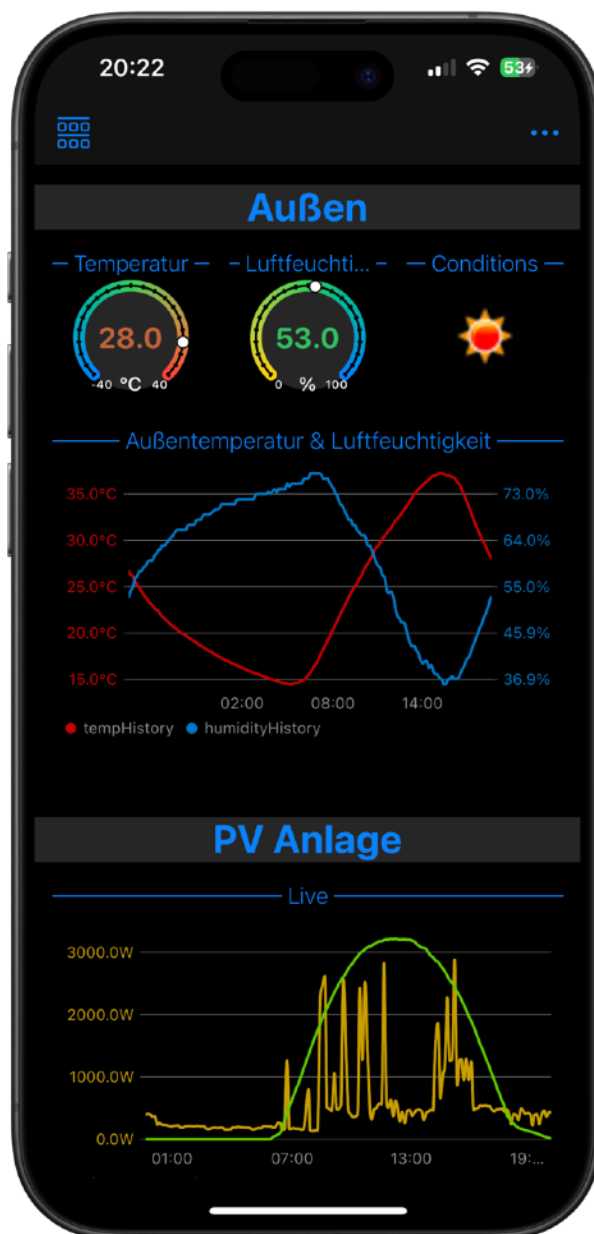


Visual V2.00

SmartHome App for iPhone & iPad



Privacy Policy	4
Data collected	4
Analytics (V1.6)	4
Personal data	4
Contact Person	4
Basic Principles	5
Overview	5
The Dashboard	6
Visual - Step by Step	9
Create Endpoints	9
Configure Endpoints	11
Add Widgets	11
Affect widget order within a category	14
Widget Management	14
Configure Widget	14
Port Connections	16
Individuelle Widget Parameter	16
Organize Dashboard	18
Endpoints (Simple)	19
Homekit	19
Homematic	19
Philips Hue	20
URL	20
Time	20
OpenWeatherMap	20
HTML Widget	20
PVOutput.org	21
Endpoints (Experts)	22
MQTT Client	22
HTTP Client	25
UDP Client	29
Lambda Functions (Advanced Features)	30
Lambda Configuration	31
Global Settings	34
iCloud Sync	35

Appendix A: Simplified JSON Path Syntax	36
Appendix B: Data Format For Diagramms	37
Contact	37

Privacy Policy

Data collected

The following data can be configured by the user in Visual to enable the function of the app:

- Configuration data for the external devices and web services (e.g., connection data, user names, passwords)

This data is only stored on the iOS device or synchronized via the user's iCloud account between iOS devices. The data will not be passed on to the developer or third parties. There is also no server-side data processing.

Analytics (V1.6)

Visual collects since version 1.6 anonymously app usage data only **after the user consented**:

- Used endpoint types
- Used widget types

No personal data is collected and the user consent can be rejected (or given) at any time in the app settings.

Personal data

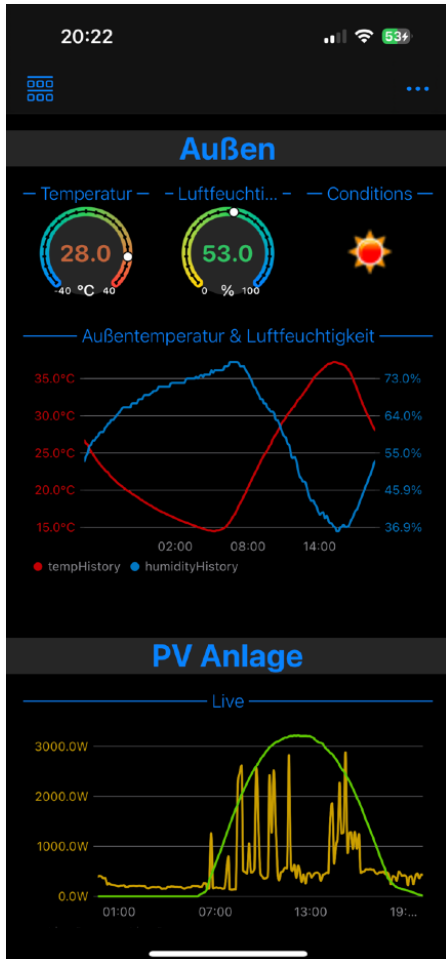
Personal data (eg name, e-mail address, telephone number) will only be collected, stored and processed if you provide the developer with this data through an explicit e-mail request. The developer uses this data solely for the fulfillment and processing of your request or for the transmission of directly related information. Personal data will never be disclosed to third parties.

Contact Person

Please direct questions to the developer of the app: me@andreas-binner.de

Basic Principles

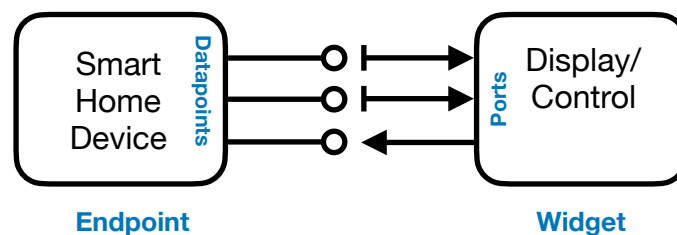
Overview



Visuals manages a so-called "Dashboard" which you can also see directly after the start of the app. The Dashboard is a scrolling list of widgets. A widget can be a pure display or an operating element such as a switch.

The counterpart to the widgets are the "endpoints". An endpoint represents an external data source such as a smart home device or web service. All endpoints have the commonality of being accessible through the network.

Widgets have "ports" and endpoints have "datapoints". One port represents a dedicated data channel. Datapoints have one direction ("read only" or "read / write") and one type (boolean, integer, float, percent, data series, color, ...). For example, a weather sensor may have one datapoint each for temperature and humidity ("read only" and "float" type).

