vDC API

Contents

API Version History

This document covers vDC API Version 2

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

Basics

- based on Google protocol buffers, see https://developers.google.com/protocol-buffers
- transport level is a TCP socket connection, established by the vdSM to the vDC.
- The TCP stream consists of a 2-byte header containing the message length (16 bits, in network byte order, maximum accepted length is 16384 bytes) followed by the protocol buffer message.
- The life time of the connection defines the vDC session. If the connection breaks, a new session needs to be established.
- Design goal for the API was to reduce the number of methods as much as possible (compared to the vdSM preliminiary device API). Only the most basic actions common to all dS devices (*callScene, saveScene, ping, pong,* ...) are implemented as separate methods, everything else should be handled via reading and writing named properties (eg. functionality like former *ProgModeOn, SetOutVal…*).

Terms

(logical) vDC	virtual device connector. A vDC is primarily a logical entity within the dS system and has its own dSUID. A vDC represents a type or class of external devices.
vDC host	network device offering a server socket for vdsm to connect to. One vDC host can host multiple logical vDCs, if the host supports multiple device classes.
vdSM	virtual digitalSTROM Meter. A vdSM can connect to one or several vDC hosts to connect one or several logical vDCs to the dS system.
vdSD	virtual digtalSTROM device. A vdSD represents a single device in the dS system, and behaves like a real digitalSTROM terminal block (dSD, digitalSTROM device)
vDC session	logical connection between a vdSM and a vDC host (representing one or multiple vDCs)

Discovery

Requirements for vDC hosts

- vDC hosts needs to be able to be discovered by vdSMs (embedded within a dS installation) automatically on a given network.
- It must be avoided that a vdSM connects to a wrong (neighbour's, for example) vDC host
- It must be avoided requiring vDCs to have UI of their own.

Requirements for vdSMs

The same requirements exist for vdSMs - these need to be discoverable by ds485p such that they can be automatically connected.

Short term solution

- vDC hosts and vdSMs announce their services using Avahi (gnu implementation of Apple's Bonjour)
- vdSMs look for available vDC hosts in the Avahi announcements and connect to at least one of them. vDC hosts might reject connections if they are already connected to another vdSM.
- ds485p looks for available vdSMs in the Avahi announcements and connect them.
- vDC hosts that come bundled with a vdSM in the same physical device (for example the plan44.ch DALI/EnOcean bridge) may choose to not announce the vDC host when the in-device vdSM is configured to always connect the in-device vDC host.
- likewise, vdSMs that come bundled with a dss/dsa/ds485p might be directly tied to their in-device dss/dsa/ds485p without announcing it into the LAN.
- To avoid wrong connections, the instance that initiates a connection must be able to check possible peers received via Avahi against an optional whitelist. If the whitelist exists, only listed peers might be connected. The idea is that in small/simple installations connection is fully automatic, but if conflicts arise in large setups (like digitalSTROM dev space) these can be solved by adding whitelists.

Long term solution idea

The whitelist might get replaced or supplemented by more end-user friendly mechanism, possibly similar to how dss API clients are approved now (client requests and receives a token, for which the user can grant access in the web configurator)

Avahi service description files

- On a system with avahi-daemon installed, announcing services consists of creating .service files and putting them into /etc/avahi/services

- The service types are chosen to be very unlikely to collide with other company's services by using the "ds-" prefix. If needed, dS service names could still be registered with IANA later (see http://www.rfc-editor.org/rfc/rfc6335.txt)
- The port numbers used below are just examples, actual ports might differ.
- The advertisment for vdSMs must contain a txt record specifying the vdsm's dSUID
- Once dS services can handle ipv6, the "protocol" attribute should be set to "any"

/etc/avahi/services/ds-vdc.service

```
<?xml version="1.0" standalone='no'?>
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
  <name replace-wildcards="yes">digitalSTROM vDC host on %h</name>
  <service protocol="ipv4">
    <type>_ds-vdc._tcp</type>
    <port>8444</port>
  </service>
</service-group>
```

