

digitalSTROM Virtual-Device-Connector API

digitalSTROM

Version: v1.4-branch*

May 22, 2017

*Revision: 6d0a8b6a9a98d5874ebfa542a78ea62bf3c04f8d

©2017 digitalSTROM AG. All rights reserved.

The digitalSTROM logo is a trademark of the digitalSTROM. Use of this logo for commercial purposes without the prior written consent of digitalSTROM may constitute trademark infringement and unfair competition in violation of international laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. digitalSTROM retains all intellectual property rights associated with the technology described in this document. This document is intended to assist developers to develop applications that use or integrate digitalSTROM technologies.

Every effort has been made to ensure that the information in this document is accurate. digitalSTROM is not responsible for typographical errors.

digitalSTROM AG
Building Technology Park Zürich
Brandstrasse 33
CH-8952 Schlieren
Switzerland

Even though digitalSTROM has reviewed this document, digitalSTROM MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT THIS DOCUMENT IS PROVIDED "AS IS", AND YOU, THE READER ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL DIGITALSTROM BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. NO DIGITALSTROM AGENT OR EMPLOYEE IS AUTHORIZED TO MAKE ANY MODIFICATION, EXTENSION, OR ADDITION TO THIS WARRANTY.

Contents

1	Basics	4
2	Terms	4
3	Discovery	5
3.1	Requirements	5
3.2	Solution	5
3.3	Notes	5
3.4	Avahi service description files	6
4	vDC host session	7
4.1	Basics	7
4.2	vDC host Session initialisation	7
4.3	vDC host Session operation	7
4.4	vDC host Session termination	8
5	vDC Announcement	8
6	vdSD Announcement host session	9
6.1	Device Announcement	9
6.2	Device Operation	9
6.3	Ending Device Operation	10
7	Device and vDC Operation Methods and Notifications	11
7.1	Named Property access	11
7.1.1	Reading Property values	12
7.1.2	Writing property values	14
7.1.3	Getting notified of property value (changes)	14
7.2	Presence polling	15
7.3	Actions	16
7.3.1	Call Scene	16
7.3.2	Save Scene	16
7.3.3	Undo Scene	17
7.3.4	Set local priority	17
7.3.5	Dim Channel	18
7.3.6	Call Minimum Scene	18
7.3.7	Identify	19
7.3.8	Set Control Value	19
7.3.9	Set Output Channel Value	20
7.3.10	Call Action	20
7.4	Configuration Control and Processes	20
7.4.1	Learn-In and -Out	20
7.4.2	Authenticate	21
7.4.3	Firmware Update	21
8	Change Log	22

Basics

- based on [Google 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 preliminary 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

Term	Description
vDC	virtual device connector. A vDC is primarily a logical entity within the dS system and has its own dSUID. A vDC connects to a Native IP Device or to a IP Gateway with Native Gateway Connected 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.
vDC session	logical connection between a vdSM and a vDC host (representing one or multiple vDCs)
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 digitalSTROM device. A vdSD represents a dS-Device system

Discovery

Requirements

- A vDC host must be discoverable by vdSMs automatically on a given network. Discovery must work without any UI on the vDC host side.
- It must be avoided that a vdSM connects to a wrong (neighbour's, for example) vDC host

Solution

- vDC hosts 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.
- To avoid wrong connections, the vdSM which initiates a connection must be able to check possible vDC announcements received via Avahi against an optional whitelist. If the whitelist exists, only listed vDCs might be connected. The idea is that in small/simple installations connection is fully automatic, but if conflicts arise in large setups (like show rooms or developer environments) these can be solved by adding whitelists. Whitelists are located and maintained on the dss device.

