API

The following sections describe the Live API. This documentation is intended for developers only. Please get in touch with your contact person to activate your access.

General

The application has a programmable interface, called API (Application Programming Interface) for short. With the help of this it is possible to exchange information between the application and third-party providers.

protocol

Communication takes place exclusively via the http protocol and port 443

Security and Clearances

All communication takes place via HTTPS with a SHA-256 RSA encrypted SSL certificate. If your system is installed behind a firewall, the following approvals are required.

- Protocol: http
- Port: 443
- DNS name / A record: kunden-systeme-384614943.eu-central-1.elb.amazonaws.com

Test / productive system

A test system is available for development and test implementation. The functionality of both systems is the same - however, the test system may already contain features that are not yet available in the productive system.

test system:

API backend: https://dev-api.meldescheine.de/api/

Web frontend: https://dev.meldescheine.de/

productive system:

API backend: https://api.meldescheine.de/api/

Web front end: https://v2.meldescheine.de/

Access data for the test and productive system differ from each other.

Switching from test to production system

Since both systems offer the same functionalities, only the following parameters usually need to be adjusted when converting after successful tests:

- API URL
- Access data (username, password)
- fare allocations

HTTP header

The following Http headers are absolutely necessary for the correct transfer in the system

Surname	value	description
content type	application/json or application/xml	Content of the request, accordingly JSON or XML
user agent	any character string	Any character string - ideally you should transfer a unique identifier for your system here, e.g. "mustermann-booking-1.0.4-x86"

If the above headers are not transferred, your requests will be rejected with an HTTP error code 403 "forbidden" .

URL extension

All calls to a URL should end with a slash (/) .

Incorrect:

Generate token
POST https://api.meldescheine.de/api/token

Correctly:

generate tokens
POST https://api.meldescheine.de/api/token/

formats

Character Set / Encoding

Regardless of the format, the data is always transferred and returned in UTF8 code. If your application uses a different character encoding (e.g. ISO-8859-1), the posting data must be converted into the appropriate format before it is transferred.

Validation of requests

All writing requests (e.g. booking a registration form) are validated using a JSON schema to ensure a valid data structure. Corresponding schemes are noted under the respective points of the documentation.

Official documentation and syntax explanation at json-schema.org

info

Requests in a different format (e.g.) are **also** checked via the schemas, but are first converted to JSON internally.

Supported Formats

The interface can be used with different formats. These will be in the explained in the following points and special features are highlighted.

The following formats are supported:

format	code	Used	URL parameters
JSON	json	**default**	?format=json (not necessary as standard)
XML	xml	explicit statement	?format=xml

Call JSON endpoint

```
POST /endpunkt-url #
or explicitly POST /
endpunkt-url?format=json
```

Call XML endpoint

```
POST /endpunk-url?format=xml
```

danger

The `format` parameter must be passed as a URL parameter for both reading (GET) and writing (POST, PUT) if a format other than the standard is to be used.

JSON

Only valid JSON syntax according to the specification can be accepted. Spaces and/or line breaks are not relevant here.

```
{"a":"b","c":"d"}
```

is equivalent to:

```
{
    "a": "b",
    "c": "d"
}
```

Libraries that can be used to create the JSON format can be found in all common programming languages and environments, which simplifies implementation. A list of common libraries is listed at json.org.

Official specification and examples: json.org

XML

The API can also be addressed via the XML format. All API URL endpoints remain, but must be passed with the format parameter (see below)

info

Both transfer and return of the API are transmitted in XML.

URL parameters

In order to use the XML format, it is necessary to use the GET parameter format with the value xml after all calling URLs .

To call up authentication in XML format, the URL is therefore:

```
POST /api/token/?format=xml
```

Special features XML format

root element

XML documents will **always** start with the <root> element and end with </root> accordingly

Example:

XML

```
<root>
    <!-- The actual transfer document follows --> </root>
```

lists/arrays

Array elements are **always** tagged with <list-item>

Example: Transfer of a list of 3 registration forms

XML

```
<root>
    <registration forms>
        <list-item><!-- more data --></list-item>
        <list-item><!-- more data --></list-item>
```

```
            <list-item><!-- more data --></list-item>
        </registration forms>
        <!-- further data -->
    </root>
```

authentication

Before secure queries can be carried out, a valid authentication feature - referred to in the future as a token - is required. The authentication takes place via API call with user name and password key and returns a JWT token if successful.

> The token must be passed on all requests except login.

The following information is required for authentication:

- Server URL
- Username (E-mail
- password

The following information is also required for further calls.

- Community ID(s)
- Object ID(s)

After authentication, this information can be accessed directly via the get_config call.

---

generate tokens

```
POST api /token/
```

Field	data type	Explanation
username	string	Username or e-mail address
password	string	the associated password

special feature

The authentication can also be passed as application/x www-form-urlencoded in addition to being passed as JSON or XML. - better known as form data.

