DIVUS VISION API

DIVUS VISION API – User Manual

Version 1.00

REV01-20240528

GENERAL INFORMATION

DIVUS GmbH
Pillhof 51
I-39057
Eppan (BZ) - Italy

The manual is subject to change without notice. We cannot guarantee that the data contained in this document and on the storage media supplied are free of errors and correct. Suggestions for improvements as well as hints on errors are always welcome. The agreements also apply to the specific annexes to this manual.

The designations in this document may be trademarks whose use by third parties for their own purposes may infringe the rights of their owners.

User instructions: Please read this manual before using it for the first time and keep it in a safe place for future reference.

Target group: The manual is written for users with previous knowledge of PC and automation technology.

PRESENTATION CONVENTIONS

[KEY]	Keystrokes of the user are shown in square brackets, e.g. [CTRL] or [DEL].
COURIER	Screen output is described in the Courier font, e.g. `C:\>`
COURIER FAT	Keyboard input by the user is described in Courier font bold, e.g. **`C:\> DIR`**
"„…“	Names of buttons, menus or other screen elements to be selected are displayed in "inverted commas".
PICTOGRAMS	The following pictograms are used in the manual to identify certain sections of text:
	*Watch your step!* Possibly dangerous situation. Damage to property can be the result.
	*Notes Tips* and supplementary information
	*New* Marks changes and new features

TABLE OF CONTENTS

1 Introduction

1.1 GENERAL INTRODUCTION

This manual describes the VISION API (Application Programming Interface) - an interface through which VISION can be addressed and controlled from external systems.

In practical terms, this means that you can use systems such as

MQTT Explorer (https://www.microsoft.com/store/... - for testing),
Home Assistant (https://www.home-assistant.io/) or
Node-RED (https://nodered.org/)

to control the elements managed by VISION or read out their status.

Access and communication take place via the MQTT protocol, which uses so-called topics to address individual functions or sets of functions or to be informed about changes to them. An MQTT server (broker) is used for this purpose, which handles security and the management/distribution of messages to the participants. In this case, the MQTT server is located directly on the DIVUS KNX IQ and is specially configured for this purpose.

Although the VISION API can also be used without programming knowledge, this functionality is suitable for advanced users.

1.2 PREREQUISITES

As explained in the VISION manual, the API user must by default first be activated in order to be able to use it – the API access only works using the Api user's authentication data. As far as the user rights are concerned, the activation for this functionality can then be configured either on all or on individual elements. See Chap.0.

Of course, you also need a VISION project in which the elements that you want to control from outside are fully configured and the connection to them has been successfully tested. To be able to address individual elements via the API, their element ID must be known: this is displayed at the bottom of the element's settings form. See Chap. 1.4.

1.3 SECURITY

For security reasons, API access is only possible locally (i.e. not via the cloud). The security risk when activating API access is therefore low. Nevertheless, security-relevant elements should not be enabled or explicitly denied for API access.

1.4 MQTT AND ITS TERMS - BRIEF EXPLANATION

The main difference between MQTT and the classic client-server model, where the client requests and then changes data, is centred on the concepts of ***subscribe*** and ***publish***. Participants may publish data, making it available to others, who – if interested – may subscribe to it. This architecture makes it possible to minimise data exchange and still keep all interested parties up to date. More about the details here:

BROKER

In MQTT, the role of centralised management and distribution of all messages is that of the ***broker***. Although MQTT server and MQTT broker are not synonyms (server is a broader term for a role that MQTT clients can also play), the broker is always meant in this manual when MQTT server is mentioned. The DIVUS KNX IQ itself plays the MQTT broker / MQTT server role in the context of this manual.

TOPICS

An MQTT server uses so-called ***topics***: a hierarchical structure with which data is categorised, managed and published.