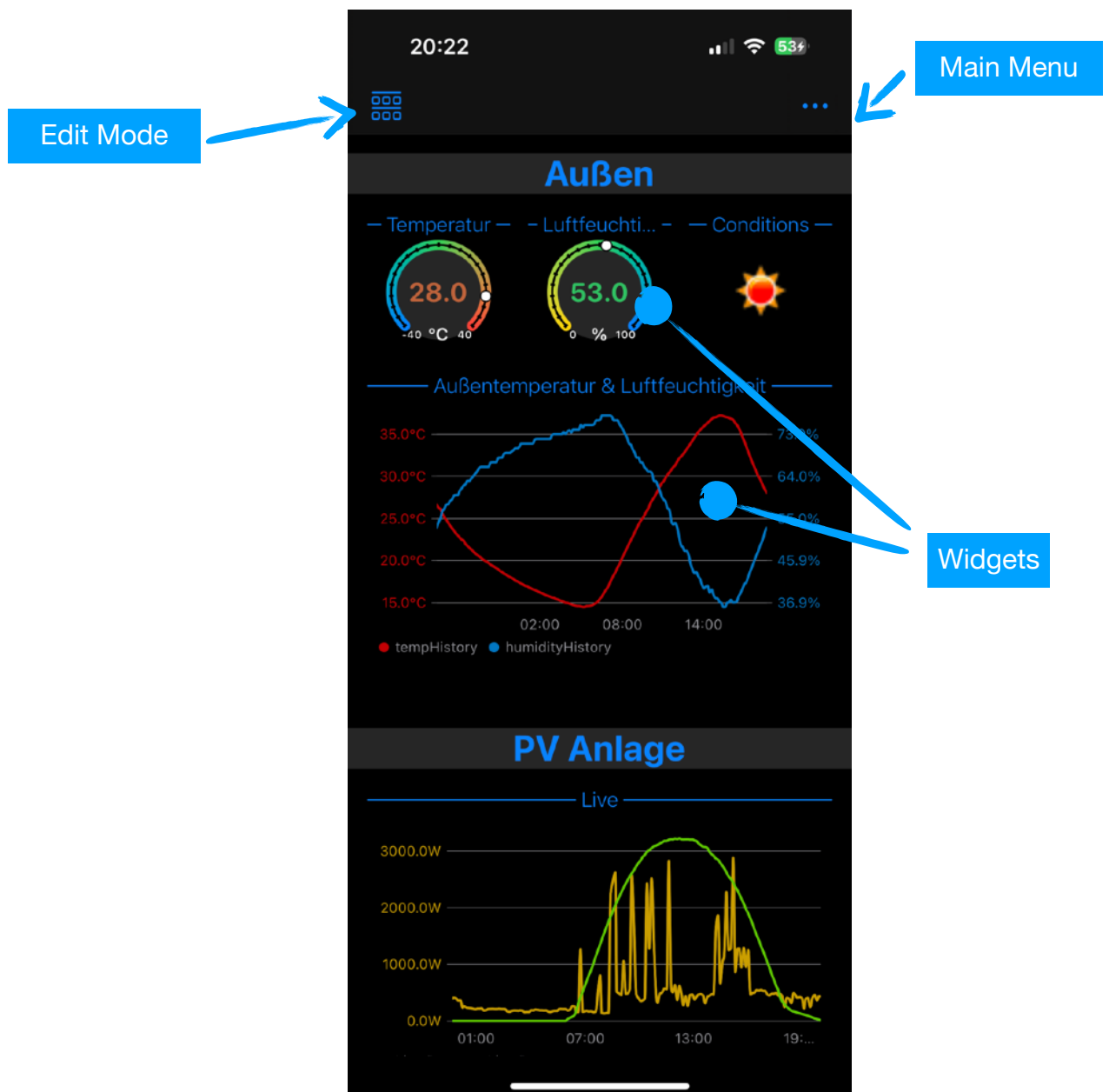


A widget on the other side can also have exactly one port (eg. a simple gauge display) or multiple ports for multiple data sources (eg a line chart). Also, controls can have multiple ports ("read / write") to output a value in different representations (eg, the color picker for RGB and HSV color space) or simply to send a value to multiple endpoints. During configuration widget ports are connected to datapoint of endpoints.

The Dashboard

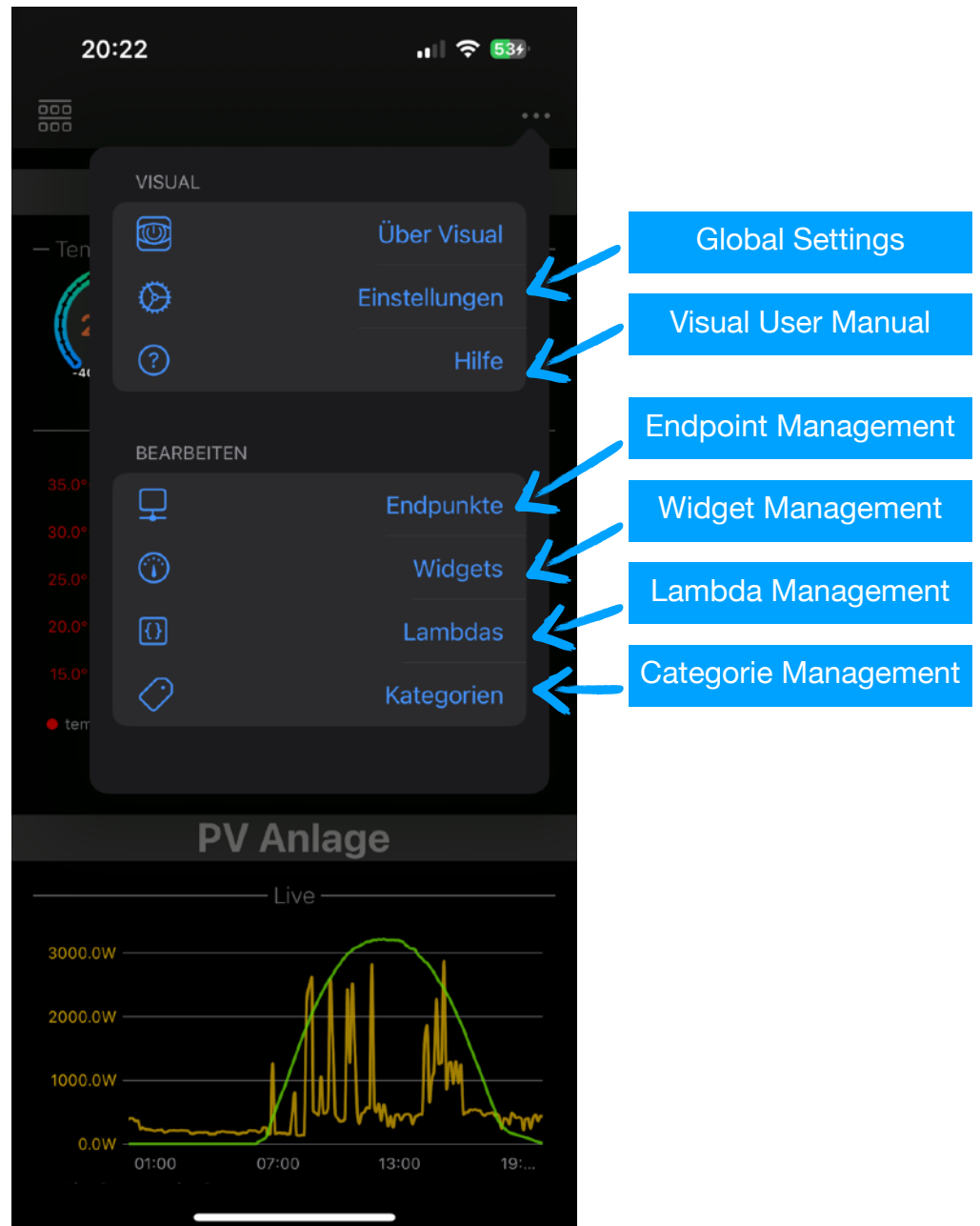
The dashboard is the central view in Visual. From here you can also reach all important other views via the toolbar at the bottom:

- Enter edit mode
- Main menu



Main Menu

Via the main menu you can access all important administration and settings menus as well as the user manual (this text). All settings are explained in detail below.



Grid

The dashboard displays the configured widgets in a fixed grid. Widgets can be up to 3 units wide and 2 units tall. The following 6 sizes are available (depending on the widget type):



Alignment

The dashboard automatically adapts to the screen size and orientation.

