plan44 vdcd APIs

Contents

Basics

- this document describes plan44.ch specific APIs of the plan44 vdcd (vdc daemon) beyond the official digitalSTROM vDC API as described in "vdSM vDC API" and "vdSD properties"

Terms

vdcd	plan44.ch's C++ implementation of a vdc host in form of a daemon process running on Linux or Mac OS X.
vDC api	the official digitalSTROM API for virtual device connectors as described in "vdSM vDC API" and "vdSD properties".
config API	a separate, TCP socket + JSON based API to access the vDC structures mainly for Web configuration. The mg44 (mongoose webserver variant) can be used as proxy to provide this API as a http based JSON API.

Config API

When the vdcd is started with the command line option --cfgapiport, the config API can be accessed opening a TCP connection to this port. Standard port number is 8090. Usually, an instance of the mg44 (mongoose based) web server is handling incoming JSON http requests and translates these into TCP socket connection JSON queries.

Config API TCP socket level JSON requests

Direct TCP socket level JSON requests to the config API have a form like:

```
{ "method":"GET", "uri":"vdc", "uri_params": {"method": "getProperty",
"param1": "x", "param2": 42 }, "data": {"foo":42, "bar":77 }}
```

Parameter	Type	Description
method	string	the HTTP method which generated this request
uri	string	the part of the path of the HTTP uri not already processed by the mg44 webserver to differentiate API calls from regular pages. Usually the complete uri is something like `http://host/api/json/`***`path`***`?name=val&name2=val2.` The "uri" parameter carries the "path" part.
uri-params	object	JSON object containing the variables passed in the CGI parameters. `http://host/api/json/path?`**`name=val&name2=val2.`**

data	optional JSON	For HTTP post requests, this contains the raw data from the POST request, which is expected to be proper JSON.

Config API TCP socket level JSON responses

Responses from the API have one of two possible forms:

- For successful requests, the JSON response is an object with one field named "result". Its value can be NULL if no actual result was returned, or it will contain the result:

```
{"result":42 }
```

- For errors, the JSON response is an object with three fields:

```
{"error":400,"errormessage":"unknown method","errordomain":"p44vdc" }
```

Config API HTTP level JSON requests

When the mg44 web server is used to convert the Config API to http, the HTTP level JSON requests work as follows:

HTTP GET

To call a API method using HTTP GET, specify the method and all parameters as CGI parameters in the URL, like:

```
GET http://localhost:8080/api/json/vdc?method=getProperty&name=
buttonInputDescriptions&index=0&dSUID=3504175FE000000044BA3480
```

HTTP POST/PUT

To call a API method using HTTP POST or PUT, specify the method and all parameters as JSON object in the data part of the request, like:

```
POST http://localhost:8080/api/json/vdc

{ "method":"getProperty", "name":"buttonInputDescriptions", "index":0,
"dSUID":"3504175FE000000044BA3480" }
```

Config API HTTP level JSON responses

Same as TCP socket level JSON responses

Config API passthrough to regular vDC API

The config API can be used to query/control all the properties and call the actions the regular vDC API provides to the digitalSTROM system.

To access the vDC API methods and notifications (details see "vdSM vDC API" and "vdSD properties" documents) via the config API, the "uri" parameter must be set to "vdc" in requests.

For direct socket access, the request will look like:

```
{ "method":"GET", "uri":"vdc", "uri_params": {"method": "getProperty",
"name": "name":"buttonInputDescriptions", "index":0,
"dSUID":"3504175FE000000044BA3480"}}
```

For access via mg44 and HTTP, the request will look like:

```
GET http://localhost:8080/api/json/vdc?method=getProperty&name=
buttonInputDescriptions&index=0&dSUID=3504175FE000000044BA3480
```

To call vDC API notifications (calls that don't return anything), use the "notification" keyword instead of "method":

```
GET http://localhost:8080/api/json/vdc?
notification=callScene&scene=5&force=0&dSUID=3504175FE000000044BA3480
```

The result from calling a notification is always NULL.

Navigating the device tree via config API

In the regular digitalSTROM vdc API, all devices in the vdcd will be announced to the system along with their dSUIDs (unique digitalSTROM system IDs). The dSUIDs are required to access properties of devices, such as descriptions, status and config.

The config API passthrough does not have the announce mechanism, but provides a way to query all vdcs within the vdcd (one vdc is the instance responsible for one class of devices, such as enOcean, or DALI, or hue), and all devices within one vdc.

