plan44 vdcd external device API

Contents

About vdcd external device API 2
External Device API operation 2
Message Format 2
Init Message 3
Init message structure 3
Button object in the buttons field of the init message 5
Input object in the inputs field of the init message 6
Sensor object in the sensors field of the init message 7
Messages from vdcd to device 8
Messages from device to vdcd 9
Experimenting 10
Light button simulation 10
Light dimmer simulation 10
Temperature sensor simulation 11

About vdcd external device API

This document describes the socket based API included in the plan44.ch *vdcd* (virtual device connector daemon).

The external device API allows external scripts and programs to register themselves to the *vdcd* to implement custom digitalSTROM devices with very little effort.

To host external devices, *vdcd* must be started with the *--externaldevices* option, providing a port number or absolute unix socket path for device implementation to connect to. For security reasons, it is recommended to run the scripts and programs implementing devices on the same device as vdcd itself, however for development purposes the *--externalnonlocal* command line option can be specified to allow device API connections from non-local clients.

The plan44.ch digitalSTROM products P44-DSB-DEH and P44-DSB-E support the *external device API* from Version 1.5.0.8 onwards. However, at the time of writing the *external device API* is active only for devices enabled for "testing"/beta (available upon request). In the free "P44-DSB-X" plan44.ch image for RaspberryPi, the *external device API* is always enabled. By default, vdcd used port 8999 for the *external device API*

External Device API operation

Each external device implementation needs to

- open a connection to the TCP port or unix socket specified with the *--externaldevices* vdcd command line option (usually port 8999).
- send a *init* message declaring the properties of the device (specifying outputs, inputs, names, default group membership etc.). The *init* message uses JSON syntax. However, no JSON support is actually needed in a device implementation, because the *init* message can specify to use a extremely simple text protocol for any communication beyond the *init* message itself. And the *init* message is usually a constant string that can be sent by any language.
- enter a loop, waiting for messages from the *vdcd* indicating output channel changes, or sending messages to the *vdcd* indicating input changes.
- When connection closes (due to error or when vdcd explicitly closes it), the device implementation should restart, see first bullet point. This can be achieved within the device implementation itself, or by having the device implementation run as a daemon under control of a daemon supervisor like *runit*, which re-starts daemons when they terminate.

Message Format

Messages consist of strings, delimited by a single LF (0x0A) character. The *init* message must always be in JSON format. Further messages are either JSON or simple text messages, depending on the *protocol* option in the *init* message (see below).

Init Message

The init message is sent by a external device implementation as the first message after opening the socket connection. It needs to be formatted as a single line JSON object. It describes the device's outputs and inputs and other properties, such that vdcd can instantiate an appropriate digitalSTROM device with all standard behaviour required.

A simple init message for a light dimmer might look like (on a single line):

```
{'message':'init','protocol':'simple','output':'light','name':'ext
dimmer','uniqueid':'myUniqueID1234'}
```

The following tables describes all possible fields of the *init* message JSON object:

Init message structure

Field	Type	Description
message	string	identifies the message type, must be **init** for the *init* message
protocol	optional sting	Can be set to **simple** to use the simple text protocol for all further communication beyond the *init* message. This allows implementing devices without need for any JSON parsing. If set to **json** (default), further API communication are JSON messages.
uniqueid	string	This string must uniquely define the device at least among all other external devices connected to the same vdcd, or even globally. To identify the device globally, use a UUID string (XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX) or a valid 34 hex digit digitalSTROM dSUID. To identify the device uniquely among all other devices in the same vdcd, use any other string.
group	optional integer	defines the primary color (group) of the device: 1: yellow/light, 2: grey/shadow 3: blue/heating 4: cyan/audio 5: magenta/video 6: red/security 7: green/access 8: black/joker 9: white/cooling 10: ventilation 11: windows