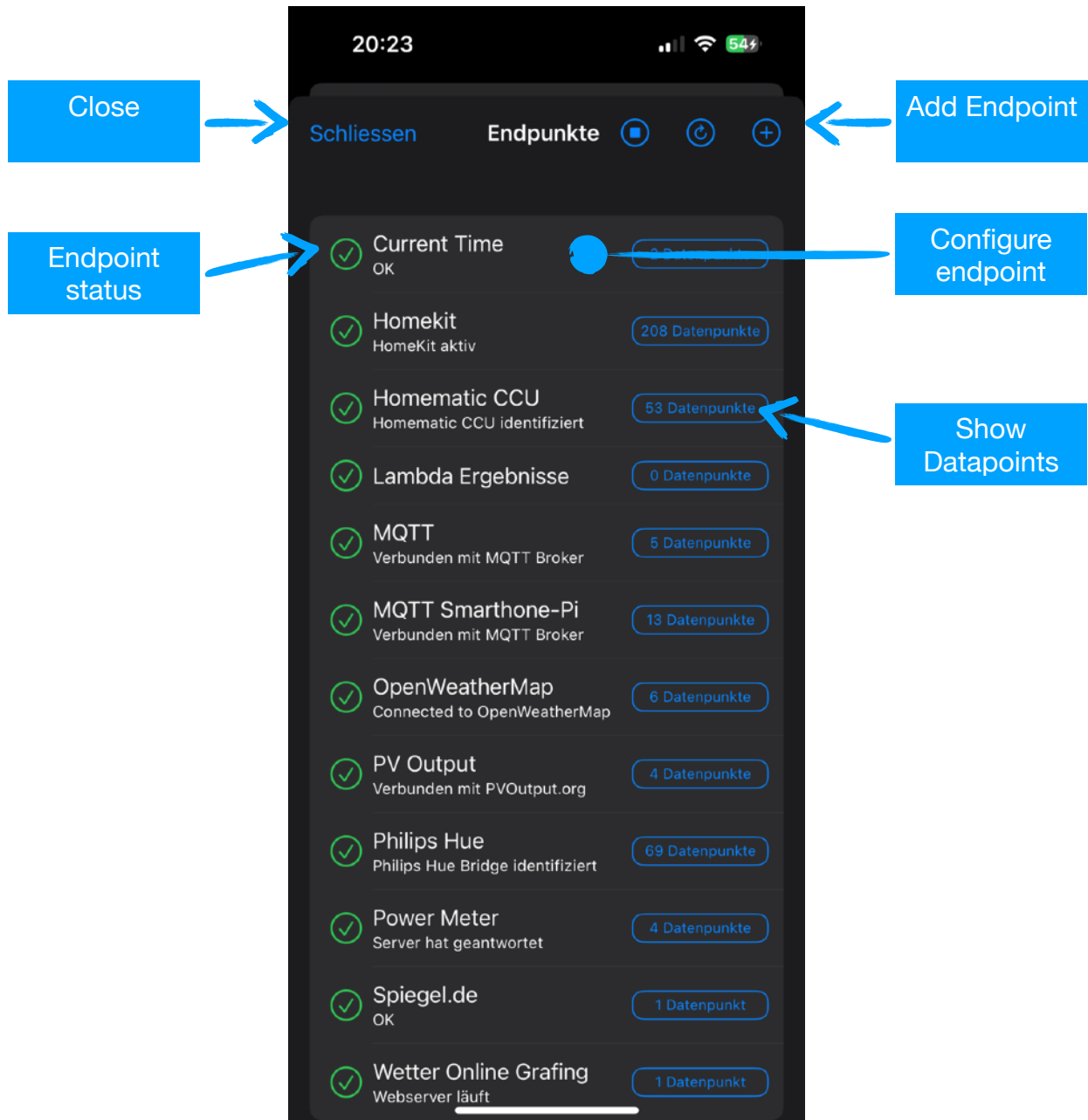
Here, the Apple-defined "size class" is crucial. All iOS devices in portrait format have the size class "Compact". Landscape devices are either "Compact" (iPhones) or "Normal" (iPad, iPhone Max).

In the class "Compact" the dashboard has by default a width of 3 units. In the class "Normal" the number doubles to 6 units. But this can be configured in the Dashboard edit mode.

Visual - Step by Step

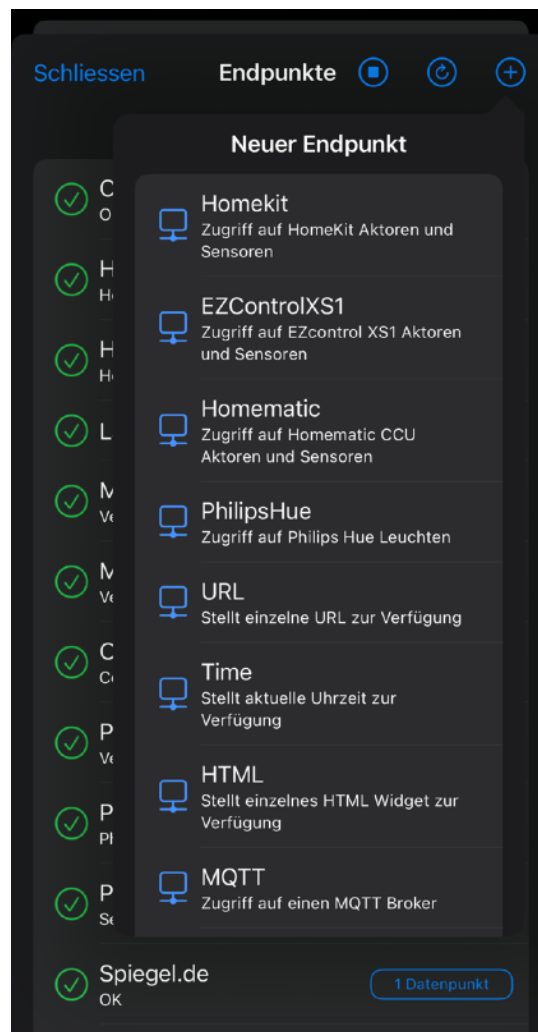
Create Endpoints

To create a new endpoint, select "Endpoints" in the main menu



To do this, first open the endpoint manager in the dashboard. Add an endpoint by simply tapping on the "+" button and then selecting the appropriate entry in the list. The selection dialog can also be closed without adding, by tapping outside the dialog.

Currently, Visual supports the following endpoints:



Homekit	Access to Homekit devices and services
Homematic	Access to devices via Homematic CCU
Philips Hue	Access to Philips Hue controlled devices
URL	Static URL (in combination with HTML Widget)
Time	Provides current time of day
OpenWeatherMap	Access to OpenWeatherMap data
HTML Widget	Provides HTML Widget
MQTT Client	Generic access to MQTT Broker (JSON Payload)
HTTP Client	Generic access to HTTP Server (JSON Payload)
UDP Client	Generic UDP client sending datagrams (Binary, JSON)

A detailed description of the individual types can be found in the chapter "Endpoints".

Configure Endpoints

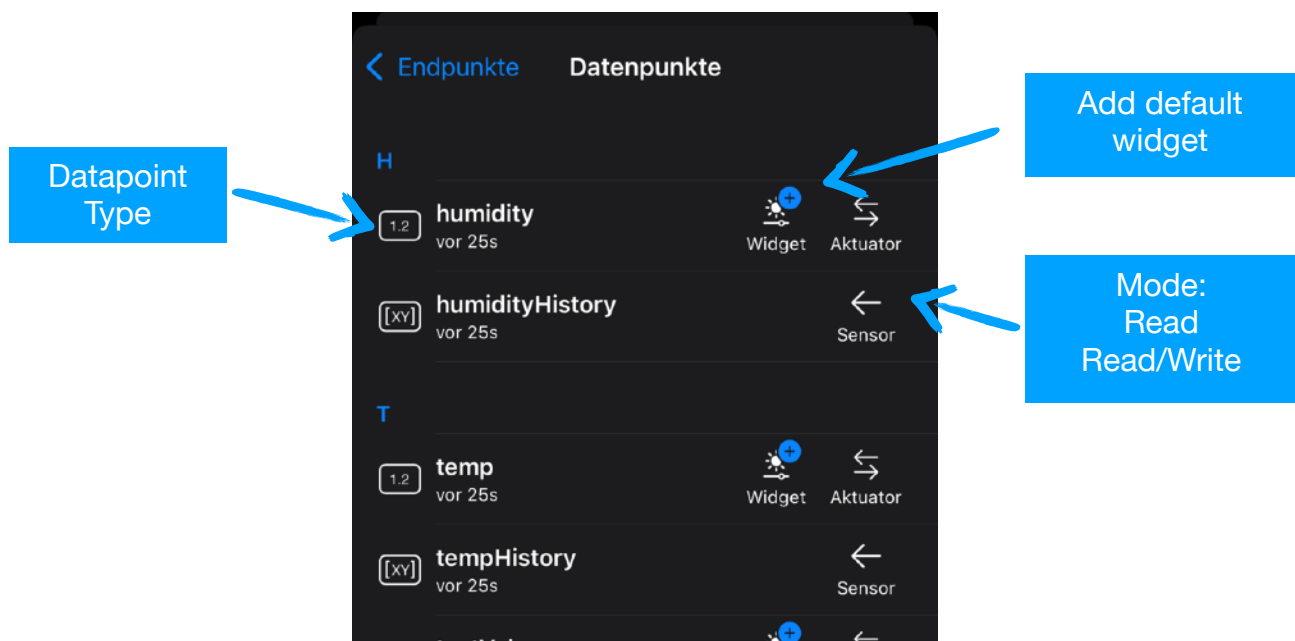
To configure an endpoint, simply tap on the corresponding entry.

Depending on the endpoint type, there are different configuration parameters (see chapter "Endpoints"). All endpoints have the parameter "Name" to give the list entry a unique name.

Important: For all endpoints communicating with an external device or service, it is at least necessary to provide a URL or IP address.

After an endpoint connects, you can see the endpoint's datapoints. Simply tap on the blue button showing the number of datapoints endpoint name:

6 Datenpunkte



You can directly add a default widget for some datapoint types. To do so tap on the "Widget Plus" button.

Add Widgets

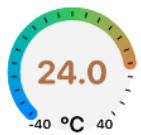
First enter the "Edit Mode" by tapping on the Dashboard icon. All widgets will show a moving dashed border and two button at the bottom".

