DEC112 - Deaf Emergency Call 112
Registration API

DEC112

Deaf Emergency Call - 112

Version 0.0.3
Richard Prinz
April 2020

PRELIMINARY VERSION – SUBJECT TO CHANGE
INTERNAL USE ONLY

This page was intentionally left blank.

Table of Contents

Table of Figures

Abbreviations

BGW	Border Gateway
DEC112	Deaf Emergency Call
JSON	JavaScript Object Notation
PSAP	Public Safety Answering Point
REST	Representational State Transfer
WS	WebSocket
REGAPI	Registration API
DID	Decentralized Identifier

Terms

Call	This document uses the term "Call" for text based (text chat) communication between peers. There is no voice connection in the conventional sense.
Client	A DEC112 client software (e.g. mobile application)
Device	A REGAPI v1 Client running on a device (e.g. mobile phone)
Registration	A REGAPI v2 association between a Client and REGAPI v2

This page was intentionally left blank.

1 DEC112 Registration API description

The DEC112 Registration API (REGAPI) is responsible for managing DEC112 registrations and to provide configuration information for DEC112 clients (Client). It uses REST[1] and JSON[2] technologies.

This document describes version 1 and version 2 of the API.

Version 1 of REGAPI is the legacy version originally developed for the DEC112 project. It uses the concept of a device (e.g. cell phone) to be registered as a DEC112 client (e.g. through a mobile APP). It also provides multiple methods to verify if a registration is valid like SMS, email or manual verification. In addition, personal data like a person name, address, and contact information like email or phone number are stored.

The newer version 2 of the REGAPI provides only the absolute necessary functions to handle a more abstract form of registration and does not store any personal information. Only a phone number verification is available.

1.1 Common data structures and behaviour

Some data formats and structures are common for multiple API methods and are described below.

1.1.1 Timestamps

All timestamps used in API messages are in the form `yyyy-mm-ddTHH:MM:SS.ssZ` where `y`=year, `m`=month, `d`=day, `M`=minutes, `S`=seconds, `s`=milliseconds. Time is always UTC and should be converted to local time by clients for display.

1.1.2 Phone numbers

Phone numbers must be provided and are returned to/from the REGAPI in the international format: `[00]` **`[country code]`** **`[area code]`** **`[phone number]`** without any whitespaces or other formatting characters like "-" or parenthesis e.g. **00430**11234567 for a landline number in Vienna/Austria.

1.1.3 Locations

All locations are in WGS84 degrees provided as follows:

```
{
  "lat": 48.227007156238,
  "lon": 16.3463576417416,
  "alt": 220,
  "rad": null,
  "method": "gps"
}
```

[1] https://en.wikipedia.org/wiki/Representational_state_transfer
[2] https://tools.ietf.org/html/rfc7159.html

The `alt` attribute is the altitude in meters, `rad` a radius around a latitude longitude point in meters (which then represents a circle) and `method` the method how the position was determined (e.g. gps, manual, wifi etc.).

The `lat` and `lon` attributes are always provided with valid values (otherwise no location at all would be provided). All other attributes values are 'null' if not available.

1.1.4 Language and localization

To provide localized content to Client and responses from REGAPI, some API methods support a `lang` attribute containing an ISO639-1 two letter language code. If omitted the first available language configured on server side will be used which, in a default installation is "en" for English.

1.1.5 Responses

A response to a REGAPI call is always a valid JSON object – even if it is empty.

The optional `runtime_ms` attribute might be present in a response. It is the runtime of the method call to complete in milliseconds and is only available if server runs in debug mode.

Successful API responses are indicated by a http status code in the 2xx range. REGAPI v1 in addition also includes the http result code in the JSON response as separate `code` attribute.

1.1.6 Registration State

A Registration always has one of the following states:

0	no Registration available[3]
1	registered but one or more verifications (e.g. SMS, email missing)
2-8	ongoing verification process
9	Registration is in error state[4]
10	Registration fully verified, ready to be used

Some methods return the following state object:

1.1.6.1 REGAPI v1 Registration State

```
{
  "code": 200,
  "device_id": "e5ddb1283009d6ddbcf5",
  "state": 2,
  "lang": "en",
```

[3] REGAPI v1 returns in some cases registration state 0 for unknown or invalid registrations, REGAPI v2 consequently returns a non-successful http status code.

[4] Reserved for future use.

```
    "owner_verified": null,
    "phone_verified": "2018-04-27T10:25:01.042Z",
    "email_verified": null,
    "device_registered": "2018-04-27T10:13:02.453Z",
    "runtime_ms": "0.49305"
  }
```

Attribute	Description
`code`	API result code
`device_id`	Unique ID of device
`state`	The current state of a devices' registration
`lang`	An ISO639-1 two letter language code
`owner_verified`	UTC timestamp when owner was successfully verified
`phone_verified`	UTC timestamp when phone number was successfully verified
`email_verified`	UTC timestamp when email address was successfully verified
`device_registered`	UTC timestamp when device was registered
`runtime_ms`	Optional number of milliseconds needed to complete action

1.1.6.2 REGAPI v2 Registration State

```
  {
    "reg_id": "5b80637f-df3b-fe13-de27-6791a5178028",
    "lang": "de",
    "state": 1,
    "phone_verified_ts": null,
    "phone_privacy": true,
    "registered_ts": "2018-04-27T10:13:02.453Z",
    "runtime_ms": "0.49305"
  }
```

Attribute	Description
`reg_id`	Unique ID of Registration
`lang`	An ISO639-1 two letter language code
`state`	The current state of the Registration
`phone_verified_ts`	UTC timestamp when phone number was successfully verified
`phone_privacy`	Indicates if server is configured to delete a registrations phone number after successful verification
`registered_ts`	UTC timestamp when Registration was created
`runtime_ms`	Optional number of milliseconds needed to complete action

1.1.7 Errors

Errors are indicated using http status codes in the 4xx and 5xx range. In addition, a JSON response with a human readable reason is returned.