PUBLISH

Publishing has the primary goal of making data available to other participants through topics. If you want to change a value, you write to the desired topic together with the desired value change, also using a publishing action. The target device or the MQTT server reads the desired change that affects it and adopts it accordingly. To check that the change has been applied, you can look in the subscribed real-time topic to see whether the change is reflected there - if everything has worked out fine.

SUBSCRIBE

Clients select the topics that interest them: this is called *subscribing*. Every time a value changes in/below a topic, all subscribed clients are informed - i.e. without having to explicitly ask whether something has changed or what the current value is.

CLIENT_ID

You can open (or address) a separate communication channel with the MQTT server by entering any unique string called ***client_id*** in a topic. The client_id **must be used** in the topic to process values. This serves to identify the origin of each change, helps with any errors and does not affect the other clients, as the corresponding responses from the server, including any error codes and messages, also only reach the topic with the same client_id (and thus only that client).

The client_id is a unique character string consisting of any combination of the characters 0-9, a-z, A-Z, "-", "_".

STATUS / REQUEST

In general, the subscribe topics of the MQTT server of the DIVUS KNX IQ contain the keyword ***status***, while the publish topics contain the keyword ***request***. Those with status are automatically updated as soon as there is an external value change or as soon as a value change has been requested by the client itself via a publish and has been successfully applied. Those for publishing are further divided into those of type (request/)get and those of type (request/)set.

PAYLOAD

Value changes and other optional parameters are added to the topic with the so-called ***payload***. The parameters of the individual elements (element-id, name, type, functions)

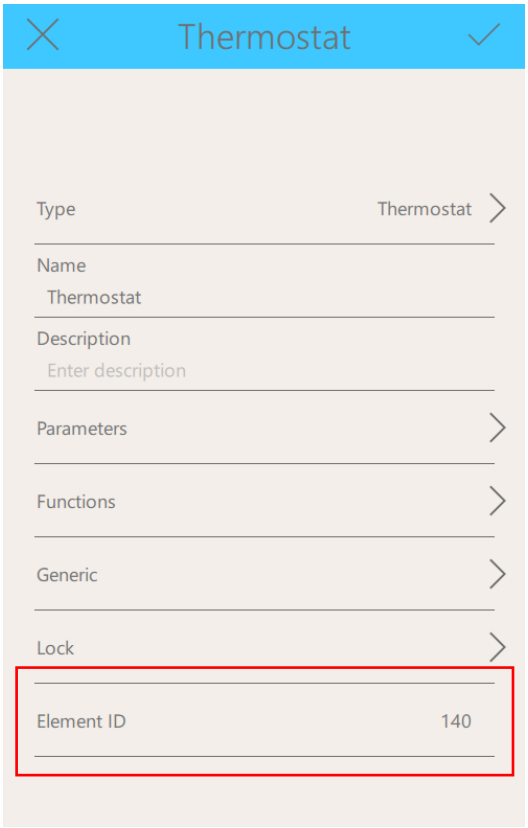
and special parameters (uuid, filters) are to be used here. Although there are several options, the payload is shown formatted as JSON in this manual. JSON uses brackets and commas to represent data of any structure and thus minimises the size of the data packets to be transmitted. More details about payloads can be found later in the manual.

FILTER

For special purposes, it is possible to filter according to the type of function, e.g. to address only on/off i.e. 1-bit switches. The ***filters*** parameter in the payload is used for this purpose. Filtering is currently only possible by function type.

ELEMENT-ID

To be able to address individual elements, their ***element ID*** is required. This can be found in VISION in the element properties menu or can also be read directly from the data that is displayed in front of each available element in the general subscribe of the MQTT Explorer (elements there are listed alphabetically by element ID).



2 Configuration for the API access

2.1 CONFIGURING VISION FOR API USER ACCESS

In VISION as an administrator, go to Configuration - User/API Access Management, click on Users/API access and right-click on API User (or press and hold) to open the editing window.

