vDC API properties

Contents

API Version History

This document covers vDC API Version 2

2 cleaned up, added multiple vdcs per vdc host, added MOC support
1 initial version of vDC API (shown as “1.0” in JSON API)

Basics

- This document is based on the "vDC API" specification. Please refer to the corresponding document for general description of the API and the available messages/ calls.
- This document specifies the named properties available for different types of *addressable entities* (vDC, vdSD, vDC host) as well as the properties common for all types of *addressable entities.*

Common properties for all addressable entities

- The common properties must be supported by entities which can be addressed via a dSUID using this API (*addressable entities*). At the time of writing, this includes virtual devices, logical vDCs and the vDC host, but may be extended to other entities.
- The "type" property reveals the kind of entity.
- Properties common to a specific entity type are maintained in separate documents. In particular, see "vdSD properties" document for common properties of virtual devices.

property name	acc	Type/range	description
dSUID	r	string	the dSUID of the entity. Normally not used in regular vDC API calls, as these require the dSUID in the getProperty() call. Useful for debugging.
idBlockSize	r	integer	The number of consecutive dSUIDs (starting with this device's dSUID) that are reserved. This means that the next *idBlockSize*-1 dSUIDs are guaranteed not being used by any vdSD in any vDC. This mechanism allows that a vdSM can implement representing multi-input devices as multiple dSUIDs for backwards compatibility with dS 1.0
type	r	string	Type of entity: "vdSD" : virtual dS device "vDC" : a logical vDC "vDChost" : the vDC host "vdSM" : a vdSM

property name	acc	Type/range	description
model	r	string	human-readable model string of the entity. Should be descriptive enough to allow a human to associate it with a piece of hardware or software
hardwareVersion	r	string or NULL	string describing the hardware device's version represented by this entity, if available
hardwareGuid	r	string or NULL	hardware's native globally unique identifier (GUID), if any, in URN-like format: *formatname*:*actualID* The following formats are in use: • enoceanaddress:XXXXXXXX = 8 hex digits enOcean device address • *gs1:(01)ggggg(21)sssss* = GS1 formatted GTIN plus Serial number • *uuid:UUUUUUU* = UUID • *macaddress:MMMMM* = MAC Address
oemGuid	r	string or NULL	Globaly unique identifier (GUID) of the product the hardware is embedded in, if any - see hardwareGuid for format variants.
name	r/w	string	user-specified name of the entity. Is also stored upstreams in the vdSM and further up, but is useful for the vDC to know for configuration and debugging. The vDC usually generates descriptive default names for newly connected devices. When the vdSM registers a device, it should read this property and propagate the name towards the dSS. When the user changes the name via the dSS configurator, this property should be updated with the new name.

Virtual digitalSTROM device (vdSD) properties

- The following table applies to entities which have a value of "vdSD" for the "type" property.
- All vdSDs must also support the basic set of properties as described under "Common properties" above.

Properties related to the entire device

property name	acc	Type/range	description
primaryGroup	r	integer, dS group number 1..8	basic group (color) of the device
outputIsMemberOfGroup [groupNo]	r/w	boolean	array of boolean flags, array index represents the dS group number the output(s) of this device is/are member of.
outputLocalPriority	r/w	boolean	enables local priority of the device's output(s). In local priority, device ignores scene calls unless the scene has the *ignoreLocalPriority* flag set, or the *callScene* call has the *force* parameter set to true
zoneID	r/w	integer, global dS Zone ID	this should be updated by the vdSM to reflect the zone the device is in. The vDC may use this value to optimize zone calls (i.e. bundle calls to actual hardware if single device calls are slow)
progMode	r/w	boolean	enables local programming mode (for those devices that have it)
numDevicesInHW	r	optional integer	Number of separate vdSDs that represent the same hardware device (which MUST have the same *hardwareGUID*) Present only if a unambiguous statement can be made about the number of devices per hardware. Devices that are *usually* grouped in one chassis, but can be taken apart should not report this property.
deviceIndexInHW	r	optional integer	only if *numDevicesInHW* exists, this enumerates the devices within the same hardware