This means that an interactive registration in the system is possible, for example using an HTML form or a third-party application.

examples

request

JSON

```
{
    "username": "m.mustermann",
    "password": "5bed8a17a5f780aa3b541e41c62416a64"
}
```

response

JSON

```
{
    "token": "eJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ...<truncated>"
}
```

use tokens

The token is used with each query in the HTTP header " `Authorization` " passed. Before that in the first Step generated tokens

HTTP Header `Authorization`
Prefix: `JWT`

Example:

text

```
Authorization: JWT eyJhbGciOiJIUzI1NiIeyJ4554dscvd499dfcc....
```

Please pay attention to the prefix JWT and the space between the prefix and the actual token. [PREFIX|SPACE|TOKEN]

Alternatively, the JWT token can also be passed as a GET parameter `" "`: jwt https://.../?jwt=<token>

code samples

Query tokens

Python 3

```
# pip3 install requests import
requests payload =
{ 'username':'demo',
        'password':'demo',

}

r = requests.post('https://<API_URL>/api/token/',data=payload) server_response_json
= r.json()

jwt_token = server_response_json.get('token')
# Extract the string value from "token" from the server_response object print(jwt_token) # <str>:
eJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...
```

Pass token in header

Python 3

```
# pip3 install requests import
requests payload = {

        # Any data
}

# variable jwt_token from previous example headers =
{ 'authorization': 'JWT %s' % jwt_token # Returns "JWT
    eJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..." }



#The header is passed here with r =
requests.post('https://<API_URL>/<request>/',data=payload,headers=headers) server_response_json
= r.json()
# ...
```

meta object

In addition to a matching auth token, a meta object is required for all authenticated requests. This object identifies the query and passes a context for the invocation.

Municipality and object indicate for which municipality as well as which landlord and object of the municipality the call is used .

> The meta object cannot be passed alone. It always belongs to another call (see corresponding chapter)

parameter

Field	data type	needed	description	
local community	internal	true	The ID of the to booking Community	
object	internal	true	The ID of the to booking landlord	
requestId	string	(only if idempotent = true)	An optional Reference to booking	Return: the passed string, or if not passed, a randomly generated one string
idempotent	boolean	false	request Check uniqueness	

Unique requests (idempotence)

In order to prevent multiple postings in the event of connection breaks or other errors, the interface offers the option of transferring the field idempotent in the meta object.

In connection with the requestId it is thus possible to ensure that the system does not allow multiple requests.

The first invocation of an idempotent request is accepted normally - any further request with the same request ID is rejected as long as the system stores this request ID.

Saved request IDs expire after 5 minutes (300 seconds). After expiry, the same request ID can be used again. However, it is always advisable to create a new, randomly generated request ID.

The request IDs are always unique to the user of the request. Thus, user1 and user2 could use the request ID 123-456-789 without getting errors.

All queries containing the meta object support this feature.

The request ID is only saved on the server after a successful process. In the case of parallel requests, there is no guarantee of uniqueness.

Method:

- Generate a unique request id e.g. via a uuid function e.g. (https://docs.python.org/3/library/uuid.html) Pass
- the request ID in the requestId field. Set the
- idempotent property to true

```
{
 "meta": {
  "community":1,
        "object": 2049,
  "requestID": "123-456-789",
  "idempotent": true
 }
}
```

If you use the idempotent feature, the transfer of the requestId field is mandatory.

examples

request

JSON

```
{
    "meta": {
        "object": 806,
        "community": 1,
        "requestId":"abcdefg" // In the response, the same value is returned in "requestId"
be included
    },
    //further data
}
```

XML

```
<?xml version="1.0" encoding="utf-8"?>
<meta>
    <landlord>1</landlord>
    <municipality>1</municipality><!-- in this case, a random character string is used as requestId
returned -->
</meta>
<!-- further data -->
```

response

JSON

```
{
  "meta": {
      "requestId": "66f81f62-0942-11eb-bd68-c04a00212a69",
      "timestamp": "2020-10-08T10:43:56.496102",
      "user": {
                "id": 38,
                "username": "DemoV",
                "alias": "Lisa Mustermann"
            },
            "community": 1
    }
}
```

methods

All implemented methods are defined below. If you need data from the system that is not released via the interface, please get in touch with your contact person.

methods

Book registration forms

```
POST import/registration form/book/
```

This call creates one or more electronic registration forms in the system.

General

It is possible to book multiple registration forms at once by adding more objects to the registration form array.

parameter

Surname	data type	use
meta	objects	meta object
params	objects	Optional parameters
registration forms	array	Array with registration form data
registration forms[]	objects	The dates to be booked registration form
registration forms[].persons	array	List of all guests
registration forms[].ms_type	string	Always with the value "ems" too fill

Per person

Surname	data type	use
registration forms[].persons[].tarif_id	internal	The guest's rate ID
registration forms[].persons[].guest_type	string	Guest type (person, group)

Surname	data type	use
registration forms[].persons[].arrival_dat e	dates	Guest arrival date
registration forms[].persons[].departure_ dates	dates	Guest departure date

single person

Surname	data type	use
registration forms[].persons[].*	string	Other defined fields. (by definition)

The definition of the fields depends on the municipality and can be queried by calling / get_current_user .

tour group

Surname	data type	use
registration forms[].persons[].number	internal	Number of guests

If the field registration forms[].persons[].number is greater than 1, the registration form is booked as a group registration form. Any individual information provided for the guest (first name, last name, etc.) will not be saved.

Optional parameters (params)

Surname	data type	description
append_pdf	boolean	For each registration form object in the response, the parameter "pdf" .

params.append_pdf

This switch makes it possible to attach the PDF directly to each registration form object.
The pdf field containing the Base64-encoded registration forms is added to the server response .