If server runs in debug mode `message` contains the full runtime error message (including source location and possible stack traces). Otherwise only basic information is provided in the `message` attribute.

1.1.7.1 REGAPI v1 Errors

REGAPI v1 also includes the http response code in the JSON response:

```
{
   "code": 500,
   "message": "error message"
}
```

1.1.7.2 REGAPI v2 Errors

In case of an error the response is:

```
{
   "message": "error message"
}
```

1.2 REST API Version 1

This version of the REST API is reachable via http://host:port/api/v1.

For testing, development, and automatic client code generation, the API provides SWAGGER[5] metadata (including the interactive SWAGGER UI) accessible via http://host:port/api/v1/api-docs.(SWAGGER metadata) and http://host:port/api/ (SWAGGER UI).

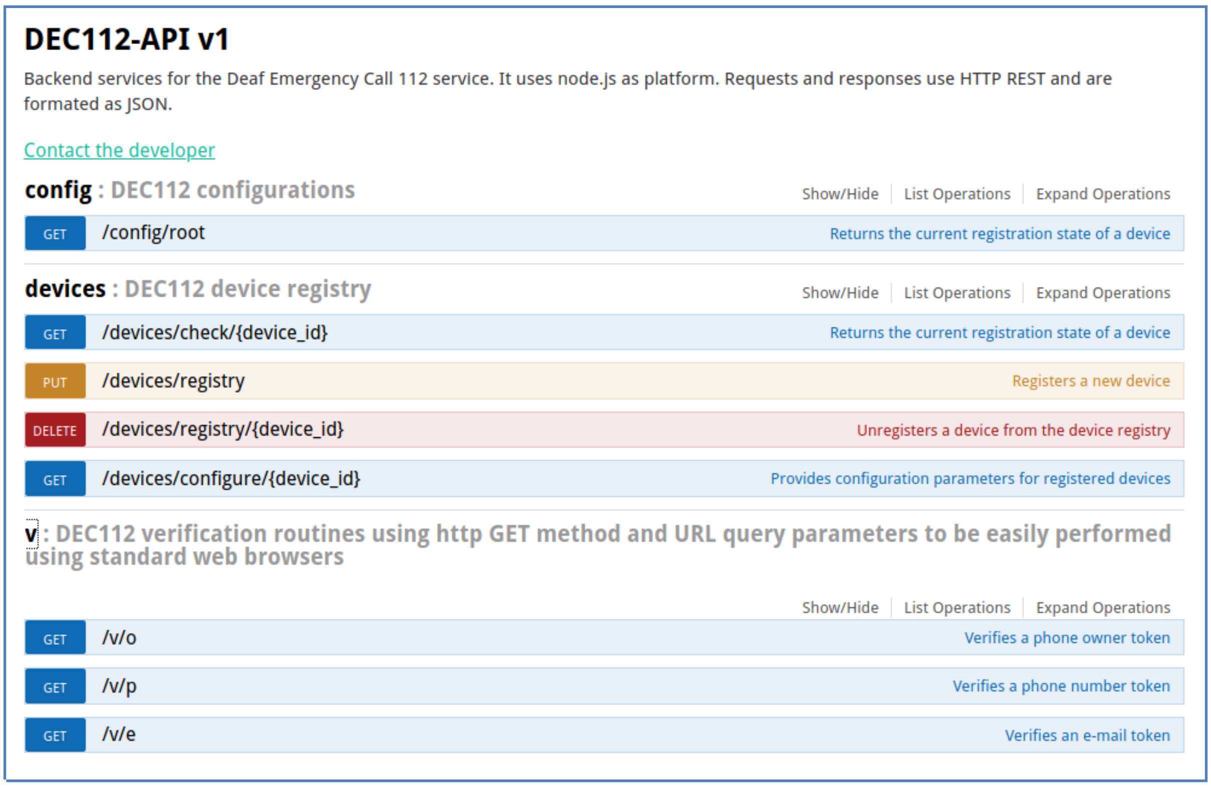
To use the REST API a client needs an API key. This key can be provided either as http header `api_key` or as URL parameter `api_key=`.

http header:

```
api_key: my_secret_api_key
```

URL parameter:

http://host:port/api/v1/path/to/method?api_key=my_secret_api_key



DEC112-API v1

Backend services for the Deaf Emergency Call 112 service. It uses node.js as platform. Requests and responses use HTTP REST and are formated as JSON.