Button Inputs

property name	acc	Type/range	description
buttonInputDescriptions[]	r	object	array of object, representing capabilities of button inputs
name	r	string	human readable name/number for the input (e.g. matching labels for hardware connectors)
supportsLocalKeyMode	r	boolean	can be local button
buttonID	r	integer 0..n (optional)	ID of physical button. No ID means no fixed assignment to a button. All elements of a multi-function hardware button must have the same buttonID.
buttonType	r	integer enum (inputs with buttons supported only)	Type of physical button 0: undefined 1: single pushbutton 2: 2-way pushbutton 3: 4-way navigation button 4: 4-way navigation with center button 5: 8-way navigation with center button 6: on-off switch
buttonElementID	r	integer (inputs with buttons supported only)	Element of multi-contact button: 0: center 1: down 2: up 3: left 4: right 5: upper left 6: lower left 7: upper right 8: lower right Note: For undefined *buttonType*, *buttonElement* just enumerates the elements (0..numElements-1)
buttonInputSettings[]	r/w	object	array of objects, representing configuration settings of buttons and binary inputs
group	r/w	integer	dS group number 1..8
function	r/w	integer 0..15	see LTNUM descriptions (0: device, 5: room, ...)
mode	r/w	integer	255: inactive 0: standard 2: presence 5..8 : button1..4 down 9..12 : button1..4 up
channel	r/w	integer enum	channel this button should control 0: (default) button controls the default channel 1..191: reserved for digitalSTROM standard channel types 192..239: device specific channel types

property name	acc	Type/range	description
setsLocalPriority	r/w	boolean	button should set local priority
callsPresent	r/w	boolean	button should call present (if system state is absent)
buttonInputStates[]	r	object	representation of the current state of the button
value	r	boolean or NULL	false=inactive, true=active, NULL=unknown state
clickType	r	integer enum	Most recent click state of the button: 0: tip_1x 1: tip_2x 2: tip_3x 3: tip_4x 4: hold_start 5: hold_repeat 6: hold_end 7: click_1x 8: click_2x 9: click_3x 10: short_long 11: local_off 12: local_on 13: short_short_long 14: local_stop 255: idle (no recent click)
age	r	double or NULL	age of the state shown in the *value* and *clickType* fields in seconds. If no recent state is known, returns NULL.
error	r	integer enum	0: ok 1: open circuit 2: short circuit 4: bus connection problem 5: low battery in device 6: other device error

Binary Inputs

property name	acc	Type/range	description
binaryInputDescriptions[]	r	object	array of object, representing capabilities of binary inputs
name	r	string	human readable name/number for the input (e.g. matching labels for hardware connectors)
inputType	r	integer (inputs with binary functions supported only)	0: poll only 1: detects changes
inputUsage	r	integer enum	Describes the usage field for the input (beyond device color) 0: undefined (generic usage or unknown) 1: room climate 2: outdoor climate 3: climate setting (from user)
hardwareSensorFunction	r	integer enum	hardwired function of this input if it is not freely configurable. See sensorFunction in binaryInputSettings[] below for possible values. 0 means generic input with no hardware-defined functionality.
updateInterval	r	double	how fast the physical value is tracked, in seconds
binaryInputSettings[]	r/w	object	array of objects, representing configuration settings of buttons and binary inputs
group	r/w	integer	dS group number 1..8
sensorFunction	r/w	integer enum	0x00 App Mode (no system function) 0x01 Presence (Präsenz) 0x02 Light (Helligkeit) – aktuell noch nicht in Verwendung 0x03 Presence in darkness (Präsenz bei Dunkelheit) – aktuell noch nicht in Verwendung 0x04 Twilight (Dämmerung) 0x05 Motion detector (Bewegung) 0x06 Motion in darkness (Bewegung bei Dunkelheit) – aktuell noch nicht in Verwendung 0x07 Smoke detector (Rauchmelder) 0x08 Wind monitor (Windwächter) 0x09 Rain monitor (Regenwächter) 0x0a Solar radiation (Sonneneinstrahlung) 0x0b Thermostat (Thermostat)
binaryInputStates[]	r	object	representation of the current state of the inputs
value	r	boolean or NULL	false=inactive, true=active, NULL=undefined

property name	acc	Type/range	description
age	r	double or NULL	age of the state shown in the *value* field in seconds. If no recent state is known, returns NULL
error	r	integer enum	0: ok 1: open circuit 2: short circuit 4: bus connection problem 5: low battery in device 6: other device error