Activating this switch counts as a printing process and is logged accordingly -

see explanation here.

example

```
{
  "meta": {
    "community": 1,
    "requestID": "123-456-789",
    "idempotent": true,
    "object": 2049
  },
  "params": {
    "append_pdf": true
  },
        ....
  }
```

```
{
        "meta": {
                "community": 1,
                "requestID": "123-456-789",
                "idempotent": true,
                "object": 2049,
                "timestamp": "2020-12-18T11:51:41.154820",
                "user": {
                        "id": 38,
                        "username": "DemoV",
                        "alias": "Lisa Mustermann"
                }
        },
        "status": "success",
        "registration forms": [
                {
                        "registration form": {
                                "id": 28,
                           //....
                        },
                        "id": 28,
                        "pdf": "JVBERi0xLjMKMSAwIG9iago8PAov....."
                }
        ]
}
```

server response

If successful, the server response returns an array of registration form objects with the entire registration form object.

> The response structure corresponds to that from the registration form update / get call

> The order of the people is always the same between the request and the answer. This way you can keep the IDs of the people in your system for later updates

Response:

JSON


```
{
    "meta": {
        "requestId": "5ca0b885-11e3-4abd-9e60-cb8265ea9899",
        "timestamp": "2019-10-24T14:06:10.673840",
        "user": {
            "id": 14,
            "username": "api_demo",
            "first_name": "API",
            "last_name": "Demo",
            "email": "api_demo@apidemo.de",
            "is_active": true,
            "user_type": "lodging provider",
            "alias": "API Demo",
            "last_login": "2019-10-24T11:35:18.015947+02:00"
        },
        "community": 1
    },
    "status": "success",
    "registration forms": [
        {
            "id": 396,
            //.....
            "persons": [
                {
                    "id":3334,
                    "tariff_id": 1,
```

```
                "Guest_FirstName": "Max",
                "Guest_Name": "Doe",
                "arrival_date": "2019-01-01",
                "departure_date": "2019-01-15"
            }
        ]
    }
  ]
}
```


Example bookings

Examples of different scenarios

registration form

The following examples each show the posting of a ticket. Since the response from the server is always the same, regardless of the type.

requests:

JSON

```
{
    "meta":
        { "landlord": 1,
        "municipality": 1,
        "object": 1
    },
    "registration forms": [
        {
            "input_id": "test-1", "people":
            [ {

                    "guest_type": "person",
                    "tariff_id": 1,
                    "Guest_FirstName": "TEST_BF",
                    "arrival_date": "2019-07-14",
                    "departure_date": "2019-07-17"
                }
            ]
        }
    ]
}
```

XML

Group registration form

requests:

JSON

```
{
    "meta":
        { "landlord": 1,
        "municipality": 1,
        "object": 1
    },
    "registration forms": [
        {
            "input_id": "test-1", "people":
            [ {

                    "guest_type": "group",
                    "number": 5, "tarif_id": 1,
                    "arrival_date": "2019-07-14",
                    "departure_date": "2019-07-17"

                }
            ]
        }
    ]
}
```

XML

code samples

The following example authenticates the request and posts a registration form

Python 3

```
# $ pip3 install requests
import requests
payload = {
        'username':'api_demo',
        'password':'demodemo',
}

r = requests.post('https://apiv2.meldescheine.de/api/token/',data=payload)
server_response_json = r.json()



jwt_token = server_response_json.get('token') # From the server_response object the string value
extract from "token".
```

```
# Book now
meldeschein_data = {} # Empty for test
headers = {
  'Authorization': "JWT %s" % jwt_token
}
r =
requests.post("https://apiv2.meldescheine.de/api/import/meldeschein/buchen/",data=meldeschein_
data,headers=headers)
```

Update registration forms

Documentation incomplete.

General

This call updates an existing electronic registration form in the system.

Only one (1) registration form can be updated per call.
If multiple registration forms are listed in the array explained below, the first registration form (Index:0) is updated.

Existing registration forms must always have at least one guest. It is not possible to "empty" a registration form by deleting all guests.

restrictions

A registration form can only be changed as long as its status is "Draft" or "Printed". If a registration form has left, no further changes are possible.

"Departed" is defined as follows: All guests on the registration form have departed. An automatic system-level process checks all current registration forms and guests based on their departure date. If all guests have left, the registration form receives the status "departed". The registration form is therefore considered "ready for settlement" for the municipality.

Changes to these registration forms can only be requested through the municipality.

parameter

POST import/registration form/update/

Surname	data type	use

meta	objects	meta object
registration forms	array	Array with registration form data
registration forms[]	objects	The dates to be booked registration form
registration forms[].persons	array	List of all guests

The structure of the request corresponds exactly to the "book" call with the difference that in the registration form object the ID from the "book" call must also be transferred in order to uniquely identify the registration form to be updated.

To update guests, the "id" field must also be transferred

server response

The "update" response contains the updated registration form as a complete object. Regardless of the action taken, the response is always in the same format and reflects the current status of the registration form in the system.