Contact the developer

config : DEC112 configurations — Show/Hide | List Operations | Expand Operations

GET	/config/root	Returns the current registration state of a device

devices : DEC112 device registry — Show/Hide | List Operations | Expand Operations

GET	/devices/check/{device_id}	Returns the current registration state of a device
PUT	/devices/registry	Registers a new device
DELETE	/devices/registry/{device_id}	Unregisters a device from the device registry
GET	/devices/configure/{device_id}	Provides configuration parameters for registered devices

v : DEC112 verification routines using http GET method and URL query parameters to be easily performed using standard web browsers — Show/Hide | List Operations | Expand Operations

GET	/v/o	Verifies a phone owner token
GET	/v/p	Verifies a phone number token
GET	/v/e	Verifies an e-mail token

Figure 1: SWAGGER UI Web interface for API v1

[5] https://swagger.io/

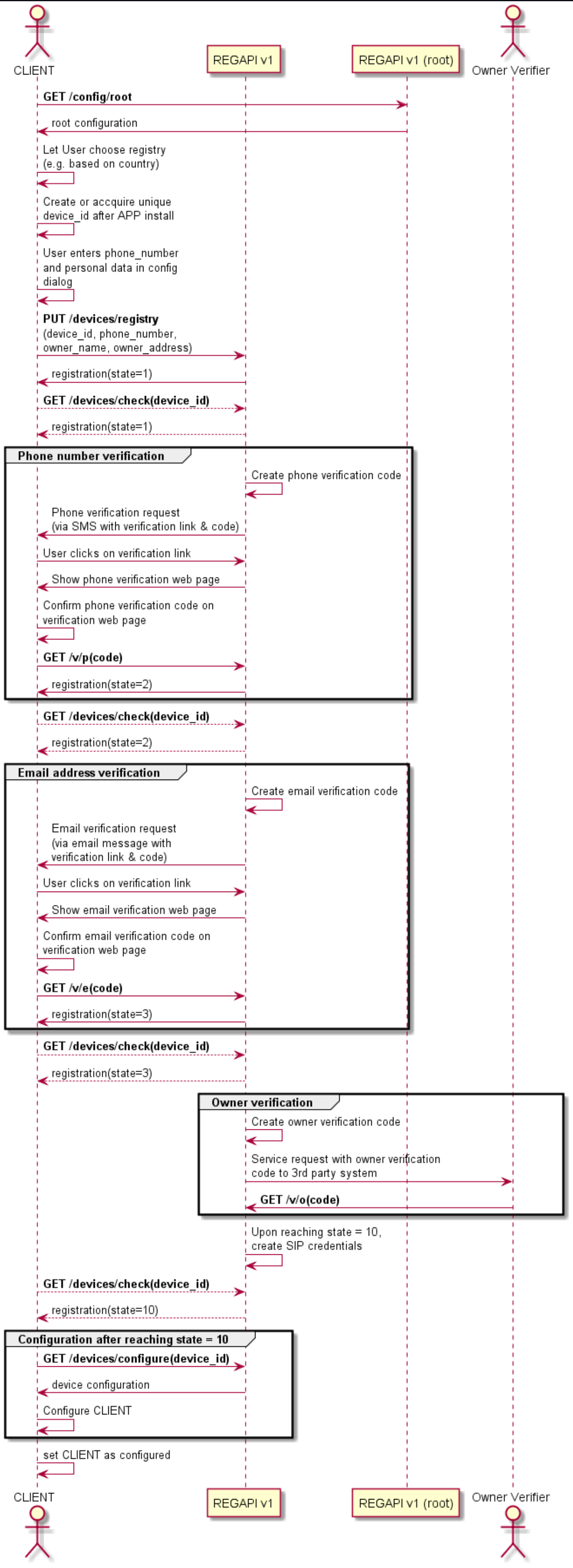

CLIENT
REGAPI v1
REGAPI v1 (root)
Owner Verifier
GET /config/root
root configuration
Let User choose registry (e.g. based on country)
Create or accquire unique device_id after APP install
User enters phone_number and personal data in config dialog
PUT /devices/registry (device_id, phone_number, owner_name, owner_address)
registration(state=1)
GET /devices/check(device_id)
registration(state=1)
Phone number verification
Create phone verification code
Phone verification request (via SMS with verification link & code)
User clicks on verification link
Show phone verification web page
Confirm phone verification code on verification web page
GET /v/p(code)
registration(state=2)
GET /devices/check(device_id)
registration(state=2)
Email address verification
Create email verification code
Email verification request (via email message with verification link & code)
User clicks on verification link
Show email verification web page
Confirm email verification code on verification web page
GET /v/e(code)
registration(state=3)
GET /devices/check(device_id)
registration(state=3)
Owner verification
Create owner verification code
Service request with owner verification code to 3rd party system
GET /v/o(code)
Upon reaching state = 10, create SIP credentials
GET /devices/check(device_id)
registration(state=10)
Configuration after reaching state = 10
GET /devices/configure(device_id)
device configuration
Configure CLIENT
set CLIENT as configured
CLIENT
REGAPI v1
REGAPI v1 (root)
Owner Verifier

Figure 2: REGAPI v1 initial registration procedure

1.2.1 Verification

This API provides three options to check a device registration for validity: phone number, email address and owner verification.