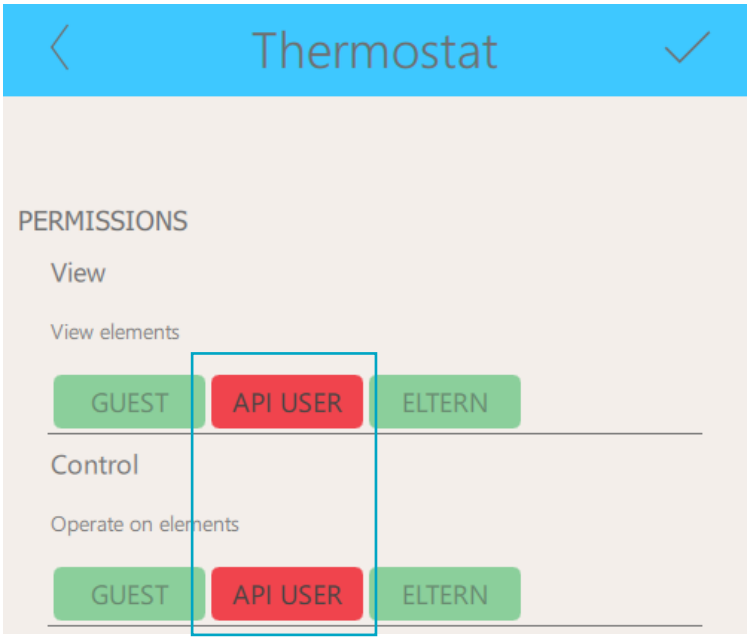
There you will find these parameters and data:

Enable (checkbox)	The user is first enabled here. Default is disabled
Username	This string is required for access via API - copy it from here
Password	This string is required for access via API - copy it from here
Permissions	The default rights for reading and writing the values of the VISION elements can be defined here, i.e. what is defined here applies to all existing and future elements. If you only want to allow access to individual elements, you should not change these default rights.

2.2 PERMISSIONS ON INDIVIDUAL ELEMENTS

It is recommended that you do not grant API access to the entire project, but only to the desired elements. Proceed as follows:

1. log in to VISION as an administrator
2. select the desired element and open its settings menu (right-click or keep pressed, then Settings)
3. under the menu entry General - Permissions, activate "Override default permissions" and then go to the sub-item Permissions, which shows the permissions matrix.



4. activate the *control* permission here, which also enables the *view* permission directly. If you only want to read data via the API access, it is sufficient to enable the view permission.

5. repeat the same procedure for all the elements you want to access.

3 Connection via MQTT

3.1 INTRODUCTION

As an example, we will demonstrate access via the MQTT API of the DIVUS KNX IQ with a relatively simple, free software called MQTT Explorer (see chap. 1.1), which is available for Windows, Mac and Linux. A basic knowledge and experience with MQTT is implied.

3.2 DATA REQUIRED FOR THE CONNECTION

As mentioned earlier (see section 2.1), the username and password of the API user are required.

Here is an overview of all the data that must be collected before a connection is established:

Username	Read out on the detail page of the API user
Password	Read out on the detail page of the API user
IP address	Read out in the launcher settings under General - Network - Ethernet (or via Synchronizer)
Port	8884 (this port is reserved for this purpose)

3.3 FIRST CONNECTION WITH MQTT EXPLORER AND GENERAL SUBSCRIBE

Normally, MQTT distinguishes between the activities *subscribe* and *publish*. MQTT Explorer simplifies this by automatically subscribing to all available topics (topic #) when the first connection is made. As a result, the tree that leads to all available elements (i.e. API user access granted) can be seen directly in the left-hand area of the MQTT Explorer window after a successful connection. To enter further subscribe topics or to replace the # with a more specific topic, go to *Advanced* in the connection window.

The topic shown on the upper right looks something like this:

```
local/api/v1/7f4x0607849x444xxx256573x3x9x983/status/elements/objects_list
```

where 7f4x0607849x444xxx256573x3x9x983 is the API username and objects_list contains all the available elements.

