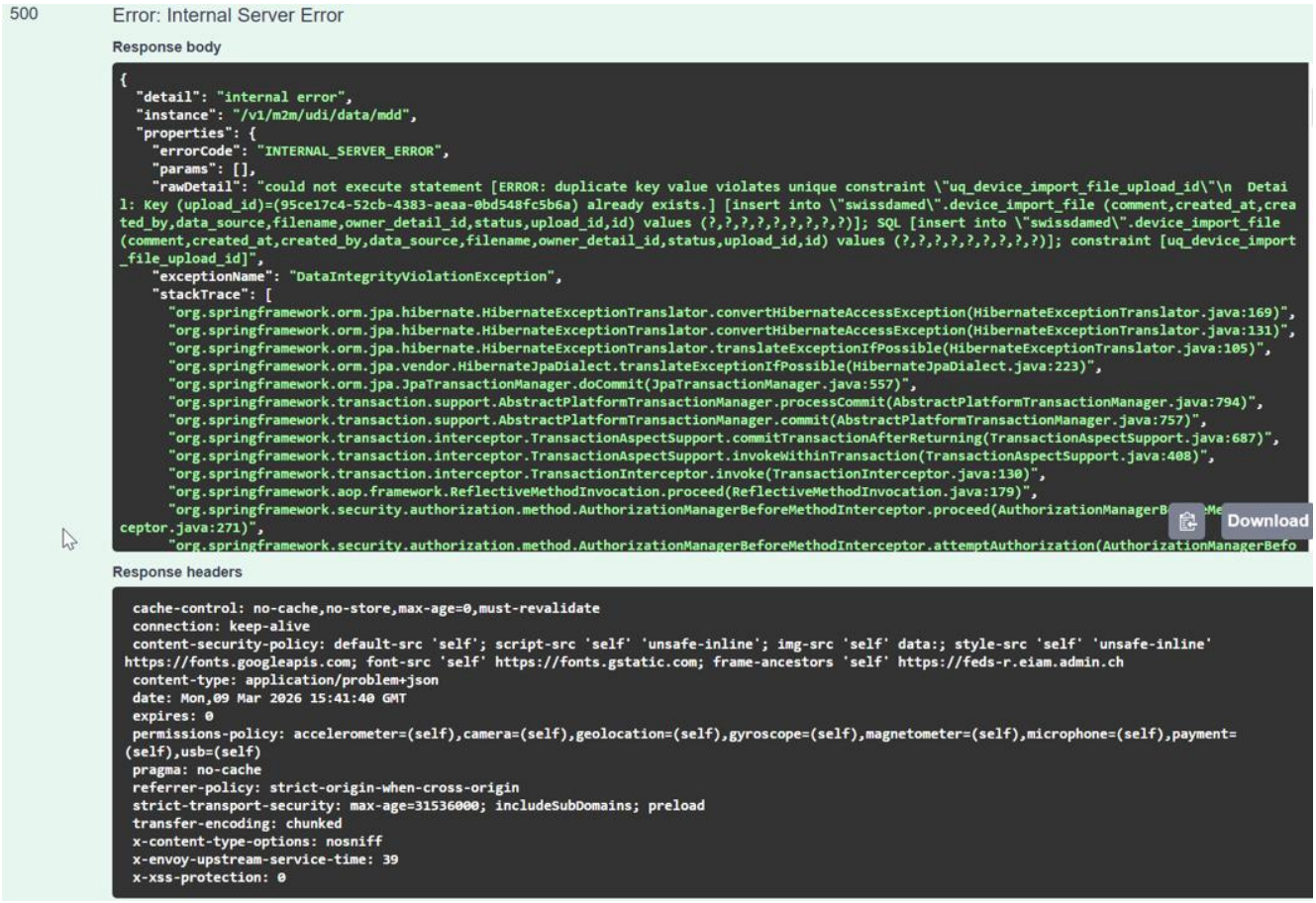
- Submit UdiDi (endpoint 1): The correlationID was already used in a prior message. Use a unique correlationID for each message.
- Set market status (endpoint 3): Devices (Basic UDI-DI or UDI-DI) are being processed at the same time the market status is tried to be set. Retry endpoint 3 and/or send market status with a delay.

429: exceeded rate limits

In order to address the 429 error code, a back-off-strategy must be implemented by the user.

500: unexpeted error on the server

If you encounter a 500 error code indicating an unexpected server error, please open a ticket at Support and describe the actions you took before the error occurred as well as the JSON payload which led to the error.



12 Frequently Asked Questions

Refer to the Swissmedic website for frequently asked questions regarding Machine-to-Machine registration.

Change History

Version	Change	sig
1.0	Initial document	lao
2.0	Update of chapter 5 Authentication Addition of section 6, 11 and 12	lao
3.0	Update of section 1, 2 Update with OpenAPI specification link for the production environment in section 4.1 Update of section 11 with error code 409	enb
4.0	Update of section 11 with a more detailed description of error code 409 - conflict	enb