1.2.1.1.1 Phone number verification

Phone number verification, when enabled server side, verifies the phone number of the registrant by sending a SMS (shown in Figure 3) which includes a web link. This link needs to be opened in a standard Web-Browser either through a click or by copying the URL.

***Note**: In case there is no Web-Browser available on receiving device, this verification method cannot be used!*

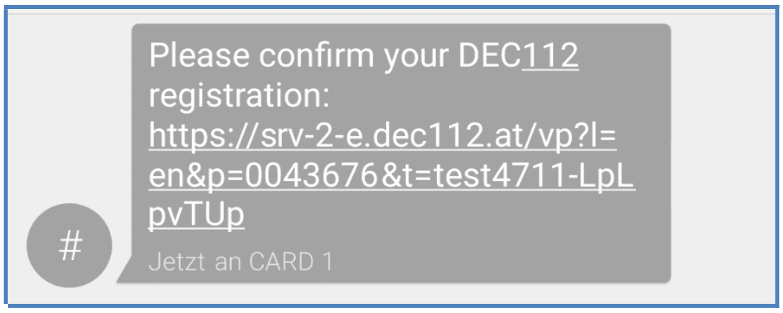
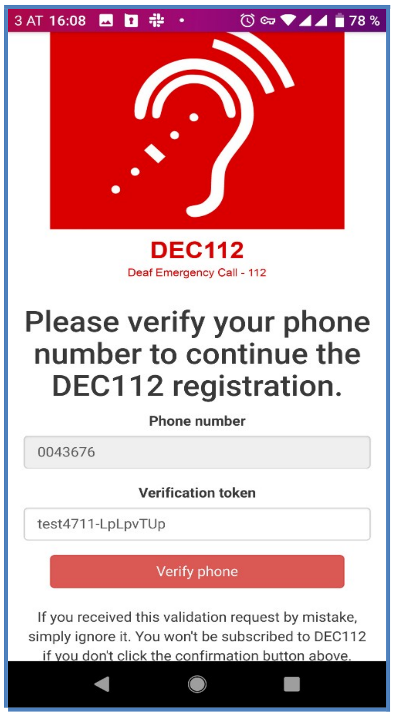

Please confirm your DEC112 registration: https://srv-2-e.dec112.at/vp?l=en&p=0043676&t=test4711-LpLpvTUp

Figure 3: REGAPI v1 phone number verification SMS

When opening the link in a Web-Browser, phone number can be checked for correctness and finally approved.



3 AT 16:08 78 %

DEC112

Deaf Emergency Call - 112

Please verify your phone number to continue the DEC112 registration.

Phone number

0043676

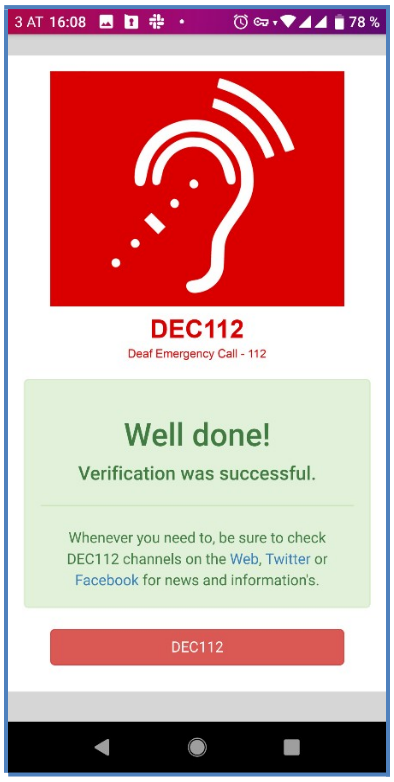
Verification token

test4711-LpLpvTUp

Verify phone

If you received this validation request by mistake, simply ignore it. You won't be subscribed to DEC112 if you don't click the confirmation button above.

Figure 4: REGAPI v2 phone number verification web page



3 AT 16:08 78 %

DEC112

Deaf Emergency Call - 112

Well done!

Verification was successful.

Whenever you need to, be sure to check DEC112 channels on the Web, Twitter or Facebook for news and information's.

DEC112

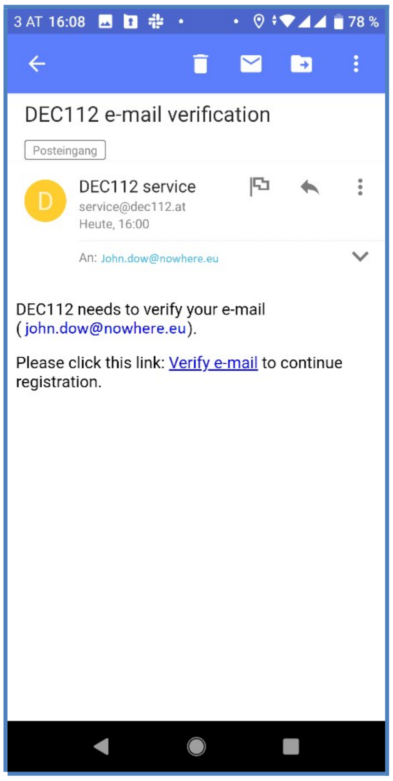
Figure 5: REGAPI v2 phone number successfully verified web page

1.2.1.1.2 Email verification

Email verification, when enabled server side, verifies the email address of the registrant by sending an email to the email address provided during initial registration.