This topic is always kept up to date i.e. any value changes are reflected there in real-time. If you only want to subscribe to individual elements, enter the element ID of the desired element after objects_list/.

Note: This type of subscribe roughly corresponds to the logic behind the KNX feedback addresses; it shows the current status of the elements and may be used to check whether the desired changes have been successfully applied. If you only want to read out data but not change it, this type of subscribe is sufficient.

A single simple element looks something like this in JSON notation:

```
{
   "data": {
      "functions": {
         "onoff": {
            "value": "1"
         }
      }
   },
   "name": "Light Kitchen On/Off",
   "type": 0
   }
}
```

Note: All values have the syntax shown above e.g. { "value": "1" } as the output of the subscribe topics, while the value is written directly in the payload to change a value (i.e. for publish topics) - the brackets and "value" are omitted e.g. "onoff": "1".

4 Advanced commands

4.1 INTRODUCTION

There are 3 kinds of topics in general:

1. Subscribe topic(s) to see the available elements and to get real-time value changes
2. Subscribe topic(s) to get the answers to (the client's) publish requests
3. Publish topic(s) to get or to set elements with their values

We shall later refer to these kinds using the numbering shown here (e.g. topics of type 1, 2, 3). More details in the following sections and in chap. 4.2.

4.1.2 SUBSCRIBE TOPICS TO SEE THE AVAILABLE ELEMENTS AND TO GET REAL-TIME VALUE CHANGES

These have already been described in chap. 3.3

4.1.3 SUBSCRIBE TOPICS TO GET THE ANSWERS TO THE CLIENT'S PUBLISH REQUESTS

This kind of topics is optional. It allows to

- open a unique communication channel with the MQTT server by using an arbitrary client_id. More about that in chap. 4.2.2
- get the result of publish requests on the corresponding subscribe topic: success or failure with error code and message.

There are different topics to get answers to get or to set publish commands. The corresponding difference in their paths is the "status/get" vs "status/set" part. More about this in chap. 4.2.

Once you get the needed topics for your system straight, you *may* decide to remove this step and directly use publish topics.

4.1.4 PUBLISH TOPICS TO GET OR TO SET ELEMENTS WITH THEIR VALUES

These topics use a path similar to the ones for subscribing – the only change is the word "request" in place of the "status" used to subscribe. Complete topic paths are shown later in chap. 4.2.2

A get topic will request to read the MQTT server's elements and values. The payload may be used to filter based on the function type of the elements.

A set topic will request to change some parts of an element, as detailed in its payload.

4.2 PREFIX FOR COMMANDS AND CORRESPONDING RESPONSES

4.2.1 SHORT EXPLANATION

All commands that are sent to the MQTT server have a common initial part, namely:

```
local/api/v1/[API username]
```

which we may call general prefix.

Subscribe topics to get the answers to our value change requests will all start with

```
[general prefix]/[client_id]/status/
```

followed by get or set depending on the purpose.

```
[general prefix]/[client_id]/status/get

[general prefix]/[client_id]/status/set
```

Publish topics will all start with

```
[general prefix]/[client_id]/request/
```

and again, followed by get or set.

```
[general prefix]/[client_id]/request/get

[general prefix]/[client_id]/request/set
```

4.2.2 DETAILED EXPLANATION

The real-time topics (type 1) will have the general prefix (see above) then followed by

```
/status/elements/objects_list
```

and objects_list, which as the name says contains the available elements, may be followed by a # to show them all or by an element ID to show just one specific element. e.g.:

```
local/api/v1/[API username]/status/elements/objects_list/#          //show them all
```

or

```
local/api/v1/[API username]/status/elements/objects_list/140        //show just the element with ID 140
```

With a few changes, starting from the topics above, we obtain those of type 2 to subscribe to our own channel characterized by a client_id of our choice. So, we'll add the client_id after the API username

```
local/api/v1/[API username]/[client_id]/
```

and, depending on the type of answers we want to receive, we'll use get or set after the *status* keyword e.g.

```
local/api/v1/[API username]/[client_id]/status/get/elements/objects_list/140
```

or

```
local/api/v1/[API username]/[client_id]/status/set/elements/objects_list/140
```

Note: as mentioned before, these topics must be entered under *Advanced* in the case of the MQTT Explorer.