Since only one registration form can be updated at the moment, the updated registration form is located in the object response.meldescheine[0]

The answer returns the current status of the registration form in the system - here you can check whether changes were applied successfully.


```
{
    "meta": {
        "requestId": "afc8673a-2a4c-11eb-b5a3-c04a00212a69",
        "timestamp": "2020-11-19T10:50:41.996029",
        "user": {
            "id": 38,
            "username": "DemoV",
            "alias": "Lisa Mustermann"
        },
        "community": 1
```

```
},
"status": "success",
"response": {
    "registration forms": [
        {
            "id": 688,
            "persons": [
                {
                    "id": 2015,
                    "tariff_id": 1,
                    "registration form_id": 688,
                    "guest_type": "person",
                    "created": "2020-11-19T10:48:47.228504+01:00",
                    "modified": "2020-11-19T10:50:42.311202+01:00",
                    "amount: 1,
                    "arrival_date": "2019-07-14",
                    "departure_date": "2019-07-17",
                    "CSID": "6b59af3c-2a4c-11eb-aa07-c04a00212a69-WLclSDowcn",
                    "tourist_tax_calc": 6.0,
                    "guest_firstname": "updated?",
                    "guest_name": null,
                    "guest_birthdate": null,
                    "guest_street": null,
                    "guest_zip": null,
                    "guest_residence": null,
                    "guest_country": null,
                    "guest_email": null,
                    "Guest_ID_Number": null,
                    "guest_nationality": null,
                    "address_id": null,
                    "address_lat": null,
                    "address_lng": null,
                    "address_label": null
                },
                {
                    "id": 2016,
                    "tariff_id": 1,
                    "registration form_id": 688,
                    "guest_type": "person",
                    "created": "2020-11-19T10:48:47.571441+01:00",
```

```
        "modified": null,
        "amount: 1,
        "arrival_date": "2019-07-14",
        "departure_date": "2019-07-17",
        "CSID": "6b59af3c-2a4c-11eb-aa07-c04a00212a69-WLclSDowcn",
        "tourist_tax_calc": 6.0,
        "Guest_FirstName": "TEST_BF_2",
        "guest_name": null,
        "guest_birthdate": null,
        "guest_street": null,
        "guest_zip": null,
        "guest_residence": null,
        "guest_country": null,
        "guest_email": null,
        "Guest_ID_Number": null,
        "guest_nationality": null,
        "address_id": null,
        "address_lat": null,
        "address_lng": null,
        "address_label": null
    },
    {
        "id": 2017,
        "tariff_id": 1,
        "registration form_id": 688,
        "guest_type": "person",
        "created": "2020-11-19T10:48:47.611475+01:00",
        "modified": null,
        "amount: 1,
        "arrival_date": "2019-07-14",
        "departure_date": "2019-07-17",
        "CSID": "6b59af3c-2a4c-11eb-aa07-c04a00212a69-WLclSDowcn",
        "tourist_tax_calc": 6.0,
        "Guest_FirstName": "TEST_BF_3",
        "guest_name": null,
        "guest_birthdate": null,
        "guest_street": null,
        "guest_zip": null,
        "guest_residence": null,
        "guest_country": null,
```

```
                "guest_email": null,
                "Guest_ID_Number": null,
                "guest_nationality": null,
                "address_id": null,
                "address_lat": null,
                "address_lng": null,
                "address_label": null
            },
            {
                "id": 2018,
                "tariff_id": 1,
                "registration form_id": 688,
                "guest_type": "person",
                "created": "2020-11-19T10:48:47.655957+01:00",
                "modified": null,
                "amount: 1,
                "arrival_date": "2019-07-14",
                "departure_date": "2019-07-17",
                "CSID": "6b59af3c-2a4c-11eb-aa07-c04a00212a69-WLclSDowcn",
                "tourist_tax_calc": 6.0,
                "Guest_FirstName": "TEST_BF_4",
                "guest_name": null,
                "guest_birthdate": null,
                "guest_street": null,
                "guest_zip": null,
                "guest_residence": null,
                "guest_country": null,
                "guest_email": null,
                "Guest_ID_Number": null,
                "guest_nationality": null,
                "address_id": null,
                "address_lat": null,
                "address_lng": null,
                "address_label": null
            }
        ],
        "travel_span": {
            "arrival_date": "2019-07-14T00:00:00",
            "departure_date": "2019-07-17T00:00:00",
            "travel_days": 3,
```

```
            "overnight stays_sum": 12
        },
        "object": {
            "id": 2049,
            "accommodation_id": 13455,
            "lodging": {
                "company": "Ferienhof Müller",
                "firstname": "Melissa",
                "surname": "Miller",
                "comment": "<br>",
                "customer_nr": "08154711",
                "alias": "Ferienhof Müller (08154711)",
                "id": 13455,
                "contact person": 37,
                "municipality_id": 1
            },
            "beherberger_alias": "Ferienhof Müller (08154711)",
            "district": {
                "id": 1,
                "alias": "Default",
                "community": 1
            },
            "created": "2020-03-30T08:34:03.052153+02:00",
            "modified": "2020-07-21T13:16:53.520759+02:00",
            "number_of_beds": 4,
            "alias": "House Sun",
            "strasse": "Edgar-Anstett-Strasse 4",
            "zip code": "66978",
            "place": "Merzalben",
            "tariff zone": 1,
            "accommodation type": 1
        },
        "registration form number": "688",
        "object_id": 2049,
        "accommodation_id": 13455,
        "used_tax": 24.0,
        "creation_user": {
            "id": 38,
            "username": "DemoV",
            "alias": "Lisa Mustermann"
```

```
                    },
                    "precheck_errors": [],
                    "state_display": "Draft",
                    "invoice_date": null,
                    "CSID": "6b59af3c-2a4c-11eb-aa07-c04a00212a69-WLclSDowcn",
                    "form_id": null,
                    "BatchNo": null,
                    "BatchRDate": "2020-11-19",
                    "BatchPgDta": null,
                    "BatchTrack": null,
                    "ms_type": "ems",
                    "state": "draft",
                    "created": "2020-11-19T00:00:00+01:00",
                    "modified": null,
                    "user_note": null,
                    "is_cancelled": false,
                    "source": null,
                    "cancellation_confirmed": false,
                    "tourist_tax_actual": null,
                    "manual_amount": 0.0,
                    "indicator_manual_amount": false,
                    "kurtaxe_calc": 24.0,
                    "stat_age_child_1": null,
                    "stat_age_child_2": null,
                    "stat_age_child_3": null,
                    "stat_age_child_4": null,
                    "invoice": null
                }
            ]
        }
}
```