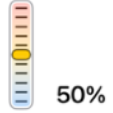






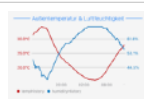
Now tap on the "Add widget" icon in the title bar. In the appearing dialog, tap on the corresponding entry in the list. If you are in the single column dashboard, additionally the category where to put the new widget needs to be selected.



In addition the grid width of the dashboard can be changed. Note the grid width options depend on the so called "size class" (*compact* e.g. iPhone and *normal* e.g. iPad)!

WIDGET TYPES

Gauge	Display of a single value gauge style (with unit)	
-------	---	---

Level	Display of a single value bar style (with unit)	
Status Indicator	Display of a binary value (e.g. 0/1 or on/off) or a color (hue)	
Text	Display of a single line of text	Hallo
WebView	Display of HTML content (e.g. an HTML widget or an external website)	
Selection	Select a single option from a list	<div>Rot</div> <div>Grün</div> <div>Blau</div>
Switch	Simple on / off switch	
Button Array	Display of (up to 6) stateless buttons	<div>1</div> <div>2</div> <div>3</div> <div>4</div>
Slider	Horizontal slider controller	
Color	Color picker (or only display a color)	
Button	Stateless button	On
Image	Shows an image (received as data or loaded from URL)	
Line diagram	Display of (up to 8) data series. Note: Ports 1-4 and the ports 5-8 share a y-axis each! This means two different value ranges are supported.	

Affect widget order within a category

Select widget by tapping the title once. Now the position of the selected widget can be changed with the arrow buttons in the bottom toolbar. *Visual always tries to arrange the widgets within a category as compactly as possible (with few gaps).*

Therefore the position can't be chosen freely!

Alternatively a widget can be dragged after a long tap onto another widget which causes them to switch places.

Widget Management

As an alternative to the dashboard editing mode, you can also edit the widgets via the widget management. To do this, tap on "Widgets" in the main menu.

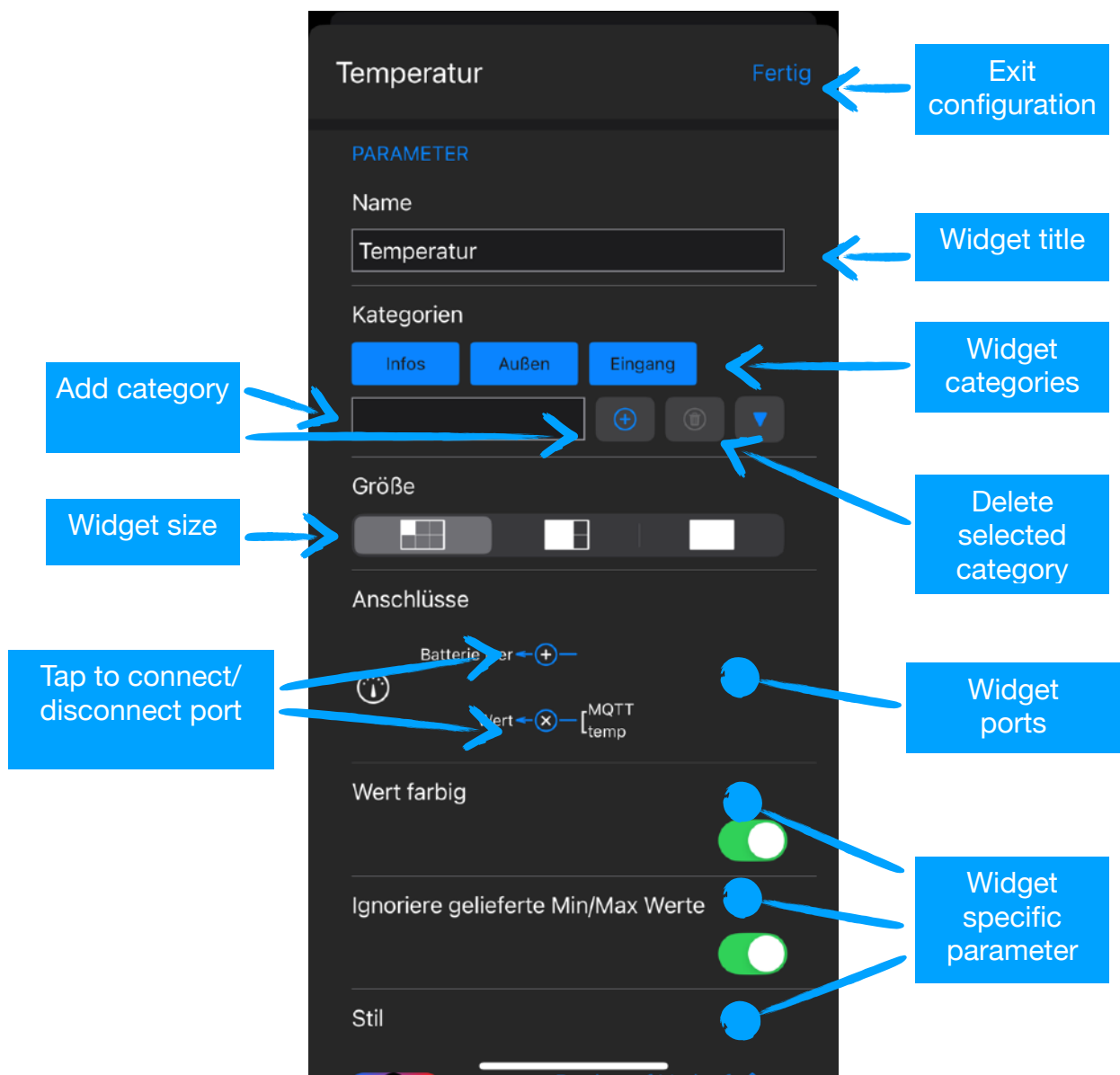


Widgets




Here you can see all widgets in a simple list. The order of the widgets can be easily changed by drag & drop - can also be moved between categories. By tapping a list entry, the setting for the widget opens.

Configure Widget



To configure a widget, just tap on the "gears" bottom of the widget in the bottom, right corner. Now the configuration dialog opens.

The following parameters are available for all widget types:

Name	The title of the widget
Size	<p>A widget can take one of 6 sizes (not all sizes are allowed on all widgets!)</p> 

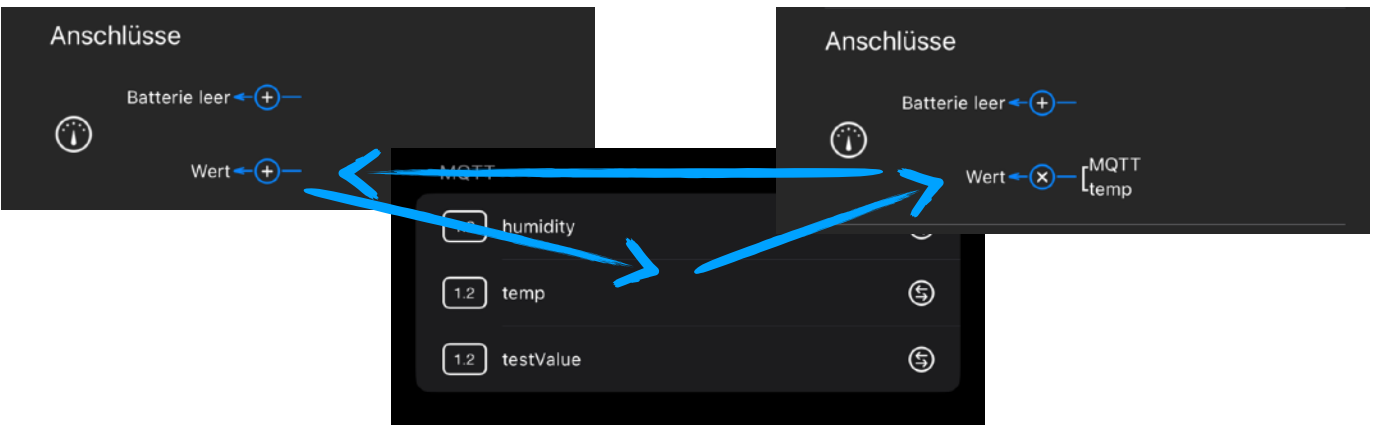
Categories	<p>Widgets can be assigned to one or more categories. Widgets appear in the Dashboard in all assigned categories.</p> <p>Attention: If a widget is no longer assigned to a category, the widget will be deleted!</p>
Ports	<p>Here are all ports of the widget listed and it can be linked to an datapoint (or deleted)</p>

Port Connections

Tap ⊕ on a port to connect to a datapoints.

Note: Only compatible datapoints are offers for selection!

Tap ⊗ to disconnect the port.