Note: these topics will not return anything until we start to publish something to them.

For type 3 topics to get or to set values on the KNX IQ's elements, you just need to substitute ***status*** with ***request*** in the type 2 topics e.g.:

```
local/api/v1/[API username]/[client_id]/request/get/elements/objects_list/140
```

or

```
local/api/v1/[API username]/[client_id]/request/set/elements/objects_list/140
```

For set commands, the payload obviously plays the main role as it will contain the desired changes (i.e. changed values for the element's functions).

Warning: Never use the **retain** option in your type 3 commands as it may cause issues on the KNX side.

4.3 EXAMPLE: PUBLISH FOR CHANGING A SINGLE ELEMENT'S VALUE(S)

The simplest case is to want to change the value of one of the elements shown by the general subscribe (see previous chapter).

Generally speaking, changing/switching a function of VISION via MQTT consists of 3 steps, not all of which are absolutely necessary, but we nevertheless recommend carrying them out as described.

1. The topic that contains the function we want to edit is subscribed using a custom client_id
2. The topic for editing is published together with the payload with the desired changes using the client_id chosen in 1.
3. To check, you can then see the answer in topic (1.) - i.e. whether (2.) worked or not
4. In the general subscribe, where all values are updated when changes are made, you can see the desired value change(s) if everything has worked out fine.

The steps to do this are:

1. select a client_id e.g. "Divus" and insert it in the path after the API username

   ```
   local/api/v1/7f4x0607849x444xxx256573x3x9x983/Divus/status/set/elements/objects_list/41
   ```

   This is the complete topic for subscribing to your own communication channel with the MQTT server. This tells the server where you expect the responses to the changes you intend to send. Notice the *status/set* part which defines a. that it is a subscribe topic and b. that it will get the answers to set type commands.

2. The publish topic will be the same except for switching the status-request keywords

```
local/api/v1/7f4x0607849x444xxx256573x3x9x983/Divus/request/set/elements/objects_list/41
```

3. what the change should consist of is written in the payload. Here are some examples.

   - Switching **off** an element that has the on/off function (1 bit):

```
{
   "data": { "functions": { "onoff": 0}
}
```

   - Switching **on** an element that has the on/off function (1 bit). In addition, if several such commands are started from the same client, the *uuid* parameter ("unique ID", is usually a 128-bit string formatted as 8-4-4-4-12 digits hex) can be used to assign the response to the corresponding query, as this parameter - if present in the query - can also be found in the response.

```
{
   "uuid": "550e8400-e29b-41d4-a716-446655440000",
   "data": { "functions": { "onoff": 1}
}
```

   - Switching on and setting the brightness of a dimmer to 50%

```
{
   "uuid": "550e8400-e29b-41d4-a716-446655440001", // optional
   "data": { "functions": { "onoff": 1, "dimming": "50" }
}
```

   The answer to the topic shown and subscribed to above (its payload, to be precise) is then, for example.

```
{
   "uuid": "550e8400-e29b-41d4-a716-446655440001", // only if set in subscribe
   "data": {
     "functions": {
          "onoff": "202",
          "dimming": "404"
     }
   }
}
```

   The above response is an example in the case of a correct payload, although the element has no dimming function.

   If there are more serious problems leading the payload not to be interpreted correctly, the response will look like this (e.g.):

```
 {
   "uuid": "550e8400-e29b-41d4-a716-446655440001", // only if set in subscribe
   "error": "400",
   "message": "BadRequest, Non-compliant topic"
}
```

   See chapter 5.1 for an explanation of the error codes and messages – but in general, like for http, 200 codes are positive answers while 400 are negative.