To get a list of all vdcs (device classes) query the *x-p44-vdcs* property of the root object as follows:

```
GET http://localhost:8080/api/json/vdc?method=getProperty&name=x-p44-
vdcs&dSUID=root
```

The result is a list of device class dSUIDs:

```
{"result":
["17EA3027642C5CACC0943AFDA3A4765800","5524FB72CA935A73802D148648A2E76900"]
}
```

Using these, details about the classes can be queried, such as the *model*, giving more information about the device class:

```
GET http://localhost:8080/api/json/vdc?
method=getProperty&name=model&dSUID=17EA3027642C5CACC0943AFDA3A4765800
```

To get a list of all devices within a class query the *x-p44-devices* property of the device class:

```
GET http://localhost:8080/api/json/vdc?method=getProperty&name=x-p44-
devices&dSUID=17EA3027642C5CACC0943AFDA3A4765800
```

The result is a list of device dSUIDs:

```
{"result":
["3504175FE00000000845C480","3504175FE00000000061BE30","3504175FE0000000103
57980","3504175FE00000001035C630","3504175FE000000008B1C5C0","3504175FE0000
00008B1C5C2"]}
```

Using these IDs, details about every device can be queried, like again *model*:

```
GET http://localhost:8080/api/json/vdc?
method=getProperty&name=model&dSUID=3504175FE00000000845C480
```

The result:

```
{"result":"Kieback + Peter enOcean device"}
```

Wildcard property queries

Instead of querying properties one by one, the plan44 vDC API supports querying all properties of an object with a single call. Two variants are available:

- When using "*" as name in a getProperty call, the entire property tree of the addressed device will be returned, including possibly large arrays like "scenes".
- When using "#" as name (encode as %23 in URLs!) in a getProperty call, all simple value properties of the device will be returned, and for each array property, the number of elements in that array (but not the contents).

First level values and array sizes only:

```
localhost:8080/api/json/vdc?method=getProperty&name=
%23&dSUID=3504175FE00000000845C480

{"result":{"dSUID":"3504175FE00000000845C480","model":"Kieback + Peter
enOcean device","hardwareGuid":"enoceanaddress:8674376","name":"alpha.drive
luz","primaryGroup":3,"isMember":64,"zoneID":
0,"localPriority":false,"progMode":false,"idBlockSize":
1,"buttonInputDescriptions":0,"buttonInputSettings":0,"buttonInputStates":
0,"binaryInputDescriptions":0,"binaryInputSettings":0,"binaryInputStates":
0,"outputDescriptions":1,"outputSettings":1,"outputStates":
1,"sensorDescriptions":1,"sensorSettings":1,"sensorStates":1,"scenes":
0,"undoState":null}}
```

Full property tree:

```
http://localhost:8080/api/json/vdc?
method=getProperty&name=*&dSUID=3504175FE00000000845C480

{"result":{"dSUID":"3504175FE00000000845C480","model":"Kieback + Peter
enOcean device","hardwareGuid":"enoceanaddress:8674376","name":"alpha.drive
luz","primaryGroup":3,"isMember":
[true,false,false,true,false,false,false,false,false,false,false,false,fals
e,false,false,false,false,false,false,false,false,false,false,false,false,f
alse,false,false,false,false,false,false,false,false,false,false,false,fals
e,false,false,false,false,false,false,false,false,false,false,true,false,fa
lse,false,false,false,false,false,false,false,false,false,false,false,false
,false],"zoneID":0,"localPriority":false,"progMode":false,"idBlockSize":
1,"buttonInputDescriptions":[],"buttonInputSettings":
[],"buttonInputStates":[],"binaryInputDescriptions":
[],"binaryInputSettings":[],"binaryInputStates":[],"outputDescriptions":
```

[{"name":"Valve, 0.000..100.000 ","type":"output","function":
2,"outputUsage":1,"variableRamp":false,"maxPower":
0.000000}],"outputSettings":[{"group":48,"mode":
2,"pushChanges":false}],"outputStates":[{"error":0,"value":0,"age":
404.348000}],"sensorDescriptions":[{"name":"Temperature, 0.000..40.000 Â
°C","type":"sensor","sensorType":1,"sensorUsage":1,"min":0.000000,"max":
40.000000,"resolution":0.156863,"updateInterval":
100.000000}],"sensorSettings":[{"group":48,"minPushInterval":
2.000000,"changesOnlyInterval":0.000000}],"sensorStates":[{"error":
0,"value":25.411766,"age":404.348061}],"scenes":[],"undoState":null}}