The email message contains a link that must be opened in a standard Web-Browser same as described in 1.2.1.1.1 for phone number verification.

***Note**: In case there is no Web-Browser available on receiving device, this verification method cannot be used!*



DEC112 e-mail verification

DEC112 service
service@dec112.at
Heute, 16:00

An: John.dow@nowhere.eu

DEC112 needs to verify your e-mail
(john.dow@nowhere.eu).

Please click this link: Verify e-mail to continue registration.

Figure 6: REGAPI v1 email address verification message

By clicking the "Verify e-mail" link, similar verification and status web pages as for SMS verification shown in Figure 4 and Figure 5 are displayed.

1.2.1.1.3 Owner verification

Owner verification is a special type of verification. It can be used to hand over verification to other 3rd party systems by implementing a server-side http request. This request will pass on personal data received by REGAPI during initial registration which can then be validated based on any criteria.

1.2.2 API methods

Version 1 of the REGAPI provides methods in three different areas: configuration functions, device registry and verification functions.

1.2.2.1 Configuration functions

These functions provide configuration data for a Client.

1.2.2.1.1 get_root_config

The root configuration provides a Client with a list of DEC112 registries and other resources (e.g. language resources). A user can select the local registry he wants to use during initial registration via the Client GUI.

REST Request

http method: GET

http://host:port/api/v1/config/root?api_key={api_key}

Response example:

```
{
  "root": {
    "country_registries": [
      "name": "AT",
      "registration_api: [
        "type": "dec112_v1",
        "url": "https://service.dec112.at"
      }
    ]
  },
  "runtime_ms": "0.49305"
}
```

1.2.2.1.2 get_config

REST Request

http method: GET

http://host:port/api/v1/configure/{reg_id}?api_key={api_key}

Response example:

```
{
  "code": 200,
  "device_id": "5b80637f-df3b-fe13-de27-6791a5178028",
  "server": "ws://service.dec112.at",
  "public_id": "sip:6da87ddcfa10ac65ea00437f@service,dec112.at",
  "private_id: "6da87ddcfa10ac65ea00437f",
  "password": "4460af63bd663ffc109",
  "realm": "service.dec112.at",
  "services": [
    {
      "urn": "urn:service.police",
      "enabled": true,
    }
  ],
  "runtime_ms": "0.49305"
16:   }
```

1.2.2.2 Device registry

Version 1 of the API uses the concept of devices to identify a DEC112 user. A device is the user's device which has a Client installed which interacts with the REGAPI. It is uniquely identified by a `device_id` provided by the Client (either directly provided by the platform it is running on or a unique identifier generated by the Client itself if the mobile platform does not provide such an ID).

1.2.2.2.1 register

REST Request

http method: PUT

http://host:port/api/v1/devices/registry/{reg_id}?api_key={api_key}

Response example:

Returns a state object as described in 1.1.6.1.

1.2.2.2.2 unregister

REST Request

http method: DELETE

http://host:port/api/v1/devices/register/{reg_id}?api_key={api_key}

Response example:

Returns a state object as described in 1.1.6.1.

1.2.2.2.3 check

REST Request

http method: GET

http://host:port/api/v1/devices/check/{reg_id}?api_key={api_key}

Response example:

Returns a state object as described in 1.1.6.1.

1.2.2.3 Verification functions

A DEC112 registration can be verified by means of SMS, email or manual intervention. This is to ensure to some extent that registrations are valid. Which (if any) verifications are required is configured on server side.

1.2.2.3.1 verify_phone

Verifies a devices phone number. This method is only useful if REGAPI server is configured with enabled phone number verification. The `code` URL parameter is the code sent to Client via SMS during initial registration.

REST Request

http method: GET

http://host:port/api/v1/v/p?d={device_id}&p={code}&api_key={api_key}

Response example:

Returns a state object as described in 1.1.6.1.

1.2.2.3.2 verify_email

Verifies a devices email address. This method is only useful if REGAPI server is configured with enabled email address verification. The `code` URL parameter is the code sent to Client via email message during initial registration.

REST Request

http method: GET

http://host:port/api/v1/v/e?d={device_id}&e={code}&api_key={api_key}

Response example:

Returns a state object as described in 1.1.6.1.

1.2.2.3.3 verify_owner

Verifies a devices owner. This method is only useful if REGAPI server is configured with enabled owner verification. The `code` URL parameter is the code sent to owner verification system during initial registration.

REST Request

http method: GET

http://host:port/api/v1/v/o?d={device_id}&e={code}&api_key={api_key}

Response example:

Returns a state object as described in 1.1.6.1.

1.3 REST API Version 2

This version of the REST API is reachable via http://host:port/api/v2.

For testing, development, and automatic client generation, the API provides SWAGGER[6] metadata (including the interactive SWAGGER UI) accessible via http://host:port/api/v2/api-docs.(SWAGGER metadata) and http://host:port/api/ (SWAGGER UI).