Sensors

property name	acc	Type/range	description
sensorDescriptions[]	r	object	description of sensor capabilities
name	r	string	human readable name/number for the sensor
sensorType	r	integer enum	Describes the type of physical unit the sensor measures 0 : none 1 : temperature in °C 2 : relative humitity in % 3 : illumination in lux 4 : supply voltage level in V 5 : CO concentration in ppm 6 : Radon activity in Bq/m3 7 : gas type sensor 8 : particles <10μm in µg/m3 9 : particles <2.5μm in µg/m3 10 : particles <1μm in µg/m3 11 : room operating panel set point, 0..1 12 : fan speed, 0..1 (0=off, <0=auto) 13 : wind speed in m/s
sensorUsage	r	integer enum	Describes the usage field for the sensor 0: undefined (generic usage or unknown) 1: room 2: outdoor 3: user (setting, dial)
min	r	double	min value
max	r	double	max value
resolution	r	double	resolution (size of LSB of actual HW sensor)
updateInterval	r	double	how fast the physical value is tracked, in seconds
sensorSettings[]	r/w	object	sensor configuration
group	r/w	integer	dS group number 1..8
minPushInterval	r/w	double	minimum interval between pushes of changed state in seconds, default=2
changesOnlyInterval	r/w	double	minimum interval between pushes with same value (in case sensor hardware sends update, but with same value as before - only age will differ). default=0=all updates from hardware trigger a push
sensorStates[]	r	object	sensor states
value	r	double or NULL	current sensor value in the unit specified in SensorCapabilities.unit If no recent state is known, returns NULL.

property name	acc	Type/range	description
age	r	double or NULL	age of the state shown in the *value* field in seconds. If no recent state is known, returns NULL
error	r	integer enum	0: ok 1: open circuit 2: short circuit 4: bus connection problem 5: low battery in device 6: other device error

Outputs

There are two different ways to look at outputs:

- by *index* - each device can have N outputs, which are numbered 0..N-1. To access outputs by index (especially useful to enumerate all available outputs of a device), use the *outputDescriptions[], outputSettings[]* and *outputStates[]* property arrays. The index for these arrays is the output number (0..N-1)
- by *channel* type - each device output is usually assigned a *channel* type describing the functional purpose of the output (such as brightness, hue, saturation for color lights). The channel type is a number between 1..239. This number can be used as index into the *channelDescriptions[], channelSettings[]* and *channelStates[]* property arrays to access outputs by channel.

Notes:

- *channelDescriptions[], channelSettings[]* and *channelStates[]* are not separate properties, but just an alternate access method to simplify access by channel.
- The *channelDescriptions[], channelSettings[]* and *channelStates[]* arrays will not always include all outputs of the device. For example, for devices with generic outputs which have a configurable channel type (see *outputSettings[].channel*), unconfigured outputs will not be accessible by channel.
- If the device does not have a channel of the specified type, *channelDescriptions[channel], channelSettings[channel]* and *channelStates[channel]* will return NULL when reading, and will perform no operation when written.
- channel 0 is the default channel, and always accesses the first (main) output of the device (e.g. brightness for lights). The same output may be accessible a second time under its actual channel number (e.g. as channel 1 = brightness for lights).