Individuelle Widget Parameter

Each widget type has special parameters with which you can influence the appearance and behavior.

Gauge	Ignore delivered min/max values	<p>On: Default values (derived from the unit) are used for maximum and minimum value</p> <p>Off: The "min" and "max" value of the connected datapoint is used</p>
Level	Display type	<ul style="list-style-type: none"> • Normal: Grey bar • Green→Red: Gradient from green to red • Red→Green: Gradient from red to green • Peak Red: Above 80% Red

Status Indicator	Off Text	Text displayed in the "Off" state**
	On Text	Text displayed in the "On" state**
	On Color	Color in the "On" state (ignored if the "Hue" port is used!)
Text	Font size	Font size in point
	Use monospace font	The non-proportional system font is used
Selection	Selection list	List of possible values <i>Note: Depending on the connection type, a selection index (0...5) or the converted value is derived from the label (e.g. 18°C -> 18.0).</i>
Switch	Off Text	Text displayed in the "off" state**
	On Text	Text displayed in the "An" state**
Keymatrix	Selection list	List of possible values <i>Note: Depending on the connection type, a selection index (0...5) or the converted value is derived from the label (e.g. 18°C -> 18.0).</i>
Slider	Delay in ms	Minimum time between two sent new values
	Show values	Current value is displayed
	Number of steps	Step width of the controller
Colorpicker	Delay in ms	Minimum time between two sent new values
Button	Value to send	Value sent when the button is triggered: 0/0.0/0% or 1/1.0/100% (depending on the connection type)
	Button text	Text displayed in the button**
Line Chart	Interpolation	<ul style="list-style-type: none"> • Linear: Connection over a straight line • Stepped: Connection via steps • BezierCubic: Connection via Bezier curves

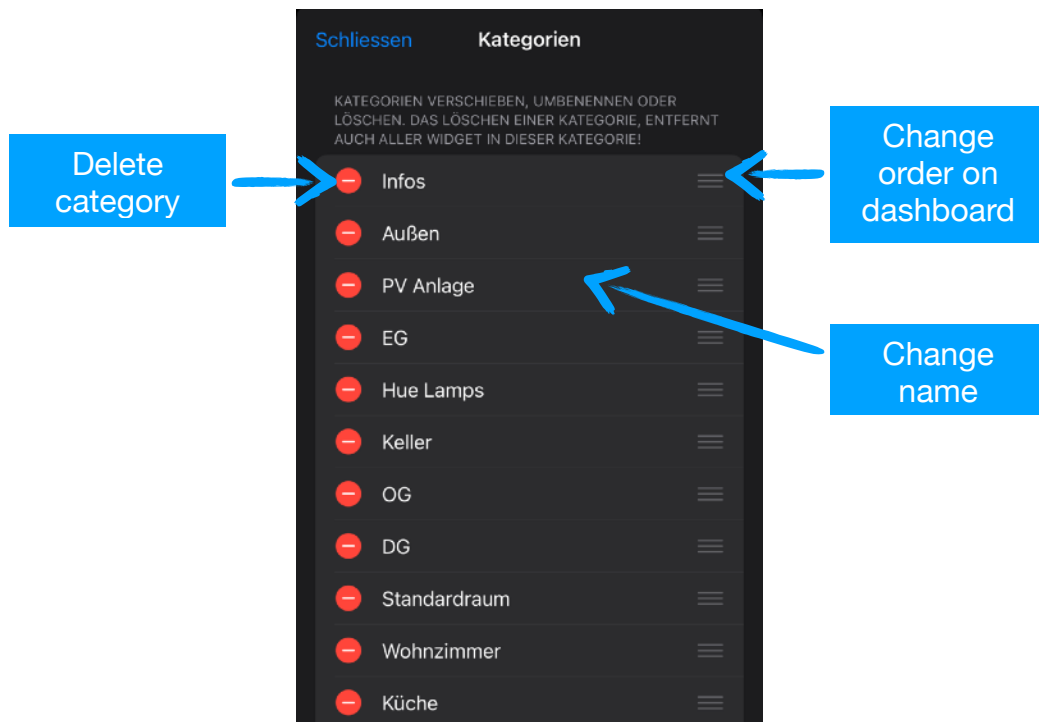
	Filled	Area under the curve is filled
	Autoscale y-axis	On: Scale derived from minimal and maximal value of the series Off: The "min" and "max" value of the connected datapoint is used
	Show points	Shows individual values as points
	First line color	Color of the first series of values
	Color distance	Define the colors of the following series
	Format X-Axis	<ul style="list-style-type: none"> • None: No X-axis • Time: Show x-values as time • Date: Show x-values as date • Minutes: Show x-values as minutes

** SF-Symbols are supported. To do this, instead of the text, specify the name of the SF symbol with a preceded '#'. For example, #power for the display of the following symbol: ⚡

Organize Dashboard

There are several ways to influence the appearance of the dashboard:

- **Change the configuration of the individual widget**
Many widgets have display options found in the widget configuration (see above).
- **Change the order of the categories**
To do this, select the "Category" entry from the main menu and change the order in the dialog by dragging and dropping it in the list. Here the categories can also be renamed.



Endpoints (Simple)

Visual supports a number of endpoints, which are described below.

Homekit

Provides all Homekit devices and their "Services/Characteristics" in Visual as an endpoint.

Important: Using this endpoint requires that you give Visual the permission to access Homekit devices!

Homematic

This endpoint allows access to homematic devices. This requires a CCU or CCU2 gateway with installed XML-API patch. The following parameters have to be configured:

URL	URL of the CCU
Login	Login name (if assigned)
Password	Password (if given)
Time interval for reloading	The data from the CCU is periodically read out in the specified time interval (in seconds)

Hide unreadable datapoints	Homematic datapoints that are unreadable are ignored
-----------------------------------	--

Philips Hue

With this endpoint you can access Philips Hue lamps. For this a Philips Hue gateway must be present. The following parameters have to be configured:

URL	URL of the Hue Gateway
Time interval for reloading	The data from the gateway is periodically read out in the specified time interval (in seconds)

Important: On the first start you will be asked to pair the app with the Hue Gateway. Please follow the instructions and press the pairing button on the Hue Gateway.

URL

This endpoint provides only one connection. This port provides a configurable URL. This can be used in conjunction with the WebView widget to display any web page in the dashboard.

Time

This endpoint provides only one connection. This port provides the current time as text. Currently, the only meaningful use is currently the connection with the text widget to display the time in the dashboard.

OpenWeatherMap

The OpenWeatherMap endpoint provides connections for the current temperature, humidity, wind force and wind direction. In the endpoint configuration you can specify the location for the current weather data and your OpenWeatherMap App-Key.

Note: You need to register at OpenWeatherMap first and generate an App-Key. The registration is free.

HTML Widget

In this endpoint you can deposit an HTML widget. HTML widgets are usually intended for embedding into a web page. They can be found on the Internet eg many weather sides are offering HTML widgets. The HTML code is simply copied to the designated field in the endpoint configuration via "Copy / Paste". The port then provides a local URL that can then be linked to a WebView widget.

PVOutput.org

This endpoint reads data from the photovoltaic web service pvoutput.org

Time interval for reloading	The data is requested periodically in the specified time interval (in seconds)
System-ID	System ID (taken from the personal PVOutput.org profile)
API-ID	API-ID (from the personal PVOutput.org profile)

Endpoints (Experts)