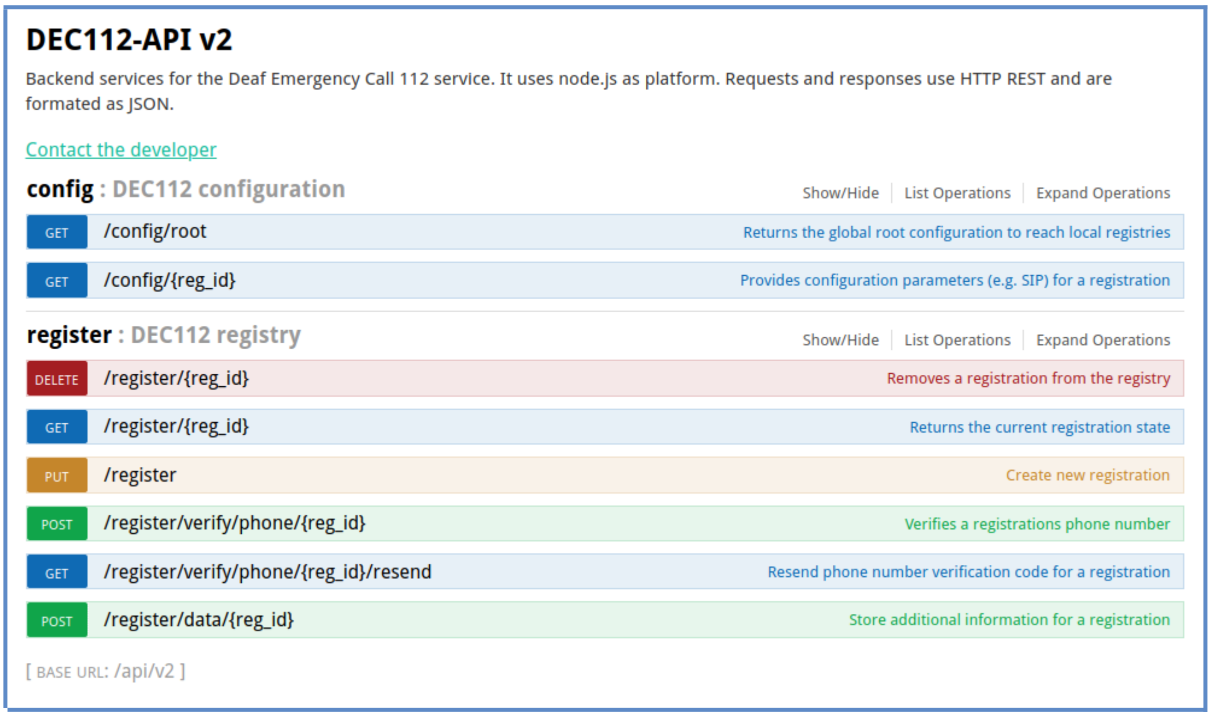
To use the REST API a client needs an API key. This key can be provided either as http header `api_key` or as URL parameter `api_key=`.

http header:

```
api_key: my_secret_api_key
```

URL parameter:

http://host:port/api/v2/path/to/method?api_key=my_secret_api_key



DEC112-API v2

Backend services for the Deaf Emergency Call 112 service. It uses node.js as platform. Requests and responses use HTTP REST and are formated as JSON.

Contact the developer

config : DEC112 configuration — Show/Hide | List Operations | Expand Operations

GET	/config/root	Returns the global root configuration to reach local registries
GET	/config/{reg_id}	Provides configuration parameters (e.g. SIP) for a registration

register : DEC112 registry — Show/Hide | List Operations | Expand Operations

DELETE	/register/{reg_id}	Removes a registration from the registry
GET	/register/{reg_id}	Returns the current registration state
PUT	/register	Create new registration
POST	/register/verify/phone/{reg_id}	Verifies a registrations phone number
GET	/register/verify/phone/{reg_id}/resend	Resend phone number verification code for a registration
POST	/register/data/{reg_id}	Store additional information for a registration

[BASE URL: /api/v2]