/etc/avahi/services/ds-vdsm.service

```
<?xml version="1.0" standalone='no'?>
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
  <name replace-wildcards="yes">digitalSTROM vdSM on %h</name>
 <service protocol="ipv4">
    <type>_ds-vdsm._tcp</type>
    <port>8441</port>
    <txt-record>dSUID=198C033E330755E78015F97AD093DD1C00</txt-record>
  </service>
</service-group>
```

vDC host Session

Basics

- a session represents the connection from a single vdSM to a single vDC host (which may host one or multiple logical vDCs)
- a session is identical with having a TCP connection.
- a vdSM aims to keep the vDC sessions active all the time.
- if a session is terminated for any reason, the vdSM must try to establish a new session.
- Only if fundamental incompatibility between vDC host and vdSM is detected (no common API version), the vdSM might cease trying to establish a session..
- at a given time, a vdSM might have at most one single session with one particular vDC host (it may of course have connections to multiple distinct vDC hosts)
- a vDC session must always start with a session initialisation phase, before it changes into the operation phase.

vDC host Session initialisation

- vdSM connects to the vDC host
- vdSM calls "Hello" method on vDC host.

Method	**hello**	vdSM -> vDC host
Request Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	dSUID of the vdSM
api_version	integer	vDC API version The API version as described in this document is 2
Response		
dSUID	17 binary bytes representing dSUID	dSUID of the vDC host Note: although the vDC host does not yet appear as a logical entity in the current digitalSTROM specification, it still has a dSUID in order to be addressable by custom apps and for future dS system evolution
Error case: GenericResponse		
code	integer	RESULT_SERVICE_NOT_AVAILABLE: This means the vDC host cannot accept the connection because it is already connected to *another* vdSM. RESULT_INCOMPATIBLE_API: The vdSM does not support the API version presented in *APIVersion*
message	string	explanation text

vDC host Session operation

- session operation consists of announcing one or multiple logical vDCs (see below) and then announcing none, one or multiple vdSDs (see below)
- To avoid a vDC host session to implicitly end, some minimal communication must occur between vdSM and vDC host in regular intervals (for example: *ping/pong*)

vDC host Session termination

- a vDC session is explicitly terminated when the *Bye* command is called:

Method	**bye**	vdSM -> vDC host
Request Parameter	Type	Description
Response: GenericResponse		
code	integer	0 - success

- a vDC session is implicitly terminated when a *Hello* command asks for starting a new vDC session
- Closing the connection implicitly terminates the vDC session as well

vDC Announcement

- after vDC session is established, the vDC host must announce every logical vDC it hosts, before it announces any of that logical vDC's devices.
- It does so by calling the "announcevdc" method
- unlike individual devices (see below), logical vDCs cannot vanish during an established vDC session.

vDC Method	**announcevdc**	vDC host -> vdSM
Request Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the vDC to be announced
Response: GenericResponse		
code	integer	0 - success
Error case: GenericResponse		
code	integer	RESULT_INSUFFICIENT_STORAGE - vdSM cannot handle another vDC
message	string	explanation text

vdSD Announcement

- for every vDC announced, the vDC must announce all device managed by the vDC.
- A device is considered managed by the vDC when the vDC has resonably reliable information that the device is in fact connected to or connectable from the vDC. This means that the vDC should announce devices even if the device is temporarily offline.
- The vdSM can explicitly request removal of managed devices

Device Announcement

- vDC calls "announcedevice" method on vdSM

vDC Method	**announcedevice**	vDC host -> vdSM
Request Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the device to be announced
vdc_dSUID	17 binary bytes representing dSUID	The dSUID of the logical vdc which contains this device
Response: GenericResponse		
code	integer	0 - success
Error case: GenericResponse		
code	integer	RESULT_INSUFFICIENT_STORAGE - vdSM cannot handle another device
message	string	explanation text

Device Operation

- after announcing a device, device level methods can be invoked by either party on the other, and device level notifications can be sent by either party to the other.
- See separate chapter on Device Level Method Notifications below for specification of individual methods/notifications supported

Ending Device Operation

- either vDC sends "Vanish" notification to vdSM to indicate a device has been physically disconnected or unlearned (think: enOcean unidirectional switches for example) from the vDC.