output	optional string	Defines the type of output: light: dimmer output with light behaviour colorlight: 6-channel digitalSTROM color light (brightness, hue, saturation, colortemp, cieX, cieY) movinglight: color light with additional X and Y position channels heatingvalve: 0..100% heating valve shadow: jalousie type device with position and angle channel. Also see *move* field. basic: basic 0..100% output with no special behaviour. Can be used for relay outputs. Default is no output.
kind	optional string for “shadow” output type	Defines the kind of shadow device roller: simple roller blind, no angle sun: sunblind jalousie: jalousie with blade angle control
endcontacts	optional boolean for “shadow” output type	If set to true, device implementation must report reaching top and bottom positions by updating channel value to 100 or 0, resp. (using the “channel” message). Otherwise, the shadow behaviour uses move time settings to derive actual positions from timing alone.
move	optional boolean	If set to true, the device must support the "move" or "MV" message (see below), which is issued by the device to start or stop a movement (increase, decrease) of the output value. The move semantic might be more useful for blind type devices than channel output values. Default is false (no move semantics)
sync	optional boolean	if set to true, the device mus support the "sync" message and must respond to "sync" with the "synced" message. "sync" is issued by the vdcd when it needs to know current output values (e.g. for a saveScene operation). "synced" is sent by the device when updated output channel values have been sent to the vdcd. Default is false (no output value sync requests)
groups	optional array of integers	can be used to specify output group membership different from the defaults of the specified *output* type.
hardwarename	optional string	a string describing the type of hardware, such as "dimmer" or "relay" etc.
name	optional string	the default name the device will have in the digitalSTROM system. Note that this can be changed by the user via dSS Web interface.
buttons	optional array of objects	Defines the buttons of the device. See table below for fields in the button objects
inputs	optional array of objects	Defines the binary inputs of the device. See table below for fields in the input objects
sensors	optional array of objects	Defines the sensors of the device. See table below for fields in the sensor objects

Button object in the buttons field of the init message

Field	Type	Description
id	optional integer	identifies the hardware button this button input belongs to. Two-way or multi-way buttons will have multiple button definitions with the same id. Defaults to 0
buttontype	optional integer	Defines the type of button: 0: kind of button not defined by device hardware 1: single pushbutton 2: two-way pushbutton or rocker 3: 4-way navigation button 4: 4-way navigation with center button 5: 8-way navigation with center button 6: On-Off switch Defaults to 1 (single pushbutton)
element	optional integer	Defines which element of a multi-element button is represented by this button input: 0: center element / single button 1: down, for 2,4,8-way 2: up, for 2,4,8-way 3: left, for 2,4,8-way 4: right, for 2,4,8-way 5: upper left, for 8-way 6: lower left, for 8-way 7: upper right, for 8-way 8: lower right, for 8-way Default is 0 (single button)
group	optional integer	defines the primary color (group) of the button: 1: yellow/light, 2: grey/shadow 3: blue/heating 4: cyan/audio 5: magenta/video 6: red/security 7: green/access 8: black/joker Defaults to primary device group
hardwarename	optional string	a string describing the button element, such as "up" or "down" etc.

Input object in the inputs field of the init message

Field	Type	Description
inputtype	optional integer	Defines the type of input: 0: no system function 1: Presence 2: Light 3: Presence in darkness 4: twilight 5: motion 6: motion in darkness 7: smoke 8: wind 9: rain 10: solar radiation (sun light above threshold) 11: thermostat (temperature below user-adjusted threshold) 12: device has low battery 13: window is open 14: door is open 15: window is tilted instead of fully opened 16: garage door is open 17: protect against too much sunlight 18: frost detector Defaults to 0 (no system function)
usage	optional integer	Defines usage: 0: undefined 1: room (indoors) 2: outdoors 3: user interaction Default is 0 (undefined)
group	optional integer	defines the primary color (group) of the button: 1: yellow/light, 2: grey/shadow 3: blue/heating 4: cyan/audio 5: magenta/video 6: red/security 7: green/access 8: black/joker Defaults to primary device group
updateinterval	optional double	defines the expected update interval of this input, i.e. how often the actual state is reported by the device. Defaults to 0, which means no fixed interval
hardwarename	optional string	a string describing the button element, such as "up" or "down" etc.