Figure 7: SWAGGER UI Web interface for API v2

[6] https://swagger.io

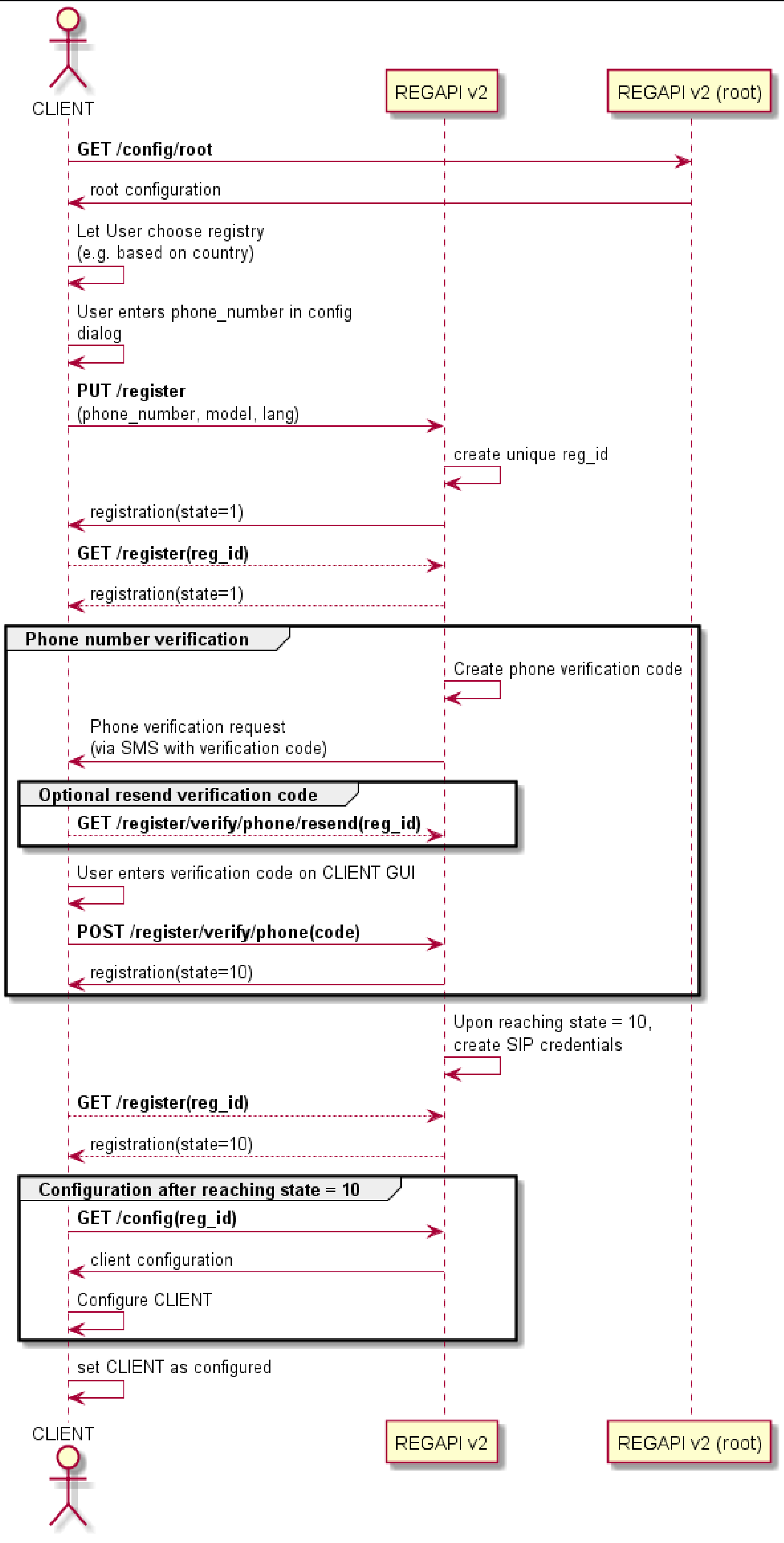
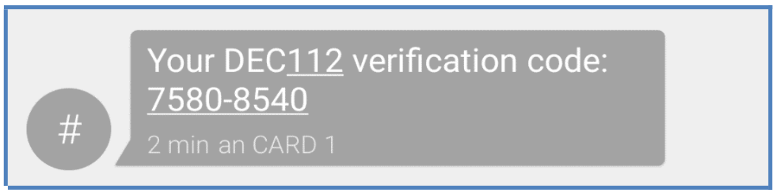


Figure 8: REGAPI v2 initial registration procedure

1.3.1 Verification

This API only supports (an optional) phone number verification. If enabled server side, a phone number is required to register which needs to be verified through a code sent via SMS to Client. After initial registration the Registration state stays at 1 (registered but unverified) until the phone number is successfully verified by using the code sent via SMS to Client with the **verify_phone** (1.3.2.2.4) method. This sets the state to 10 (fully verified).



Your DEC112 verification code: 7580-8540

2 min an CARD 1

Figure 9: REGAPI v2 phone number verification SMS

When server is configured not to use phone verification, state is 10 (fully verified) immediately after initial registration.

***Note**: In contrast to REGAPI v1 phone verification process there is no need for a preinstalled Web-Browser on registrants' device. Instead Client must provide functions which enables to enter received code. Client then must call REGAPI v2 method* **verify_phone** *(1.3.2.2.4) with entered code.*

For privacy reasons REGAPI v2 can be configured to delete phone number after successful verification process (or to not require a phone number at all when phone number verification is disabled).

1.3.2 API methods

Version 2 of the REGAPI provides methods in two different areas: configuration and registry functions.

1.3.2.1 Configuration functions

These functions provide configuration data for a Client.

1.3.2.1.1 get_root_config

The root configuration provides a Client with a list of DEC112 registries and other resources (e.g. language resources). A user can select the local registry he wants to use during initial registration via the Client GUI.

REST Request

```
http method: GET
http://host:port/api/v2/config/root?api_key={api_key}
```

Response example:

```
{
  "root": {
    "country_registries": [
      "name": "AT",
      "registration_api: [
        "type": "dec112_v1",
        "url": "https://service.dec112.at"
      }
    ]
  },
  "runtime_ms": "0.49305"
}
```

1.3.2.1.2 get_config

Returns the Client configuration for a given registry ID. Configuration contains SIP credentials needed to connect and authenticate on SIP server. Credentials are automatically generated on server during initial registration when (optional) phone number verification successfully completes and contains no personal data.

Also included in the response is a list of available services a user can call. A Client must configure its GUI elements (call buttons / icons) according to the provided service list.