Notification	**vanish**	vDC host -> vdSM
Notification Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the device that has vanished

- or vdSM calls "remove" method on vDC to request a device to be removed from this vDC (but might have been connected to another vDC in the meantime). vDC may reject removal only if it has 100% knowledge the device is actually connected and operable (in which case the higher levels in dS should see the device as active anyway and will not allow users to delete it)

vDC Method	**remove**	vdSM -> vDC host
Request Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the device to be removed
Response: GenericResponse		
code	integer	0 - success
Error case: GenericResponse		
code	integer	RESULT_FORBIDDEN - vDC does not allow to remove the device, for example because it is verifiably physically connected to the vDC. However, consider that for example wireless devices might get carried away without being un-paired from their former vDC, and then paired with another vDC in the same installation. In such a case, dss/dsa will want to remove the device from the former vDC/vdSM - the vDC must not reject such a deletion attempt.
message	string	explanation text

Device and vDC Operation Methods and Notifications

- Note: the vDC host and every logical vDC have dSUIDs and can be addressed by some (not all) of the device level methods, in particular reading and writing named properties for vDC configuration and the presence polling method *ping*.

Named Property access

- *Adressable entities* are items within the vdc host that have their own dSUID. The dSUID can be specified in vDC API request to specifically address the entity. The vDC host as a whole, the contained vDC(s) and every virtual device has its own dSUID and thus is a addressable entity.
- Adressable entities have named properties which can be read and written by the vdSM and are in some cases being pushed from the vDC.
- Properties that are defined in the digitalSTROM specifications (including this document) with name, type and behaviour are considered *system properties*. Implementations conforming to the specification must support these.
- Additionally, implementations might add *implementation specific properties* to extend functionality beyond what the dS system demands. These properties' names must always be prefixed by "x-". It is further recommended to include an identifier for the party who introduces a property. For example company Abc Inc. could prefix their properties with "x-abc-".
- Supported value types for properties are the simple types integer, double, boolean, string and binary bytes, or a list of property elements which in turn contain a name (key) and either a simple type (value), or yet another level of property elements. Note that while properties can be nested indefinitely this way, it is **explicitly recommended to keep nesting levels as low as possible**.
- The available properties depend on the kind of the addressable entity (vdc, vdc host, virtual device) - the complete set of properties supported by a virtual device entity is defined in the *device profile* for that type of device*.*
- A common set of properties called *common properties* must be supported by all *adressable entities*. These properties can be read to identify the type of entity, and get some basic information for this.

Reading Property values

Virtual devices can have zero to several buttons, binary (digital) inputs and sensors. The following container properties provide access to the set of properties related to each input. The individual subproperties are described in separate paragraphs further down.

Method name	**getProperty**	vdSM -> vDC host
Request Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the entity (device, vDC or vDC host) to read properties from
query	property elements submessages	A property tree structure consisting of property elements, but with no values, specifying the property or properties to be returned. • The name of each property element specifies a property on that level to access. • If the name is specified empty, this is a wildcard meaning all elements from that level (for example: all inputs or all scenes) should be returned. • If the empty name wildcard is the last element specified, this means all deeper levels should be returned as well.
Response: *ResponseGetProperty*		
properties	property elements submessages	A property tree structure consisting of property elements and values representing the properties selected by the *query* structure. • when specific properties were requested with *query*, the result has the same structure as *query*, but with values filled in from actual properties. • properties that were requested and do not exist are returned with value set to NULL (no value)
Error case: *GenericResponse*		
code	integer	RESULT_FORBIDDEN - the property exists but cannot be read (write-only, uncommon case). RESULT_NOT_FOUND - the receiver (device or vDC itself) specified through dSUID is unknown at the callee.
message	string	explanation text