Sensor object in the sensors field of the init message

Field	Type	Description
sensortype	optional integer	Defines the type of sensor: 0: undefined 1: temperature in degrees celsius 2: relative humidity in % 3: illumination in lux 4: supply voltage level in Volts 5: CO (carbon monoxide) concentration in ppm 6: Radon activity in Bq/m3 7: gas type sensor 8: dust, particles <10μm in µg/m3 9: dust, particles <2.5μm in µg/m3 10: dust, 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 14: Power in W 15: Electric current in A 16: Energy in kWh 17: Electric Consumption in VA 18: Air pressure in hPa 19: Wind direction in degrees 20: Sound pressure level in dB 21: Precipitation in mm/m2 22: CO2 (carbon dioxide) concentration in ppm Defaults to 0 (undefined)
usage	optional integer	Defines usage: 0: undefined 1: room (indoors) 2: outdoors 3: user interaction Default is 0 (undefined)
group	optional integer	defines the primary color (group) of the button: 1: yellow/light, 2: grey/shadow 3: blue/heating 4: cyan/audio 5: magenta/video 6: red/security 7: green/access 8: black/joker Defaults to primary device group
updateinterval	optional double	defines the expected update interval in seconds of this sensor, i.e. how often the actual value is reported by the device. Defaults to 5 seconds
hardwarename	optional string	a string describing the button element, such as "up" or "down" etc.
min	optional double	minimal value, defaults to 0
max	optional double	maximal value, defaults to 100
resolution	optional double	sensor resolution, defaults to 1

Messages from vdcd to device

vdcd sends (depending on the features selected in the *init* message) the following messages:

JSON protocol	Simple protocol	Description
{ *'message':'status',* *'status':**s**,* *'errorcode':**e**,* *'errormessage':**m**,* *'errordomain':**d*** *}*	OK or ERROR=**m**	Status for *init* message. If ok, **s** is the string "ok" in the JSON protocol. **m** is a textual error message **e** is the vdcd internal error code **d** is the vdcd internal error domain
{ *'message':'channel',* *'index':**i**,* *'type':**t**,* *'value':**v*** *}*	C**i**=**v**	Output channel index **i** has changed its value to **v**. v is a double value. The device implementation should forward the new channel value to the device's output. The JSON variant of this message additionally reports the channel type as **t**: 0: undefined 1: brightness for lights 2: hue for color lights 3: saturation for color lights 4: color temperature for lights with variable white point 5: X in CIE Color Model for color lights 6: Y in CIE Color Model for color lights 7: vertical position 8: horizontal position 9: opening angle position 10: permeability
{ *'message':'move',* *'index':**i**,* *'direction':**d*** *}*	MV**i**=**d**	When the init message has specified *move=true*, the vdcd can request starting or stopping movement of channel **i** as follows: 0: stop movement 1: start movement to increase channel value -1: start movement to decrease channel value
{ *'message':'sync'* *}*	SYNC	When the init message has specified *sync=true*, the vdcd can request updating output channel values by sending *sync*. The device is expected to update channel values (using the "channel"/"C" message, see below) and then sending the *synced* message.