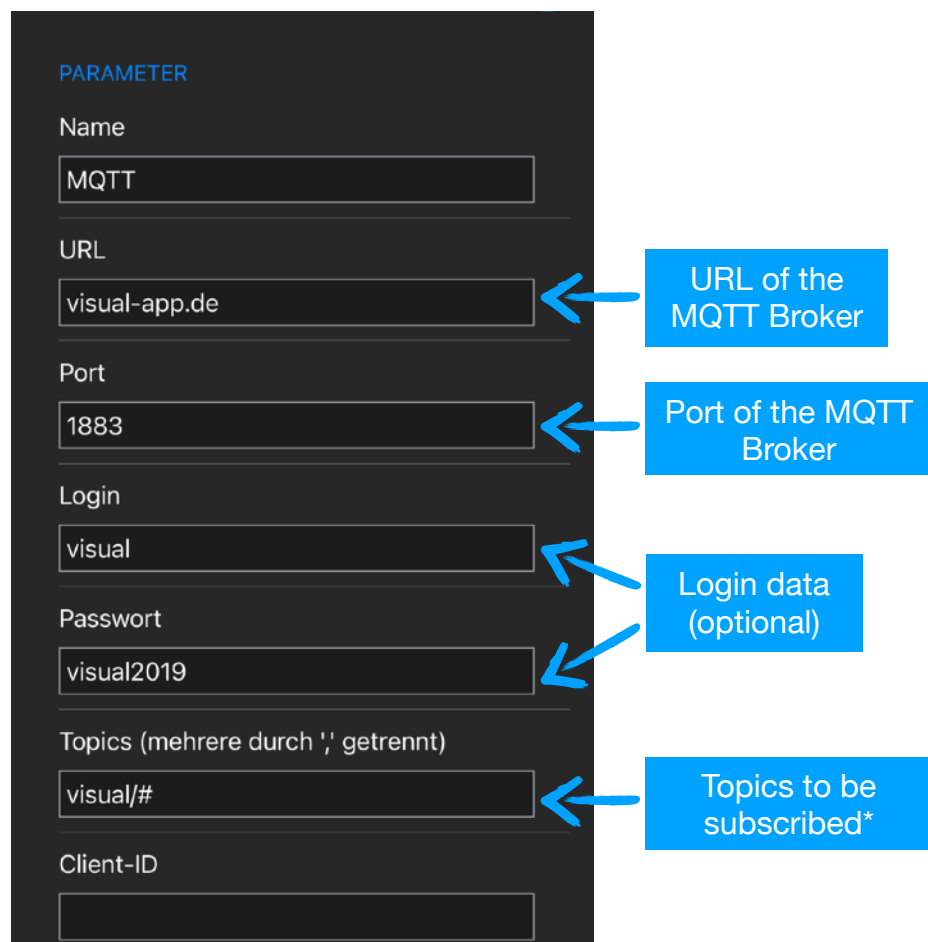
MQTT Client

MQTT CLIENT CONFIGURATION

This endpoint allows to subscribe or publish data to a MQTT broker.

Important: the MQTT topic payloads have to be in JSON format!

Basic Configuration



The image shows a configuration form for an MQTT client. The form has a dark background with white text. The title 'PARAMETER' is at the top in blue. Below it are several input fields with labels to their left. Blue arrows point from blue text boxes on the right to specific input fields. The input fields contain the following values: 'MQTT', 'visual-app.de', '1883', 'visual', 'visual2019', and 'visual/#'. The 'Client-ID' field is empty.

Parameter	Value	Annotation
Name	MQTT	
URL	visual-app.de	URL of the MQTT Broker
Port	1883	Port of the MQTT Broker
Login	visual	Login data (optional)
Passwort	visual2019	Login data (optional)
Topics (mehrere durch ',' getrennt)	visual/#	Topics to be subscribed*
Client-ID		

* Multiple Topics have to be comma separated. Wildcards ("#" oder "+") are allowed

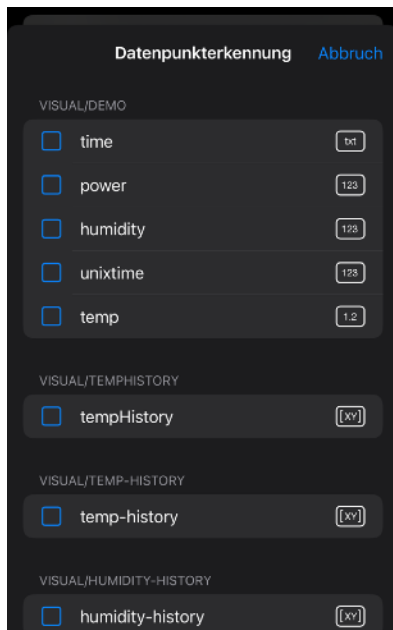
The menu bar shows if the connection has been successfully established. Only in this case the Port-Wizard can be used!



PORT-WIZARD



The wizard tries to fill in the data point configuration semi-automatically. To do this, the wizard "listens" to all subscribed topics and analyzes the payload. Then a list of possible data points is displayed. Here select the data points to be generated. Then tap on "Apply".



Important notes for the wizard to work:

- *The payload must be in JSON format!*
- *First enter all topic names in the "Topics" field to which the wizard should react!*
- *Data must be received, so make sure that the relevant topics are published while the wizard is running!*

MANUEL PORT CONFIGURATION

Type	Datapoint type
Name	Display name
Topic	Topic connected to this datapoint. <i>Note: No wildcards allowed!</i>
Send value	If the connection is to be used for publishing, the payload must be entered here. In this case, the placeholder %v is replaced by the actual value. Example: {"value":% v}
Receive value	Here it is specified where in the payload the connection value can be found. To do this, specify a JSONPath expression in angle brackets ("< >") - see examples below
Unit (Input)	Unit of the port - Static value or via JSONPath from the topic payload
Min (Input)	Minimal value of the port - Static value or via JSONPath from the topic payload*
Max (Input)	Maximal value of the port - Static value or via JSONPath from the topic payload*

* *always specify both "min" and "max"!*

Name

Datapoint type

Topic for this datapoint

Send value

Receive Value

Maximum value

Minimum value

Unit

Typ: Fließkomma 1.2

Name: temp
Menschenlesbare Beschreibung

Topic: visual/demo
Topic (keine Wildcards!)

SUBMIT

Wert senden: {"temp": %v}
Nutze den %v Platzhalter für den aktuellen Wert

SUBSCRIBE

Wert empfangen: <temp>
OK

Min:
Verwende einen statischen Wert, eine Ausdruck der mit '=' beginnt, oder einen JSON Pfad in <...> Klammern.

Max:
Verwende einen statischen Wert, eine Ausdruck der mit '=' beginnt, oder einen JSON Pfad in <...> Klammern.

Einheit: °C
OK

EXTRACTING VALUE FROM RECEIVED DATA

Static values can be entered directly in the fields (mostly for *Min*, *Max* and *Unit*). In most cases it is necessary to get the value dynamically from the JSON payload of the broker. This can be accessed with a simplified JSONPath notation on individual JSON properties (see Appendix A for details on the syntax). The JSON Path has to be put into "<...>" brackets!

DELETE PORT

Swipe left on the port, then tap the "Delete" button. Note: One connection must always be configured!

HTTP Client

In principle, this endpoint type is very similar to the MQTT client. Only HTTP is used here as the transport protocol instead of MQTT.

The screenshot shows a configuration form for an HTTP client. The form has a title 'PARAMETER' in blue. It contains several input fields and a toggle switch. Blue arrows point from external text boxes to specific fields in the form:

- 'Base-URL of the server' points to the 'URL' field, which contains 'http://power-pi.local/cgi-bin/meter.cgi'.
- 'Login data (optional)' points to the 'Login' field.
- 'Login data (optional)' also points to the 'Passwort' field.
- 'Load interval (sec)' points to the 'Zeitintervall zum Neuladen' field, which contains '30.0'.
- 'Authentication token (optional)' points to the 'Authentication-Token' field.
- 'Individual HTTP request per datapoint' points to the 'Individuelle Requests pro Datenpunkt' toggle switch, which is currently turned off.

DATAPoint CONFIGURATION

As with the MQTT client, the individual data points can be configured manually:

Name	Display name
Type	Datapoint type
Path	If the option " <i>Individual requests per datapoint</i> " is activated, this value is appended to the global URL. If the global URL does not end with a "/", the last path component is removed before appending. The placeholder "% v" for the value and "% i" for the unique identification of the datapoint can be used in this field.

Send value	If the connection is to be used for sending, the payload must be entered here. In this case, the placeholder %v is replaced by the actual value. Example: {"value":%v}
Receive value	Here it is specified where in the response payload the connection value can be found. To do this, specify a JSONPath expression in angle brackets ("< >") - see examples below
Unit (Input)	Unit of the port - Static value or via JSONPath from the response payload
Min (Input)	Minimal value of the port - Static value or via JSONPath from the response payload*
Max (Input)	Maximal value of the port - Static value or via JSONPath from the response payload*

* *always specify both "min" and "max"!*

The screenshot shows the configuration interface for a datapoint named '1_8_0'. The interface is in German and includes the following fields and annotations:

- Name:** The field contains '1_8_0'. A blue arrow points to it from a label 'Name' on the left.
- Datapoint type:** A dropdown menu is set to 'Fließkomma' with a '1.2' version indicator. A blue arrow points to it from a label 'Datapoint type' on the right.
- Send value:** The 'Wert senden (POST)' field contains the JSON payload '{"1_8_0": %v}'. A blue arrow points to it from a label 'Send value' on the left.
- Receive Value:** The 'Wert empfangen' field contains the JSONPath expression '<1_8_0>'. A blue arrow points to it from a label 'Receive Value' on the left.
- Minimum value:** The 'Min' field is empty. A blue arrow points to it from a label 'Minimum value' on the right.
- Maximum value:** The 'Max' field is empty. A blue arrow points to it from a label 'Maximum value' on the left.
- Unit:** The 'Einheit' field contains 'kWh'. A blue arrow points to it from a label 'Unit' on the right.