Writing property values

Method name	**setProperty**	vdSM -> vDC host
Request Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the entity (device, vDC or vDC host) to write properties to
properties	property elements submessages	A property tree structure consisting of property elements and values representing the properties to be written. • The name of each property element specifies a property on that level to access. • If the name is specified empty, this is a wildcard meaning all elements of that level (for example: all inputs or all scenes) should be set to the same value.
preload	optional boolean	Setting this flag to true indicates to the addressed entity (for example a color light device) not to apply the *properties* immediately, but buffer them until the next setProperty call is issued without *preload* set, and only apply all the buffered *values* together at that point in time. • The preload flag is effective only for selected properties (such as color components, brightness, color temperature of a color light) where applying values in the same instant is important. • It is not considered an error to set the preload flag for properties that may not support preload. • Writing multiple properties with the same setProperty call has the same effect as calling setProperty several times with *preload* set and finally once with preload not set.
Response: *GenericResponse*		
code	integer	0 = success
Error case: *GenericResponse*		
code	integer	RESULT_INVALID_VALUE_TYPE - passed type is wrong and cannot be written. RESULT_FORBIDDEN - the property cannot be written (is read-only or does not exist). RESULT_NOT_FOUND - the receiver (device or vDC itself) specified through dSUID is unknown at the callee.

Getting notified of a property value (change)

- Some properties (especially button/input/sensor states) might change within the device and can be reported to the dS system via pushProperty, avoiding the need for the vdSM to poll values.

Notification name	**pushProperty**	vDC host -> vdSM
Notification Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the entity (device, vDC or vDC host) from where this notification originates
properties	property elements submessages	A property tree structure consisting of property elements and values representing the information pushed to the vdSM.

Presence polling

- The presence polling is available for every addressable entity (vDC host, vDCs and all devices). Implementation should return a pong only if the entity can be considered active in the system. If possible at reasonable cost, a connection test with the device's hardware should be made. In some cases (unidirectional sensors) it might not be possible to query the device, in these cases the vDC should apply reasonable heuristics to decide whether to report the device as active or not.

Notification name	**ping**	vdSM -> vDC host
Notification Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the entity (device, vDC or vDC host) to send the ping to

Notification name	**pong**	vDC host -> vdSM
Notification Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the entity (device, vDC or vDC host) from where this pong originates

Actions

Actions are operations that may change the internal state of the device and/or its outputs, often depending on preconditions, but do not cause a distinct change of a single status value that could be read back.
Distinct, unconditional state changes that can be read back are always implemented as properties, not actions.

Call Scene

- calls a scene on the device

Notification name	**callScene**	vdSM -> vDC host
Notification Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the device
scene	integer	dS scene number to call Note that only regular scenes are supported. All dimming related scenes (10..15, 42..49, 52..55) are not supported. Use the dimChannel method instead (see below)
force	boolean	if true, local priority is overridden
group	optional integer	dS group number, present if the scene call was not applied to a single device, but a zone/group (informational only, vdSM already creates separate calls for every involved device)
zoneID	optional integer	dS global zone ID number, present if the scene call was not applied to a single device, but a zone/group (informational only, vdSM already creates separate calls for every involved device)

Save Scene

- save the relevant parts of the current state of the device (usually the output value, but possibly multiple output values, flags, etc. in future devices) as scene.

Notification name	**saveScene**	vdSM -> vDC host
Notification Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the device
scene	integer	dS scene number to save state into
group	optional integer	dS group number, present if the scene call was not applied to a single device, but a zone/group (informational only, vdSM already creates separate calls for every involved device)
zoneID	optional integer	dS global zone ID number, present if the scene call was not applied to a single device, but a zone/group (informational only, vdSM already creates separate calls for every involved device)

Undo Scene

- Undoes a scene call. All output values are restored to the state they had before the scene call.

Notification name	**undoScene**	vdSM -> vDC host
Notification Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the device
scene	integer	This specifies the scene call to undo. Undo is executed only if device's last called scene matches *scene*. This is to prevent undoing a scene which might have been called in the meantime from another origin (like a local on or off).
group	optional integer	dS group number, present if the scene call was not applied to a single device, but a zone/group (informational only, vdSM already creates separate calls for every involved device)
zoneID	optional integer	dS global zone ID number, present if the scene call was not applied to a single device, but a zone/group (informational only, vdSM already creates separate calls for every involved device)

Set local priority

- Sets the device into local priority mode (i.e. sets the *localPriority* property) if the passed scene does not have the *dontCare* flag set. This is used for including devices into area operations. Note that this is a compatibility method to simplify dS 1.0 interfacing and might be removed later in dS 2.x.