Creation of additional guests

To add more guests to the registration form, they can be transferred to the "persons" array without specifying the "id" field.

> The rules for creating a registration form apply to the creation of additional guests. The fields "tarif_id" , "arrival_date" and "departure_date" are mandatory. Additional fields analogous to the book call and the community set up.

Example request: Creating an additional guest:

"John Doe"
Tariff ID "1" (adult)
Travel period: October 15, 2020 - October 16, 2020 in registration form 650

```
{
    "meta": {
        "object":2080,
        "community":1,
   "requestId":"TEST"
    },
    "registration forms": [
        {
     "id": 650,
     "persons":[
       {
         "arrival_date": "2020-10-15",
         "departure_date":"2020-10-16",
         "tariff_id":1,
         "Guest_FirstName":"Max",
         "Guest_Name":"Doe"
       }
     ]
   }
    ]
}
```

Updating existing guests

In order to update guests, for example because travel dates have changed, it is necessary to transfer the "id" of the guest.

This can be done, for example, via the "get-data" can be called up or saved immediately when booking the registration forms.

Example: Changing the guest "Max Mustermann" created in the previous example "Creating additional guests".

In this case, the travel period will be changed: 10/16/2020 to 10/19/2020 as well as the last name of the guest.

Only fields that are passed and exist in the system are updated.
Therefore, please make sure that you only transfer the necessary fields and make sure that the field names are correct

Note the guest's ID "1945" in the example below.
- If the ID is specified: Update guest - No specification of
the ID: Create guest

```
{
    "meta": {
        "object":2080,
        "community":1,
    "requestId":"TEST"
    },
    "registration forms": [
        {
     "id": 650,
     "persons":[
      {
        "id":1945, //<---------- ID of the guest (e.g. retrieved via /get-data
        "arrival_date": "2020-10-16",
        "departure_date":"2020-10-18",
        "tariff_id":1,
        "Guest_FirstName":"Max",
        "Guest_Name":"Doe (changed)"
      }
     ]
    }
    ]
}
```

Delete existing guests

Note on updating guests:
Please always use the "update" process to change guests. Please do not delete all guests and create them again.

This is technically possible in a combined call (see below) - but this makes it more difficult

traceability at system level, since a history is created for each guest.

Guests can be deleted by passing them along with their ID and the "delete:true" field.

Example request: Deletion of the guest with id 1937 from the registration form with ID 651

```
{
    "meta": {
        "object":2080,
        "community":1,
    "requestId":"TEST"
    },
    "registration forms": [
        {
     "id": 651,
     "persons":[
      {
        "id": 1937,
        "delete":true //<------ Specifies that the guest should be deleted on invocation
      }
     ]
    }
    ]
}
```

If you try to delete a guest that is not assigned to the ticket, you get a corresponding error message

Various guest promotions in one call

It is possible to combine the actions described above in one call:

Example:

A combination of the above examples

- Guest 1945 update
- Delete Guest 1937

- Create a new guest


```
{
    "meta": {
        "object":2080,
        "community":1,
    "requestId":"TEST"
    },
    "registration forms": [
        {
     "id": 650,
     "persons":[
       {
        "id":1945,
        "arrival_date": "2020-10-16",
        "departure_date":"2020-10-18",
        "tariff_id":1,
        "Guest_FirstName":"Max",
        "Guest_Name":"Doe (changed)"
       },
        {
        "id": 1937,
        "delete": true
       },
       {
        "arrival_date": "2020-10-15",
        "departure_date":"2020-10-16",
        "tariff_id":1,
        "Guest_FirstName":"Max",
        "Guest_Name":"Doe"
       }

     ]
    }
    ]
}
```

Retrieve registration form data

This call queries all stored data of one or more registration forms.

```
POST import/registration form/get/
```

parameter

Surname	data type	use
meta	object	meta object
registration forms	array	Array with registration form IDs

examples

The following example retrieves the data from two registration forms.