SERVER COMMUNICATION

Read data: This endpoint uses the HTTP GET method to read data from the server. The server response to the request must be in JSON format so that Visual can evaluate the data!

Write data: The HTTP POST method is used to trigger actions on the server. The payload of the request can be defined separately for each datapoint. The placeholder "% v" is replaced by the actual value of the widget port. Alternatively, it is possible to use HTTP GET. To do so, the "% v" placeholder must be used in the path field.

Examples:

Globale URL: <http://myserver/api/>

Read temperature via HTTP GET to "getTemperature"

Server response in JSON: { "temp": 22.0, "unit": "°C" }

Path	getTemperature
Send value	<temp>
Unit	<unit>

→ GET <http://myserver/api/getTemperature>

Read specific temperature via HTTP GET to "getTemperature"

Server response in JSON: { "temp": 22.0, "unit": "°C" }

Ident	Kitchen
Path	getTemperature&ident=%i
Send value	<temp>
Unit	<unit>

→ GET <http://myserver/api/getTemperature?ident=Kitchen>

Write temperatur via HTTP GET to "setTemperature"

Widget value is 20.0

Path	setTemperature?value=%v
Send value	

→ GET `http://myserver/api/setTemperature?value=20.0`

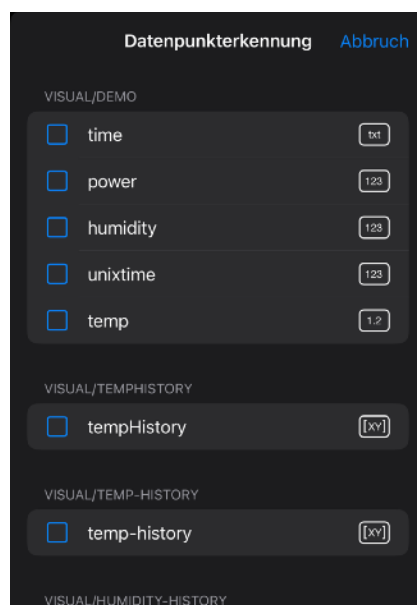
*Write temperatur via HTTP POST to "setTemperature"
Payload in JSON, Widget value is 20.0*

Path	setTemperature
Send value	{ "value": %v }

→ POST `http://myserver/api/setTemperature`
Payload: { "value": 20.0 }

DATAPOINT WIZARD

Alternatively, you can use the wizard. The global URL is used here to retrieve data from the server via HTTP GET



UDP Client

This generic endpoint allows to send UDP datagrams. The datagrams can be configured to send text based payloads (e.g. JSON) or binary data payload.

Note: This endpoint does not support receiving data!

DATAPOINT CONFIGURATION

The screenshot shows a configuration window titled 'PARAMETER' with a dark background. It contains four input fields: 'Name' with the value 'UDPClient', 'IP Adresse' with '127.0.0.1', 'Port' with '12345', and 'Big Endian' with a green toggle switch. Three blue arrows point from external labels to the configuration: 'Destination IP Address' points to the IP field, 'Port' points to the port field, and 'Endianness' points to the 'Big Endian' toggle.

BINARY PAYLOAD

In order to send a binary payload simply use the "0x" prefix in the "Send Value" configuration. In order to embed the current value of the datapoint, several placeholders are allowed:

Placeholder	Type
[b]	1-byte boolean
[u8], [u16], [u32]	Unsigned 8, 16 or 32 bit integer
[i8], [i16], [i32]	Signed 8, 16 or 32 bit integer
[f], [d]	Single or double precision float
[s]	UTF8 String

For any value types longer than one byte, the global "Endianness" setting is used to form the value in the binary buffer!

The following example will insert the current integer datapoint as a 16-bit value into a binary payload. The payload starts with the byte "0xAA" followed by the 16-bit value and ends with the byte "0xBB":

UDPClient Datapoint 'Int'

Typ: Ganze Zahl 123

Name: Int
Menschenlesbare Beschreibung

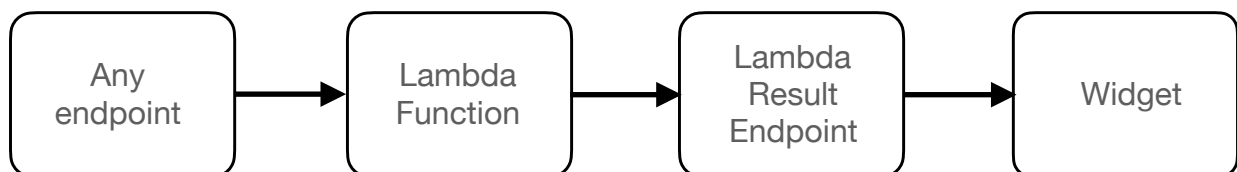
AUSGANG

Wert senden: 0xAA[u16]BB
OK

Lambda Functions (Advanced Features)

Lambdas are small functions written in Javascript that process data from endpoints and make the results available to the rest of the system.

Important: This is an expert function because it requires programming knowledge in JavaScript!



From the dashboard, the Lambda icon can be used to access the Lambda administration mode:

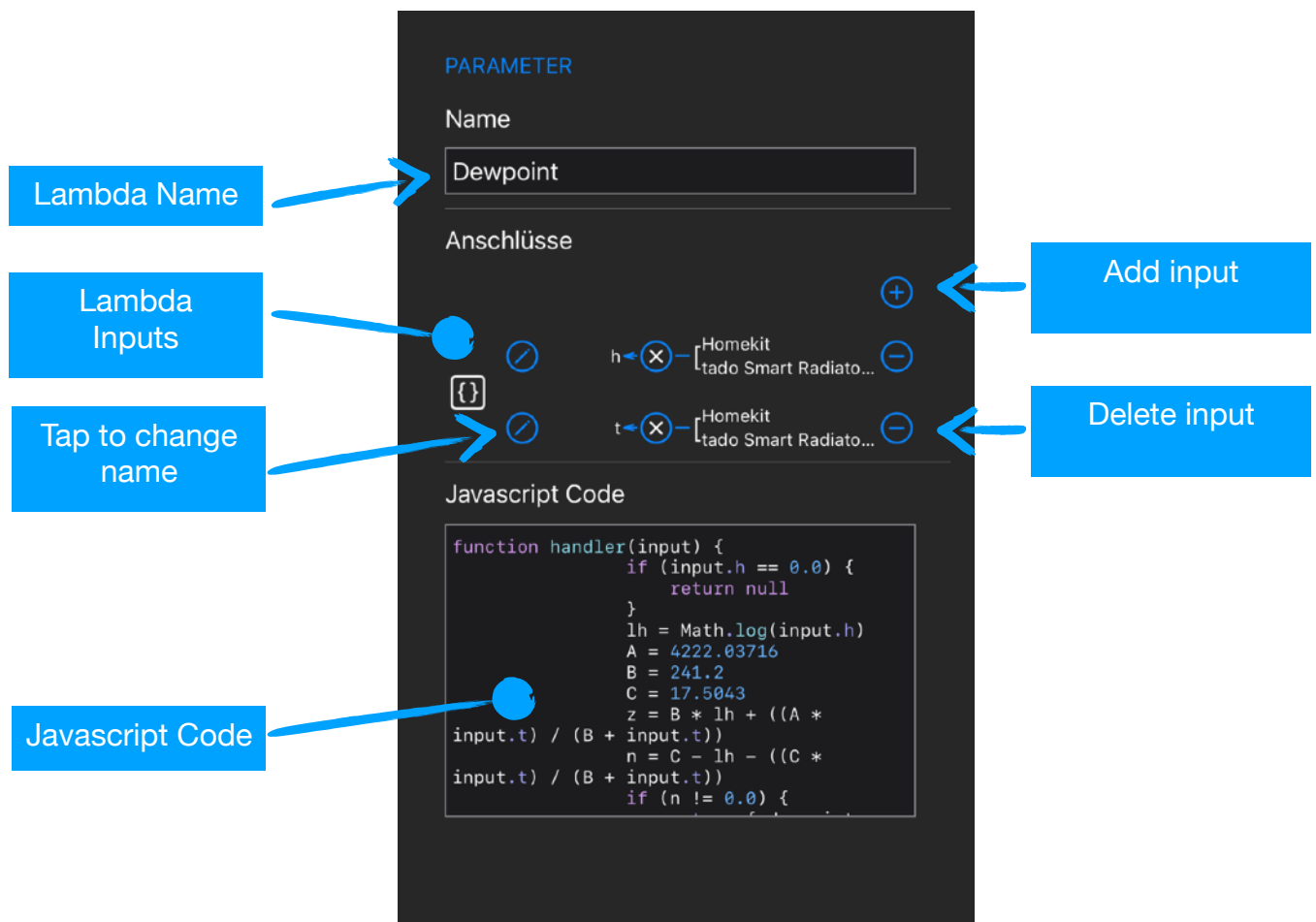
Schliessen Lambdas +

Add Lambda

Lambda Status

Lambda activate / deactivate

Status	Name	Value	Toggle
✓	Dewpoint OK		On
✓	Grundlast OK		On
✓	Intervall Ampel OK		On



By swiping from right to left, the "Delete" button appears.
Tapping the table row takes you to the lambda configuration.