property name	acc	Type/range	description
outputDescriptions[]	r	object	array of objects, representing hardware capabilities of output. In MOC (multiple output channel) devices, output index 0 must represent the default channel of the device (e.g: brightness for lights), i.e the channel that is also addressed by *channelDescription[0]*.
name	r	string	human readable name/number for the output (e.g. matching labels for hardware connectors)
function	r	integer enum	0: on/off only 1: dimmer 2: positional

property name	acc	Type/range	description
channel	r	integer enum	Channel type as given by this particular device's hardware. For devices with generic channels, which can be used for different channel types, this value can be 0. 0: generic output, no hardware-defined type 1..191: reserved for digitalSTROM standard channel types 192..239: device specific channel types Note: the value shown here may not be the actual channel type in use as devices may allow configuring the channel type in *outputSettings[].channel.*
outputUsage	r	integer enum	Describes the usage field for the output (beyond device color) 0: undefined (generic usage or unknown) 1: room 2: outdoors 3: user (display/indicator)
variableRamp	r	boolean	supports variable ramps
maxPower	r	integer	max output power in Watts. If absent, power capability is undefined
minDim	r	integer	minmum brightness that hardware supports (for dimming outputs)
outputSettings[]	r/w	object	array of objects, representing operation mode of output
group	r/w	integer	dS group number 1..8
mode	r/w	integer enum	0: disabled, inactive 1: binary 2: gradual
channel	r/ (w)	integer enum	For generic outputs (having *outputDescription[].channel*==0), the output can be assigned a channel type here. For outputs tied to a specific channel type by hardware design, this setting cannot be changed and returns the same value as *outputDescription[].channel* does.
pushChanges	r/w	boolean	if set, locally generated changes in the output value will be pushed
dimTimeUp	r/w	integer	dim up time in ms
dimTimeDown	r/w	integer	dim down time in ms
dimTimeUpAlt1	r/w	integer	alternate 1 dim up time in ms
dimTimeDownAlt1	r/w	integer	alternate 1 dim down time in ms
dimTimeUpAlt2	r/w	integer	alternate 2 dim up time in ms

property name	acc	Type/range	description
dimTimeDownAlt2	r/w	integer	alternate 2 dim down time in ms
outputStates[]	r/w	object	array of output states
value	r/w	integer	current output value (brightness, blind position, on/off)
age	r	double	age of the state shown in the *value* field in seconds. This indicates when the value was last **applied** to the actual device hardware, or when an actual output status was last received from the device. *age* is NULL when a new value was set, but not yet applied to the device
error	r	integer enum	0: ok 1: open circuit / lamp broken 2: short circuit 3: overload 4: bus connection problem 5: low battery in device 6: other device error
channelDescriptions [channel]	r/w	object or NULL	This property provides a way to access the same information as found in *outputDescriptions[],* but not by index, but by digitalSTROM channel type. See *outputDescriptions[]* for the descriptions of the fields contained.
channelSettings[channel]	r/w	object or NULL	This property provides a way to access the same information as found in *outputSettings[],* but not by index, but by digitalSTROM channel type. See *outputSettings[]* for the descriptions of the fields contained.
channelStates[channel]	r/w	object or NULL	This property provides a way to access the same information as found in *outputStates[],* but not by index, but by digitalSTROM channel type. See *outputStates[]* for the descriptions of the fields contained.

Scenes

The scenes table stores a set of values to apply to the outputs of the device when a particular scene is called.
As outputs can be looked at in two different ways, by index or by channel (see paragraph “Outputs” above), there are also two ways of looking at scene table entries:

- to access the set of values within a scene table entry by output index, use the *outputScenes[]* array. In this case, the N in *valueN* and *dontCareN* is the output index (0..number of outputs-1)
- to access the set of values within a scene table entry by channel type, use the *channelScenes[]* array. In this case, the N in *valueN* and *dontCareN* is the channel type (0..239, where 0=default channel and 1..239=specific channel).