request

JSON

```
{
    "meta": {
      //....
    },
    "registration forms": [
        123,
        456
    ]
}
```

response

JSON


```
{
    "meta":
        { "requestId": "0324b440-c0ec-4828-a13c-5b1b81797d78", "timestamp":
        "2019-06-13T11:55:26.039028", "user": { "id": 6, "username ":
        "demo_hotel" }, "municipality": 1




    },
    "response": [ {

            "id":123,
            "people": [ {

                    "tariff_id": 1,
                    "Guest_FirstName": "Max",
                    "Guest_Name": "Doe", "arrival_date":
                    "2019-01-01", "departure_date":
                    "2019-01-15"
                }
            ]
        },
        {
            "id": "456",
            "people": [ {

                    "tariff_id": 2,
                    "number": 12
                }
              //...
]
        }
    ]
}
```

Registration form spa card retrieve (PDF)

Returns the spa cards of the requested registration forms as a base64 encoded PDF. The spa card is ready to print and usually only needs to be output as a PDF file and printed.

Avoid calling the interface multiple times: Each "get-pdf" request is evaluated as a print process and logged accordingly.

It is therefore considered to be the physical print of a spa card. Therefore, please make sure that you only carry out the request if the guest card is printed immediately.

If the PDF spa card is required more than once, please save the PDF on your system for future use.

parameter

POST import/registration form/get-pdf/

Surname	data type	use
meta	objects	meta object
registration forms	array	Array with registration form IDs

request

The following example retrieves the data from two registration forms.

requests:

JSON

```
{
    "meta": {
        "object": 2049,
        "community": 1
    },
    "registration forms": [
        553
    ]
}
```


response

JSON


```
{
        "meta": {
                "requestId": "66f81f62-0942-11eb-bd68-c04a00212a69",
                "timestamp": "2020-10-08T10:43:56.496102",
                "user": {
                        "id": 38,
                        "username": "DemoV",
                        "alias": "Lisa Mustermann"
                },
                "community": 1
        },
        "response": [
                {
                        "id": 553,
                        "encoding": "base64",
                        "pdf": "JVBERi0xLjMKMSAwIG9iaHMgWyAzIDAg....<truncated>"
                }
        ]
}
```


Decode and save PDF

The return from the server contains the PDF data as base64 encoded string values in the response.
The 'value can be found in response[].pdf .

The usual procedure is to save the base64 encoded value as a PDF file and then pass this on for further use (e.g. printing out a spa card).
Possibilities to save Base64 character strings as binary files are available in many programming languages.

code samples

Python 3

```
import base64


# ....
# The server's variable response can be found here in the variable "r".


for ms in r['response']:


    base64_string = ms.get('pdf')
    filename = './Example-Meldeschein-%s.pdf' % ms.get('id') # File as "Example
Save registration form-<id>.pdf".
    fh = open(filename,'wb') #open/create local file in binary mode for writing.


    decoded = base64.decodebytes(base64_string) # Binary decode base64 string
    fh.write(decoded) # Write decoded data to file
    fh.close() # Close the file
    # The file <filename> can now be printed
```

methods

Get configuration

```
GET users/get_current_user
```

This call asks for all information about the current user and associated communities and accommodation providers.

This makes it possible to retrieve and provide defined tariffs, registration form fields and other information in the third-party software.

parameter

No parameters necessary

structure response

General

Field	data type	Explanation
user.*	objects	Information about the current logged in user (via token)
accommodation provider	array	Landlords associated with the account
accommodation provider[].*	objects	All information about landlord
accommodation[].objects	objects	All objects of the landlord, which are assigned to the account
communities	array	All communities assigned to the account (usually one)
municipalities[].districts	array	All defined districts of the Community

Field	data type	Explanation
communes[].tariff zones	array	All defined tariff zones of the Community
communes[].seasonal times	array	All defined seasons of the Community
municipalities[].meldeschein_fields	array	All defined fields of the Electronic registration form

Registration form fields

The defined guest/registration form fields can be found under the following path:

gemeinde[GEMEINDE_ID].meldeschein_fields, where GEMEINDE_ID corresponds to the respective municipality

Relevant fields:

Field	data type	Explanation
alias	string	plaintext designation
column_name	string	The value of the database column, this Value is required when transferring to / book
data_type	string	Data type of the field
required	boolean	Is the value required when booking a registration form?

a notice

All fields with required = true must be transferred when booking a registration form

Defined tariffs, season times, tariff zones

The defined tariffs, season times and tariff zones follow the same structure

Tariffs: gemeinde[GEMEINDE_ID].tarife, where GEMEINDE_ID corresponds to the respective municipality Season times: gemeinde[GEMEINDE_ID].saisonzeiten, where GEMEINDE_ID corresponds to the respective municipality Tariff zones: gemeinde[GEMEINDE_ID].tarifzonen, where GEMEINDE_ID corresponds to the respective municipality

examples

The following example retrieves all configuration for a demo tenant.