Notes

- In a future version, the dSS configurator will provide a user interface to view and edit the whitelist.
- virtual device gateways productive with dSS version 1.9 currently have a vdsM of their own in the gateway, which also announces itself via Avahi. This was required as a temporary solution, but will be phased out with the next dSS version. For new developments, running vdSM instances on devices other than a dSS is not allowed (nor needed) for production environments.
- The next planned version of the discovery mechanism will be smarter in avoiding duplicate connections and keeping existing setup stable. It will also be able to seamlessly migrate now productive virtual device gateways with a separate vdsM to using a dSS-hosted vdsM.
- vDC implementations adhering to 3.2 will continue to work as before.

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 advertisement 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	string of 34 hex characters (2*17)	dSUID of the vdSM
api_version	integer	vDC API version The API version as described in this document is 2
Response		
dSUID	string of 34 hex characters (2*17)	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 <i>another</i> vdSM. RESULT_INCOMPATIBLE_API: The vdSM does not support the API version presented in <i>APIVersion</i>
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: Generic Response		
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	string of 34 hex characters (2*17)	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 host session

- 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 reasonably 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

Method	announcedevice	vDC host -> vdSM
Request Parameter	Type	Description
dSUID	string of 34 hex characters (2*17)	The dSUID of the device to be announced
vdc_dSUID	string of 34 hex characters (2*17)	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 Chapter 7 on Device Level Method Notifications for a specification of the supported individual methods and notifications.

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.

Method	vanish	vDC host -> vdSM
Request Parameter	Type	Description
-	-	-
Response: Generic Response		
code	string of 34 hex characters (2*17)	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)

Method	remove	vdSM -> vDC host
Request Parameter	Type	Description
dSUID	string of 34 hex characters (2*17)	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

- *Addressable 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.
- Addressable 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 *addressable 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	getProperty	vdSM -> vDC host
Request Parameter	Type	Description
dSUID	string of 34 hex characters (2*17)	The dSUID of the entity (device, vDC or vDC host) to read properties from
query	property elements sub-messages	<p>A property tree structure consisting of property elements, but with no values, specifying the property or properties to be returned.</p> <ul style="list-style-type: none"> 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) will be returned. If the empty name wildcard is the last element specified in a query tree branch, this means all deeper levels of that branch should be returned as well.
Response: ResponseGetProperty		
properties	property elements sub-messages	<p>A property tree structure consisting of property elements and values representing the properties selected by the <i>query</i> structure.</p> <ul style="list-style-type: none"> The response has basically the same tree structure as the query. Where wildcards were used in the query, the response tree will expand beyond the query structure returning all substructures. Where unknown/nonexisting properties were requested in the query, the response tree will not have a corresponding element. Some properties might exist but not have a value at the time of the query - these will return an explicit NULL value.

Error case: GenericResponse

code	integer	RESULT_FORBIDDEN: the property exists but cannot be read (write-only, uncommon case).
message	string	explanation text

Writing property values

Method	setProperty	vdSM -> vDC host
Request Parameter	Type	Description
dSUID	string of 34 hex characters (2*17)	The dSUID of the entity (device, vDC or vDC host) to write properties to
properties	property elements sub-messages	<p>A property tree structure consisting of property elements and values representing the properties to be written.</p> <ul style="list-style-type: none"> 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. <p>Note: channelState property must not be changed using setProperty, but with the separate <i>setOutputChannelValue</i> action (see below)</p> <p>Note: if writing multiple property values in one request, failing to write any of these will cause setProperty to return an error. However, some of the other properties included in the same request might already be stored (in any order, not necessarily in the order of values as passed in properties). Use separate requests if you need separate ok/failure status for each value.</p>
Response: GenericResponse		
code	integer	0 = success
Error case: GenericResponse		
code	integer	<p>RESULT_INVALID_VALUE_TYPE: passed type is wrong and cannot be written.</p> <p>RESULT_FORBIDDEN: the property exists but cannot be read (write-only, uncommon case).</p> <p>RESULT_NOT_FOUND: the receiver (device or vDC itself) specified through dSUID is unknown at the callee.</p>

Getting notified of property value (changes)

- 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	string of 34 hex characters (2*17)	The dSUID of the entity (device, vDC or vDC host) from where this notification originates
properties	property elements sub-messages	A property tree structure consisting of property elements and values representing the information pushed to the vdSM.