Notification name	**setLocalPriority**	vdSM -> vDC host
Notification Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the device
scene	integer	This specifies the scene to check for the *dontCare* flag. If it is set, nothing happens, if it is not set, the localPriority flag will be set on the device level.
group	optional integer	dS group number, present if local priority was not set for a single device only, but a zone/group (informational only, vdSM already creates separate calls for every involved device)
zoneID	optional integer	dS global zone ID number, present if local priority was not set for a single device only, but a zone/group (informational only, vdSM already creates separate calls for every involved device)

Dim Channel

- performs dimming a specific channel of the device. If the device does not have an output of the specified channel type, this method call is ignored

Notification name	**dimChannel**	vdSM -> vDC host
Notification Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the device
channel	integer	Channel to start or stop dimming: 0: dim the default channel (e.g. brightness for lights) 1..239: dim channel of the specified type, if any
mode	integer	1: start dimming upwards (incrementing output value) -1: start dimming downwards (decrementing output value) 0 : stop dimming
area	integer	0: no area restriction 1..4: only perform dimming action if the corresponding area on scene (Tx_S1) does not have the dontCare0 flag set.
group	optional integer	dS group number, present if the scene call was not applied to a single device, but a zone/group (informational only, vdSM already creates separate calls for every involved device)
zoneID	optional integer	dS global zone ID number, present if the scene call was not applied to a single device, but a zone/group (informational only, vdSM already creates separate calls for every involved device)

Call Minimum Scene

- if the device is off, set it to the minimal value needed to become logically switched on and participate in dimming. Otherwise, no action is taken.

Notification name	**callSceneMin**	vdSM -> vDC host
Notification Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the device
scene	integer	This specifies the scene to check for the *dontCare* flag. If it is set, nothing happens, if it is not set, and the device is off, the minimum output level will be set.
group	optional integer	dS group number, present if the call was not applied to a single device only, but a zone/group (informational only, vdSM already creates separate calls for every involved device)
zoneID	optional integer	dS global zone ID number, present if the call was not applied to a single device only, but a zone/group (informational only, vdSM already creates separate calls for every involved device)

Identify

- identify the device for the user - usually implemented as blinking the controlled light or an indicator LED the device might have. Depending on device type, the alert might be implemented differently, such as a beep, or hum or short movement.

Notification name	**identify**	vdSM -> vDC host
Notification Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the device
group	optional integer	dS group number, present if the identify notification was not sent to a single device, but a zone/group (informational only, vdSM already creates separate calls for every involved device)
zoneID	optional integer	dS global zone ID number, present if the identify notification was not sent to a single device, but a zone/group (informational only, vdSM already creates separate calls for every involved device)

setControlValue

- sets a control value (a dS "sensor value" coming in from a dS sensor or generated by a dSS app such as a heating regulator) to the device, which will evaluate it according to the device's group (color) and settings, and derive actual output values from it (for example a valve position).

Notification name	**setControlValue**	vdSM -> vDC host
Notification Parameter	Type	Description
dSUID	17 binary bytes representing dSUID	The dSUID of the device
name	string	The name of the control value. This defines the semantic meaning of the value and thus how the value will affect (or not) the output(s) of the device. In dS 1.0, control name values are mapped to dS "sensor type" (see dS sensor type table) numbers. See chapter “Control Values” in “vDC API properties” for a list of available control values
value	double	control value (aka dS sensor value)
group	optional integer	dS group number, present if the control value was not sent to a single device only, but to a zone/group (informational only, vdSM already creates separate calls for every involved device)
zoneID	optional integer	dS global zone ID number, present if the control value was not sent to a single device only, but to a zone/group (informational only, vdSM already creates separate calls for every involved device)