Messages from device to vdcd

the device can send (depending on the features selected in the *init* message) the following messages:

JSON protocol	Simple protocol	Description
{ *'message':'bye'* *}*	BYE	The device can send this message to disconnect from the vdcd, for example when it detects its hardware is no longer accessible. Just closing the socket connection has the same effect as sending *bye*
{ *'message':'channel',* *'index':**i,*** *'value':**v*** *}*	C**i**=**v**	The device should send this message when its output channel index **i** has changed its value to **v** for another reason that having received a channel message (e.g. after initialisation, or for devices than can be controlled directly). Devices that cannot immediately detect output changes can specifiy *sync=true* in the *init* message, so the vdcd will request updating output channel values by sending *sync* only when these values are actually needed.
{ *'message':'button',* *'index':**i,*** *'value':**v*** *}*	B**i**=**v**	The device should send this message when the state of its button at index **i** has changed. If the button was pressed, **v** must be set to 1, if the button was released, **v** must be set to 0. To simulate a button press+release with a single message, set **v** to the press duration in milliseconds.
{ *'message':'input',* *'index':**i,*** *'value':**v*** *}*	I**i**=**v**	The device should send this message when the state of its input at index **i** has changed. If the input has changed to active **v** must be set to 1, if the input has changed to inactive, **v** must be set to 0.
{ *'message':'sensor',* *'index':**i,*** *'value':**v*** *}*	S**i**=**v**	The device should send this message when the value of its sensor at index **i** has changed. **v** is the new value (double) and should be within the range specified with *min* and *max* in the *init* message.
{ *'message':'synced'* *}*	SYNCED	The device must send this message after receiving *sync* and having updated output channel values

Experimenting

The external device API can be experimented easily with by connecting via telnet, then pasting an *init* message and then simulating some I/O.
The following paragraphs show this for different device types. Please also refer to the sample code in different languages contained in the *external_devices_samples* folder of the vdcd project.

Light button simulation

Connect to the device API with telnet:

```
telnet localhost 8999
```

Now copy and paste (single line!!) a simple *init* message defining a light button:

```
{'message':'init','protocol':'simple','uniqueid':'experiment42','buttons':
[{'buttontype':1,'group':1,'element':0}]}
```

The vdcd responds with:

```
OK
```

Now, the vdcd has created a light button device. If this is the vdcd of a P44-DSB, you can see the device in the P44-DSB web interface, and if the vdcd is connected to a digitalSTROM system, a new button device will appear in the dSS. By default, it will be in the default room for the “external devices” vdc, but you can drag it to an existing room with digitalSTROM light devices.

Now you can switch lights by simulating a button click (200ms active) with

```
B0=200
```

For dimming, the button can be held down...

```
B0=1
```

...and later released

```
B0=0
```

Light dimmer simulation

Connect to the device API with telnet:

```
telnet localhost 8999
```

Now copy and paste (single line!!) a simple *init* message defining a dimmer output:

```
{'message':'init','protocol':'simple','uniqueid':'experiment42b','output':‘
light’}
```

The vdcd responds with:

```
OK
```

Now, the vdcd has created a light dimmer device. If this is the vdcd of a P44-DSB, you can see the device in the P44-DSB web interface, and if the vdcd is connected to a digitalSTROM system, a new light device will appear in the dSS

Now you can call scenes in the room that contains the light device (or use the sprocket button in the P44-DSB web interface to directly change the brightness). You will see channel value changes reported from the external device API:

```
C0=3.120000
C0=8.190000
C0=14.040000
C0=21.840000
C0=30.810000
C0=40.950000
C0=56.160000
C0=63.960000
C0=69.810000
C0=78.000000
C0=79.950000
C0=78.000000
C0=65.130000
C0=53.040000
C0=42.120000
C0=33.150000
C0=26.910000
C0=40.950000
C0=53.040000
C0=58.110000
C0=63.180000
```

Temperature sensor simulation

Connect to the device API with telnet:

```
telnet localhost 8999
```

Now copy and paste (single line!!) a simple *init* message defining a light button:

```
{'message':'init','protocol':'simple','group':
3,'uniqueid':'experiment42c','sensors':[{'sensortype':1,‘usage’:1,‘group’:
48,'min':0,'max':40,'resolution':0.1}]}
```

The vdcd responds with:

```
OK
```

Now, the vdcd has created a temperature sensor device.

Now you can simulate temperature changes with

```
S0=22.5
```

e.g. to report a room temperature of 22.5 degree celsius.