Presence polling

- 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	string of 34 hex characters (2*17)	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	string of 34 hex characters (2*17)	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	string of 34 hex characters (2*17)	One or multiple dSIDS of device(s) this call applies to.
scene	integer	dS scene number to call
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	string of 34 hex characters (2*17)	One or multiple dSIDS of device(s) this call applies to.
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	string of 34 hex characters (2*17)	One or multiple dSIDS of device(s) this call applies to.
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	string of 34 hex characters (2*17)	One or multiple dSIDS of device(s) this call applies to.
scene	integer	This specifies the scene to check for the <i>dontCare</i> flag. If it is set, nothing happens, if it is not set, the <i>localPriority</i> flag will be set on the device level.
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)

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	string of 34 hex characters (2*17)	One or multiple dSIDS of device(s) this call applies to.
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	string of 34 hex characters (2*17)	One or multiple dSIDS of device(s) this call applies to.
scene	integer	This specifies the scene to check for the <i>dontCare</i> 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 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)

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	string of 34 hex characters (2*17)	One or multiple dSIDS of device(s) this call applies to.
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 Control Value

- sets the value of an output channel.

Notification name	setControlValue	vdSM -> vDC host
Notification Parameter	Type	Description
dSUID	string of 34 hex characters (2*17)	One or multiple dSIDS of device(s) this call applies to.
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 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 Output Channel Value

- sets the value of an output channel.

Notification name	setOutputChannelValue	vdSM -> vDC host
Notification Parameter	Type	Description
dSUID	string of 34 hex characters (2*17)	One or multiple dSIDS of device(s) this call applies to.
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
apply_now	optional boolean	if omitted or set to false, the new output value will be buffered but not yet applied to the hardware. The next <i>setOutputChannelValue</i> call with <i>apply_now</i> omitted or set to true will apply all buffered values.
value	double	Value to be applied to the output; maybe omitted if "automatic" flag is set
automatic	optional boolean	Switch device to internal automatic control logic
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 Action

- calls an action on the device

Notification name	GenericRequest	vdSM -> vDC host
Notification Parameter	Type	Description
method	string	invokeDeviceAction
dSUID	string of 34 hex characters (2*17)	One or multiple dSUIDs of device(s) this call applies to.
id	string	device action to execute
params	optional	additional parameters, json encoded object

Configuration Control and Processes

Learn-In and -Out

- starts the learn-in or learn-out process

Notification name	GenericRequest	vdSM -> vDC host
Notification Parameter	Type	Description
method	string	pair
dSUID	string of 34 hex characters (2*17)	One or multiple dSUIDs of device(s) this call applies to.

establish	bool	true = learn-in, false = learn-out
timeout	integer	timeout in seconds, -1 to disable timeout
params	optional	additional parameters, json encoded object

Authenticate

- starts the authentication process

Notification name	GenericRequest	vdSM -> vDC host
Notification Parameter	Type	Description
method	string	authenticate
dSUID	string of 34 hex characters (2*17)	One or multiple dSUIDs of device(s) this call applies to.
authData	string	json encoded object with authentication tokens
authScope	string	context, e.g. user id or device reference
params	optional	additional parameters, json encoded object

Firmware Update

- starts the firmware upgrade process

Notification name	GenericRequest	vdSM -> vDC host
Notification Parameter	Type	Description
method	string	firmwareUpgrade
dSUID	string of 34 hex characters (2*17)	One or multiple dSUIDs of device(s) this call applies to.
checkonly	bool	true for a dry-run check only
clearsettings	bool	true for upgrade and reset of all configuration data to factory defaults
params	optional	additional parameters, json encoded object

Change Log

Document History for DocumentID: digitalSTROM Virtual-Device-Connector API

date	change description
2014-12-01	Initial version 1.0
2015-05-21	Version 1.1 Running vdSM instances on devices other than a dSS is not allowed for production environments