REST Request

http method: GET

http://host:port/api/v2/config/{reg_id}?api_key={api_key}

Response example:

```
{
  "reg_id": "5b80637f-df3b-fe13-de27-6791a5178028",
  "server": "ws://service.dec112.at",
  "public_id": "sip:6da87ddcfa10ac65ea00437f@service,dec112.at",
  "private_id: "6da87ddcfa10ac65ea00437f",
  "password": "4460af63bd663ffc109",
  "realm": "service.dec112.at",
  "services": [
    {
      "urn": "urn:service.police",
      "enabled": true,
    }
  ],
  "runtime_ms": "0.49305"
}
```

1.3.2.2 Registry

Version 2 of the API uses the concept of an arbitrary “Registration” to identify a DEC112 user. This API does not store any personal data. A phone number verification can optionally be enabled server side. In this case a phone number is needed and is only stored until successfully verified after which it is deleted. A Registration is uniquely identified by a `reg_id` generated server side during initial registration.

1.3.2.2.1 register

Creates a new Registration.

The optional `model` attribute should contain information about the device a user is using. Its purpose is to help identifying and solving possible Client problems on different platforms. If not available or not provided, “unknown” will be used.

To provide localized content in Client and responses from REGAPI an optional `lang` attribute containing an ISO639-1 two letter language code can be specified. If omitted the first available language configured on server side will be used which, in a default installation is “en” for English.

An optional phone number required to receive the SMS verification code.

***Note**: If server is configured to verify phone numbers, the `phone_number` attribute is required. If server is not configured to verify phone numbers the provided phone number is stored unverified.*

In addition, server can be configured to delete phone numbers after successful verification for privacy reasons. In case phone numbers are not verified and should be deleted after veification then phone number in request will simply be ignored.

REST Request

http method: PUT

http://host:port/api/v2/register?api_key={api_key}

```
{
   "model": "Fairphone 3",
   "lang": "de",
   "phone_number": "004312345678"
}
```

Response example:

Returns a state object as described in 1.1.6.2.

1.3.2.2.2 unregister

Deletes an existing Registration. Any Client must call this method upon forced deregistration initiated through user by means of Client GUI or when Client software is uninstalled.

If a DID was created for the Registration then the DID would also automatically be revoked using DID providers API.

***Note**: When server is configured to store a registrations phone number after successful verification then a phone number in request would be ignored and can be omitted. It is only required when server is configured to delete phone number after verification.*

***Note**: The DEC112 Registration is deleted even in case DID providers revoke API call will fail!*

REST Request

http method: DELETE

http://host:port/api/v2/register/{reg_id}&phone_number={phone_number}?api_key={api_key}

Response example:

```
{
  "reg_id": "5b80637f-df3b-fe13-de27-6791a5178028",
  "runtime_ms": "0.49305"
}
```

1.3.2.2.3 check

Get state of an existing Registration. This method allows a Client to determine the current state of a Registration. The Client is only allowed to enable full DEC112 functionality at Registration state 10. See also 1.1.6.

REST Request

http method: GET

http://host:port/api/v2/register/{reg_id}?api_key={api_key}

Response example:

Returns a state object as described in 1.1.6.2.

1.3.2.2.4 verify_phone

Verifies a registrations phone number. This method is only useful if REGAPI server is configured with enabled phone number verification.

REST Request

http method: POST

http://host:port/api/v2/register/verify/phone/{reg_id}?api_key={api_key}

```
{
  "code": "4711-0815"
}
```

Response example:

Returns a state object as described in 1.1.6.2.

1.3.2.2.5 resend_phone_verification

Allows resending a possibly lost phone verification SMS code up to two times.

REST Request

http method: POST

http://host:port/api/v2/register/verify/phone/{reg_id}/resend?api_key={api_key}

Response example:

```
{
  "reg_id": "5b80637f-df3b-fe13-de27-6791a5178028",
  "runtime_ms": "0.49305"
}
```

1.3.2.2.6 store_data

Store additional, arbitrary data for a Registration. REGAPI v1 allows to store predefined personal data like a person name, phone number or address. The new REGAPI v2 offers two improvements. First, any type of data could be stored and second, data is no longer stored at REGAPI server. For increased privacy, REGAPI v2 uses Decentralized Identifier (DID) providers[7] where such data is stored.

This method can only be used when Registration state equals 10 (fully verified) otherwise an error will be returned. It will create a new DID and links the provided data with it. The DID

***Note**: When server is configured to delete phone number after successful verification the `phone_number` attribute in request is required. When server is configured to store a registrations phone number after successful registration, the `phone_number` attribute in request is ignored (can be omitted) and the registrations stored phone number is used instead. See also 1.3.1.*

***Note**: This is a convenience function in REGAPI v2. Additional functions to manage private data stored using a DID provider must be performed (either implemented on Client or manually by user using DID providers web application) using the API of the DID provider.*

***Note**: This method might only be called once die to DID provider restrictions. This means it will return an error even when called with valid parameters.*

REST Request

http method: POST

http://host:port/api/v2/register/data/{reg_id}?api_key={api_key}

```
{
  "phone_number": "00431234567890",
  "data": {}
}
```

Response example:

```
{
  "reg_id": "5b80637f-df3b-fe13-de27-6791a5178028",
```

[7] https://www.ownyourdata.eu

```
    "did": "did:example:1234567890",
    "runtime_ms": "0.49305"
  }
```

--- End of document ---