Lambda Configuration

A lambda consists of exactly one Javascript function called "handler". This function receives a Javascript object as a transfer argument.

Data Input

A lambda function can have any number of input connections. To do this, simply add new connections with the "+" icon and connect to any endpoints (currently only the types "Integer", "Float", "Bool", "Index" and "Percent" are supported!)

All connection values are packed in a transfer object:

```
{
  name: value,
  name: value,
  allValues: [wert, wert, ...]
  ...
}
```

```
}
```

In addition to the individual values all values are also passed into a single array named "allValues" (Attention: The values in the array are sorted by their property names!). In our example it could look like this

```
{  
  h: 78.3,  
  t: 18.7,  
  allValues: [78.3, 18.7]  
}
```

The property names in the object correspond to the selected connection names!

The lambda function is always called when one of the input values has changed. If there are several input values, their last known value is transferred, therefore each input value must have been received at least once at the first start before the lambda function is called for the first time!

Data Output

The lambda function can have several return values. This is also packaged in a Javascript object with the following scheme:

```
{  
  name: { value: wert, unit: unit, min: min, max: max },  
  name: { value: wert, unit: unit, min: min, max: max },  
  ...  
}
```

where "unit", "min" und "max" are optional. In this example

```
{  
  dewpoint: { value: 13.4, unit: "°C", min: 0.0, max: 40.0 }  
}
```

Note: Since JavaScript does not have an integer type, you can force it with "forceInt: true", e.g.

```
{  
  intvalue: { value: 42, forceInt: true }  
}
```

Each value returned by the function is published via the special endpoint "Lambda Results" and can now be connected to a widget for display.



The return value "null" is accordingly ignored and no value is published.

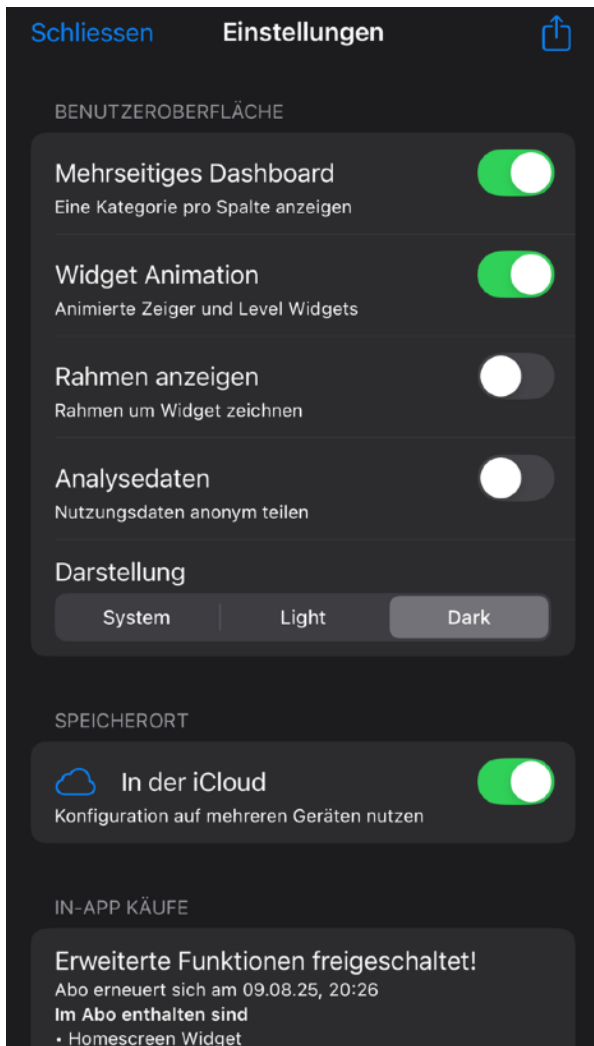
Note: The result of a lambda function can again be the input of another function (but not itself)!

The lambda used here as an example calculates the dew point from temperature and humidity using the Magnus formula:

```
function handler(input) {
  if (input.h == 0.0) {
    return null
  }
  lh = Math.log(input.h / 100.0)
  A = 4222.03716
  B = 241.2
  C = 17.5043
  z = B * lh + ((A * input.t) / (B + input.t))
  n = C - lh - ((C * input.t) / (B + input.t))
  if (n != 0.0) {
    return { dewpoint: { value: z/n, unit: "°C" } }
  }
  return null
}
```

Global Settings

The global attitude currently includes



- **Multi-Column Dashboard (In-App Feature)**

A separate column is displayed in the dashboard for each category

- **Widget Animation**

Animates value changes for the Gauge and Level widgets

- **Show Frame**

Draw frame around widgets

- **Analytics**

When you start the app for the first time, this setting is queried separately and can be changed here at any time. If the option is activated, the following information is anonymously shared with the app author:

- Type and number of widgets used
- Type and number of endpoints used

- **Appearance**

System: Appearance follows the system setting (Light/Dark)

Light: Always light appearance (white background)

Dark: Always dark appearance (black background)

- **Storage**

Select to store configuration in iCloud and can thus be kept in sync across multiple devices.

- **In-app purchases**

- In addition a monthly subscription for € 1.49 can be purchased to unlock advanced features. Previous buyers of the Ad-Free subscriptions automatically get these features unlocked, too!

The features mainly target users with Large SmartHome setups:

- Homescreen Widget (V1.7)
- Javascript Lambda Functions (V1.8)
- Multi column dashboard
- Search and filter datapoints
- Import scenes (from selected endpoints)
- Debug-Logging for all endpoints helps solving issues

Please support Visual with this in-app purchase for the cost of less than a cappuccino per month! Only then can Visual be further developed (new features) and maintained (new iOS versions). Thank you!

Note: The subscription can be ended at any time in the iCloud subscription settings!

iCloud Sync

The entire Visual configuration is stored locally in the app's sandbox and automatically synced to the user's iCloud account. That means any change in Visual will automatically be visible on all other iOS devices in Visual - as long as you're signed into the same iCloud account!

If you make changes on multiple devices at the same time, you get a conflict warning and you have to manually decide on a version.

Appendix A: Simplified JSON Path Syntax

JSON Path syntax can be used in the data point definitions of the *HTTPClient* and *MQTTClient* endpoints. **The JSONPath expression must be entered in angle bracket (<...>).**

Examples:

```
{
  "Name": "Mustermann",
  "Vorname": "Max"
}
```

<Name>	Mustermann
<Vorname>	Max

```
{
  "color": { "red": 1.0, "green": 0.2, "blue": 0.5 }
}
```

<color.red>	1.0
<color.blue>	0.5
<color.@keys>	red, green, blue

```
{
  "Fahrzeuge": [
    { "Typ": "PKW", "Räder": 4 },
    { "Typ": "Fahrrad", "Räder": 2 }
  ]
}
```

<Fahrzeuge.[0].Räder>	4
<Fahrzeuge.[1].Typ>	Fahrrad
<Fahrzeuge.*.Typ>	PKW, Fahrrad
<Fahrzeuge.@count>	2

In addition mathematical expressions can be used. To do so the field value has to start with "=" e.g.

```
{
  "color": { "red": 1.0, "green": 0.2, "blue": 0.5 }
}
```

=<color.blue>*100	50.0
=<color.blue>*<color.green>*100	10.0

Appendix B: Data Format For Diagramms

Type	JSON Format	JSON Path
[XY]	<pre>[{"x": x0, "y": y0}, {"x": x1, "y": y1}, ...]</pre>	<[*]>
	<pre>{ "data": [{"x": x0, "y": y0}, {"x": x1, "y": y1}, ...] }</pre>	<data.[*]>
[1.2]	<pre>[y0, y1, ...]</pre>	<[*]>
	<pre>{ "data": [y0, y1, ...] }</pre>	<data.[*]>

[XY] The "x" values can be a simple index or a data/time value in the Unixtime format (seconds since 01.01.1970)

[1.2] The "x" values are automatically created (0...n)

Contact

me@andreas-binner.de