property name	acc	Type/range	description
outputScenes[]	r/w	object	array of saved device states that can be recalled via callScene. Index is scene number
valueN	r/w	optional integer	output value N, where: • N is the output index (0..number of outputs-1) when scenes are accessed via outputScenes[]. • N is the channel type (0..239) when scenes are accessed via channelScenes[].
dontCareN	r/w	boolean	if set, calling this scene does not apply the stored output *valueN*
alerting	r/w	boolean	calling this scene performs an alerting action usinge the device’s output(s): • For light devices, alert means blinking the light • For other device types, the alert is something that makes sense in the context of that device. • The stored scene values are always used to parametrize the alerting action if applicable (e.g. color for multi-color lights), regardless of the dontCare flags. • When the alerting action has finished, outputs with dontCare set will revert to the value present before the alert, while outputs with dontCare *not* set will keep the values as stored in the scene.
ignoreLocalPriority	r/w	boolean	calling this scene overrides local priority
dimTimeSelector	r/w	integer 0..2	selects the dimming time: • 0 = use *dimTimeUp/dimTimeDown* • 1 = use *dimTimeUpAlt1/dimTimeDownAlt1* • 2 = use *dimTimeUpAlt2/dimTimeDownAlt2*
x-*other_scene_value*			scenes might contain additional device specific scene values not currently used by the dS system

Control Values

Control Values are not regular properties, but like properties, control values are named values and thus *similar* to properties. Unlike properties, control values cannot be read but only written to a vdSD, using the *setControlValue* call.

control value name	Type	description
heatingLevel	double	(dS Sensortype 50): level of heating intensity -100..100: 0=no heating or cooling, 100=max heating, -100 max cooling

digitalSTROM 1.0 mapping compatibility

An important design goal for the vDC API and the property set was to avoid carrying over dS 1.0 specific limitations.
On the other hand, the vDC API was designed to support capabilities current dSS 1.x architecture can't support yet, but are likely to be implemented in future dS versions.

Still, the vDC + vdSM needs to be compatible with existing dSS 1.x installations.

To achieve this, vDC devices (vdSDs) that provide functionality similar or equal to existing hardware digitalSTROM devices (dSDs), must have **sensible default settings that make them mappable into existing dSS 1.x installations**.

This chapter lists the conventions that must be followed for certain device types to make them mappable into dSS 1.x environments.

2-way buttons

2-way buttons (rockers) like present in many enOcean devices must conform to the following default behaviour:

1. the vdSD must have two button inputs (represented by 2 array elements in the *buttonInputDescriptions/Settings/States* property arrays)
2. the buttonInput with index = 0 must represent the "down" button
3. the buttonInput with index = 1 must represent the "up" button
4. buttonInputSettings[0].mode must be 6 (down button paired with second input)
5. buttonInputSettings[1].mode must be 9 (up button paired with first input)
6. in the dSUID space, the dSUID following the dSUID of the device (device's dSUID + 1) must be guaranteed unused. This means that the *idBlockSize* property must be 2 to document that the next dSUID is guaranteed unused. This *allows* the vdSM to virtually split the device into two separate dSUIDs to mimic for example a SW-TKM210 towards the dSS 1.x environment. The vdSM *may* also choose to represent the device as a single dSUID with a inseparable 2-way button instead, like a GR-TKM210.

Multiple vdSDs in a single hardware device

Some hardware devices contain more than one instance of a certain functional unit. Usually, these are represented as a separate vdSD each, to allow maximum flexibility in the way the functional units can be used.
For example, a dual 2-way button enOcean device will be represented as 2 entirely separate vdSDs, because despite the physical proximity, each button might conrol a different zone, group or function. By default, such a device will be represented as 2 separate SW-TKM210 (dual input) devices. However, the vdSM might want to represent it as a single SW-TKM200 (quad input) device. To allow the vdSM to find out which and how many vdSDs are in the same hardware device, the vdSD *should* expose this information in the *numDevicesInHW* and *deviceIndexInHW* properties as follows:

1. *numDevicesInHW* contains the number of vdSDs in the same hardware device
2. *hardwareGUID* identifies the hardware device of which the vdSD is part of

3. *deviceIndexInHW* contains an index, 0..*numDevicesInHW-1* that enumerates the vdSDs in the same hardware device
4. This association of vdSDs to a containing hardware device must only be made when the number of grouped vdSDs and enumeration is unambiguous and permanent. So just 3 modules that usually ship mounted on a common frame, but can be easily separated and used independently should not have *numDevicesInHW* and *deviceIndexInHW* properties.