4.4 EXAMPLE: PUBLISH FOR CHANGING MULTIPLE ELEMENTS' VALUES

The procedure is similar to the one shown before to change a single element. The difference is that you omit the element_id from the topics and then indicate the set of element_ids in front of the data inside the payload. See the syntax and structure below.

1. like before, you start by subscribing to the set topic using a custom client_id ("Divus" here)

```
local/api/v1/7f4x0607849x444xxx256573x3x9x983/Divus/status/set/elements/objects_list
```

2. the editing is then done to the corresponding publish topic

```
local/api/v1/7f4x0607849x444xxx256573x3x9x983/Divus/request/set/elements/objects_list
```

   using a payload structured like this:

```
{
   "uuid":"request_uuid",
   "data": {
      "8": {
               "functions" {
                  "onoff":"1",
                  "dimming":"50"
               }
           },
      "9": {
               "functions" {
                  "onoff":"1"
               }
           }
   }
}
```

4.5 FILTER BY FUNCTION TYPE IN QUERIES

The *filters* parameter in the payload allows only the desired function(s) of an element to be addressed. The on/off function of a switch or dimmer is called "onoff", for example, and the corresponding filter is defined in this way:

```
{

   "filters": { "functions": ["onoff"]}

}
```

The answer then looks like this, for example

```
{
   "data": {
       "functions": {
         "onoff": {
            "value": "1"
         }
```

```
        },
        "name": "Dimmer FL",
        "type": 1
    }
}
```

The square bracket indicates that you can also filter by several functions, e.g.

```
{

   "filters": { "functions": ["temperature", "humidity"]}

}
```

leads to an answer like this:

```
{
  "data": {
    "functions": {
      "humidity": {
        "value": "58.6"
      },
      "temperature": {
        "value": "19.87"
      }
    },
    "name": "Thermostat",
    "type": 5
  }
}
```

5 Appendix

5.1 ERROR CODES

Errors in MQTT communication result in a numerical code. The following table helps to break it down.

action	error code	meaning
Publish	202	Accepted (function processed) - N.B. This is not an error code
Request	400	Incorrect query (there is an error in the content of the query)
Request	401	Not authorised (problems with the rights of the API user)
Publish	404	Not found (refers to element ID or function)
Request	405	Topic not valid
Request	409	Conflict (error with the filter type)

5.2 PARAMETERS OF THE PAYLOAD

The payload supports different parameters depending on the context. The following table shows which parameters can occur in which topics.

topic type	parameter	sub-parameter	explanation
request	`data`		Contains the available functions with the respective values `{ "data": { "functions": { "cooling_onoff": { "value": "0"}, … } } }`
		functions	Contains one or more functions with the respective value. Which functions are available depends on the element type and its configuration. You can see this in the subscribe topics. `{ "functions": {`

			<pre>"cooling_onoff": {
			"value":"0"
			},
			"temperature": {
			"value":"23.48"
			}
			}
			}</pre>

	name		Name of the element, as defined in VISION <pre>"name": "Thermostat"</pre>
	type		Numerical value that characterises the element type e.g. On/Off elements have type 0, dimmer elements type 1, thermostat elements type 5 etc. <pre>"type": 5</pre>

subscribe or request	uuid		Optional parameter to be able to assign the future response to the query (because the same ID can be found in the response). Plays a role if many queries can also go from the client to the server at the same time and there is a risk of not being able to assign the responses to the queries.
			
			<pre>{
			"uuid": "550e8400-e29b-41d4-a716-446655440001"
			}</pre>

	filters		Optional parameter for filtering (currently only by function type) if you only want to address certain functions directly. Several functions can be used for filtering, separated by a comma. The entire filter is then read as "return only the functions and their values that I list here (from one or more elements, depending on the topic).
			
			<pre>{
			"filters": { "functions": ["onoff"]}
			}</pre>

5.3 VERSION NOTES

5.3.1 VERSION 1.00

News:

- First publication

5.4 NOTES