Response:

JSON


```
{
    "user": {
        "id": 14,
        "username": "api_demo",
        "first_name": "API",
        "last_name": "Demo",
        "email": "api_demo@apidemo.de",
        "is_active": true,
        "user_type": "lodging provider",
        "alias": "API Demo",
        "last_login": "2019-10-24T11:35:18.015947+02:00"
    },
    "lodging": [
        {
            "id": 9,
            "objects": [
                {
                    "id": 9,
                    "tariffzone_id": 1,
                    "district_id": 1,
                    "accommodation_id": 9,
                    "tariffzone": {
                        "id": 1,
                        "created": "2019-08-21T15:41:56.239951+02:00",
                        "modified": "2019-08-21T15:41:56.240018+02:00",
                        "alias": "Main Zone",
                        "community": 1
                    },
                    "district": {
                        "id": 1,
                        "created": "2019-08-21T15:42:02.596680+02:00",
                        "modified": "2019-08-21T15:42:02.596757+02:00",
                        "alias": "Fehmarn",
                        "community": 1
                    },
                    "lodging": {
```

```
                "company": "DEMO",
                "firstname": "Max",
                "surname": "Doe",
                "comments": "",
                "customer_nr": null,
                "alias": "DEMO"
            },
            "created": "2019-09-04T09:51:53.201523+02:00",
            "modified": "2019-09-04T09:51:53.201585+02:00",
            "number_of_beds": 1,
            "alias": "API Object",
            "street": "API street",
            "zip code": "86707",
            "place": "Westendorf"
        }
    ],
    "alias": "DEMO",
    "created": "2019-09-04T09:49:52.052348+02:00",
    "modified": "2019-09-04T09:56:22.192552+02:00",
    "is_active": true,
    "company": "DEMO",
    "salutation": "sir",
    "firstname": "DEMO",
    "lastname": "DEMO",
    "street": "DEMO",
    "zip code": "123456",
    "place": "DEMO",
    "phone": "12345",
    "email": "demo@demo.de",
    "allow_logins": true,
    "create_invoices": true,
    "custom_1": null,
    "custom_2": null,
    "custom_3": null,
    "custom_4": null,
    "customer_nr": null,
    "landlord_nr": "API TEST",
    "comments": "",
    "community": 1
  }
],
```

```
"communities": [
    {
        "id": 1,
        "districts": [
            {
                "id": 1,
                "created": "2019-08-21T15:42:02.596680+02:00",
                "modified": "2019-08-21T15:42:02.596757+02:00",
                "alias": "main area",
                "community": 1
            }
        ],
        "tariff zones": [
            {
                "id": 1,
                "created": "2019-08-21T15:41:56.239951+02:00",
                "modified": "2019-08-21T15:41:56.240018+02:00",
                "alias": "Main Zone",
                "community": 1
            }
        ],
        "season times": [
            {
                "id": 1,
                "created": "2019-08-21T15:44:42.087596+02:00",
                "modified": "2019-08-21T15:44:42.087663+02:00",
                "alias": "High Season 2019",
                "valid_from": "2019-05-15",
                "valid_until": "2019-09-14",
                "community": 1
            },
            {
                "id": 2,
                "created": "2019-08-22T10:11:18.114641+02:00",
                "modified": "2019-08-22T10:11:18.114702+02:00",
                "alias": "Offseason 1",
                "valid_from": "2019-01-01",
                "valid_until": "2019-05-14",
                "community": 1
            },
            {
```

```
            "id": 3,
            "created": "2019-08-22T10:12:05.322441+02:00",
            "modified": "2019-08-22T10:12:05.322534+02:00",
            "alias": "Off-Season 2",
            "valid_from": "2019-09-15",
            "valid_until": "2019-12-31",
            "community": 1
        }
    ],
    "Rates": [
        {
            "id": 2,
            "full_alias": "Disability 80% (SB 80%)",
            "created": "2019-08-22T10:16:22.572120+02:00",
            "modified": "2019-08-22T10:16:22.572201+02:00",
            "is_active": true,
            "tariff_code": "SB 80%",
            "alias": "Disability 80%",
            "allow_beherberger": true,
            "print_kurkarte": true,
            "default_selected": false,
            "community": 1
        },
        {
            "id": 3,
            "full_alias": "Disability escort (SBB 80%)",
            "created": "2019-08-22T10:16:58.052509+02:00",
            "modified": "2019-08-22T10:16:58.052578+02:00",
            "is_active": true,
            "tariff_code": "SBB 80%",
            "alias": "Disability Companion",
            "allow_beherberger": true,
            "print_kurkarte": true,
            "default_selected": false,
            "community": 1
        },
        {
            "id": 7,
            "full_alias": "Business Attendees (BA)",
            "created": "2019-08-22T10:21:39.062066+02:00",
            "modified": "2019-08-22T10:21:39.062138+02:00",
```

```
        "is_active": true,
        "tariff_code": "BA",
        "alias": "People present",
        "allow_beherberger": true,
        "print_kurkarte": true,
        "default_selected": false,
        "community": 1
    },
    {
        "id": 1,
        "full_alias": "Adult (EW)",
        "created": "2019-08-21T15:43:26.469087+02:00",
        "modified": "2019-09-05T14:54:45.644304+02:00",
        "is_active": true,
        "tariff_code": "EW",
        "alias": "adult",
        "allow_beherberger": true,
        "print_kurkarte": true,
        "default_selected": true,
        "community": 1
    },
    {
        "id": 5,
        "full_alias": "Groups of 25 or more (GG)",
        "created": "2019-08-22T10:17:47.179287+02:00",
        "modified": "2019-08-22T10:17:47.179348+02:00",
        "is_active": true,
        "tariff_code": "GG",
        "alias": "Groups of 25 or more",
        "allow_beherberger": true,
        "print_kurkarte": true,
        "default_selected": false,
        "community": 1
    },
    {
        "id": 4,
        "full_alias": "Groups under 25 people (GK)",
        "created": "2019-08-22T10:17:30.252887+02:00",
        "modified": "2019-08-22T10:17:30.252952+02:00",
        "is_active": true,
        "tariff_code": "GK",
```

```
            "alias": "Groups under 25 people",
            "allow_beherberger": true,
            "print_kurkarte": true,
            "default_selected": false,
            "community": 1
        },
        {
            "id": 8,
            "full_alias": "no category (kK)",
            "created": "2019-08-22T10:22:02.186252+02:00",
            "modified": "2019-08-22T10:22:02.186552+02:00",
            "is_active": true,
            "tariff_code": "kK",
            "alias": "no category",
            "allow_beherberger": true,
            "print_kurkarte": true,
            "default_selected": false,
            "community": 1
        },
        {
            "id": 6,
            "full_alias": "Children (child)",
            "created": "2019-08-22T10:20:33.649448+02:00",
            "modified": "2019-08-22T10:20:33.649504+02:00",
            "is_active": true,
            "tariff_code": "Child",
            "alias": "children",
            "allow_beherberger": true,
            "print_kurkarte": true,
            "default_selected": false,
            "community": 1
        }
    ],
    "registration form_fields": [
        {
            "id": 2,
            "created": "2019-08-21T15:46:11.393579+02:00",
            "modified": "2019-09-05T14:56:24.456160+02:00",
            "alias": "last name",
            "column_name": "guest_name",
            "data_type": "text",
```

```
        "size": 6,
        "transferable": false,
        "list_visible": true,
        "required": true,
        "priority": "high",
        "view_order": 1,
        "community": 1
    },
    {
        "id": 1,
        "created": "2019-08-21T15:45:57.452169+02:00",
        "modified": "2019-09-05T14:56:20.478747+02:00",
        "alias": "first name",
        "column_name": "guest_firstname",
        "data_type": "text",
        "size": 6,
        "transferable": false,
        "list_visible": true,
        "required": false,
        "priority": "high",
        "view_order": 2,
        "community": 1
    },
    {
        "id": 3,
        "created": "2019-08-22T10:26:05.736723+02:00",
        "modified": "2019-08-22T10:26:05.736825+02:00",
        "alias": "Street",
        "column_name": "Guest_Street",
        "data_type": "text",
        "size": 12,
        "transferable": false,
        "list_visible": false,
        "required": false,
        "priority": "low",
        "view_order": 3,
        "community": 1
    },
    {
        "id": 4,
        "created": "2019-08-22T10:26:29.499058+02:00",
```

```
        "modified": "2019-08-22T10:26:29.499139+02:00",
        "alias": "zip code",
        "column_name": "guest_zip",
        "data_type": "text",
        "size": 4,
        "transferable": false,
        "list_visible": false,
        "required": false,
        "priority": "low",
        "view_order": 4,
        "community": 1
    },
    {
        "id": 5,
        "created": "2019-08-22T10:26:42.577394+02:00",
        "modified": "2019-08-22T10:27:14.479290+02:00",
        "alias": "place",
        "column_name": "guest_location",
        "data_type": "text",
        "size": 8,
        "transferable": false,
        "list_visible": true,
        "required": false,
        "priority": "low",
        "view_order": 5,
        "community": 1
    },
    {
        "id": 6,
        "created": "2019-08-26T13:50:23.435905+02:00",
        "modified": "2019-09-05T15:02:12.258999+02:00",
        "alias": "date of birth",
        "column_name": "guest_birthday",
        "data_type": "date",
        "size": 4,
        "transferable": false,
        "list_visible": false,
        "required": false,
        "priority": "low",
        "view_order": 6,
        "community": 1
```

```
                }
            ],
            "created": "2019-08-21T15:00:59.512049+02:00",
            "modified": "2019-08-21T15:36:50.593090+02:00",
            "alias": "Fehmarn",
            "using_paper": true,
            "using_ems": true,
            "datatable_name": "dat_99999"
        }
    ],
    "created": "2019-09-04T09:52:03.825645+02:00",
    "modified": "2019-09-04T09:52:03.825762+02:00",
    "permissions": [
        "authentication.add_landlordaccount",
        "authentication.change_landlordaccount",
        "authentication.delete_landlordaccount",
        "authentication.view_landlordaccount",
        "common.view_beherberger",
        "common.view_object"
    ]
}
```

Cancel registration forms

This call sets the status of the transferred registration forms to "cancelled".

Canceled registration forms can no longer be changed.

POST import/registration form/cancel/

parameter

Surname	data type	use
meta	object	meta object
registration forms	array	Array with registration form IDs

examples

The following example cancels two registration forms

request

```
{
  "meta": {
    "community": 1,
    "object": 2049
  },
  "registration forms": [
    347,348
  ]
}
```

response

JSON


```
{
    "meta": {
        "community": 1,
        "object": 2049,
        "requestId": "5546961c-8d5a-11eb-b450-c04a00212a69",
        "timestamp": "2021-03-25T12:07:48.219553",
        "user": {
            "id": 2,
            "username": "admin",
            "alias": "Administrator"
        }
    },
    "status": "success",
    "registration forms": [
        347,348
    ]
}
```