Devialet IP Control

REFERENCE API DOCUMENTATION
Revision 1 - December 2021

DEVIALET

Table of contents

Overview

Introduction

Starting with DOS 2.14, Devialet devices offer a wide range of commands and information, made available via standard HTTP requests.

These commands can be used as a complement or a replacement to the Devialet companion app for iOS and Android to enable advanced automation scenarios and/or integration with other systems.

This document contains the reference API documentation. It can be used by system integrators and enthusiast developers alike.

Color code

[DOS >= 2.15]	The minimal firmware version.
`deviceId`	A field name.
`/devices/{deviceId}/identify`	An endpoint.
`"InvalidValue"`	Error codes.
GET, POST	HTTP methods.

Minimal firmware versions

For each endpoint or, when needed, its particular properties (parameters, fields, HTTP methods), the minimal embedded FW version is mentioned in the description.

DOS firmware family:

FW version	Description
DOS 2.14.x	- Support for Phantom I, Phantom II, Dialog, and Arch - List available sources and their properties, including sources of the accessories - Select any source - Play/Pause/Mute/Unmute/Next/Previous when eligible - Volume control - Getting the system name - Playback status info - Stream metadata (artist, album, track) when eligible
DOS 2.16.x	- Support for upcoming Devialet products - Launch Bluetooth pairing - Night mode setting - Equalizer

	- Power off, restart, reset to factory settings

Discovery

In order to use the IP Control API, the client application must know the IP addresses of the Devialet devices present in the local network.

One possibility is to configure the Devialet devices or the network router in such a way that the Devialet devices always use the same IP addresses, and to configure the client application accordingly.

If the client application can not use fixed IP addresses, it is possible to discover the IP addresses of the Devialet devices in the local network using mDNS. Indeed, Devialet devices register mDNS service instances for the `_http._tcp` service type. Here is an example of a scanning tool output:

```
$ avahi-browse -r _http._tcp
+  wlan0 IPv4 Living room                                  Web Site             local
=  wlan0 IPv4 Living room                                  Web Site             local
   hostname = [PhantomII98dB-L32Z12345TQ9A.local]
   address = [192.168.1.26]
   port = [80]
   txt = ["path=/"]
+  wlan0 IPv4 Living room-ipcontrol                        Web Site             local
=  wlan0 IPv4 Living room-ipcontrol                        Web Site             local
   hostname = [PhantomII98dB-L32Z12345TQ9A.local]
   address = [192.168.1.26]
   port = [80]
   txt = ["path=/ipcontrol/v1" "ipControlVersion=1" "manufacturer=Devialet"]
+  wlan0 IPv6 Living room                                  Web Site             local
=  wlan0 IPv6 Living room                                  Web Site             local
   hostname = [PhantomII98dB-L32Z12345TQ9A.local]
   address = [192.168.1.26]
   port = [80]
   txt = ["path=/"]
+  wlan0 IPv6 Living room-ipcontrol                        Web Site             local
=  wlan0 IPv6 Living room-ipcontrol                        Web Site             local
   hostname = [PhantomII98dB-L32Z12345TQ9A.local]
   address = [192.168.1.26]
   port = [80]
   txt = ["path=/ipcontrol/v1" "ipControlVersion=1" "manufacturer=Devialet"]
```

The client application should use the content of the `txt` record to filter the service instances. In particular, it has to contain the following key-value pairs:

- `"manufacturer=Devialet"`
- `"ipControlVersion=1"`

Then, it should use the provided `address`, `port`, and `path` (in the `txt` record) values to access the API.

Note: the client application should not use the mDNS service name for display purposes, but rather the `systemName` value from the response to the call to `/systems/{systemId}` (see below). Indeed, mDNS automatic name conflict resolution may interfere with the service name value in unexpected ways.

One may notice that unlike various user-friendly device names, the `hostname` parameter does not change even if the name of the device or its system is changed. The following prefixes are used by Devialet devices:

- `PhantomI`
- `PhantomII`
- `Arch`
- `Dialog`

However, for future compatibility reasons, it is strongly recommended not to filter the records using the `hostname` parameter, or use this parameter in any way.

To access the device's model name or serial number, calling the `/devices/{deviceId}` endpoint (see below) is the preferred method.

Please note that in the example above the IPv6 record provides an IPv4 address. This behaviour is not guaranteed, the address entries may be in either IPv4 or IPv6 format.

Protocol and authentication methods

IP Control uses HTTP protocol. No authentication is required.

The global prefix

All requests are performed on the URLs of the form:

`http://IPADDRESS/ipcontrol/v1/PATH/TO/ENDPOINT`

Example:

`http://192.168.1.20/ipcontrol/v1/devices/current/identify`

Note: the `/ipcontrol/v1` part of the URL can change in the future. It is strongly recommended to use the value of the `path` key of the `txt` record of the corresponding mDNS service instance (see "Discovery" section above).

In the following documentation, the `"http://IPADDRESS/ipcontrol/v1"` prefix is omitted for clarity.

One can use both IPv4 and IPv6 addresses.

Note that virtually all browsers forbid using link-local IPv6 addresses in the address bar, but they should work in command-line tools like `curl`. Example:

```
curl -H 'Content-Type:' -X POST -d '{}' -g -6
'http://[fe80::525b:c2ff:fe9c:7955%enp0s31f6]:80/ipcontrol/v1/devices/current/identify'
```

Request types and timeouts

All commands are simple requests / responses. The responses are provided in a synchronous way. The processing before the response is sent is allowed to take up to 500 ms on the device, so the client application should allow for additional network related delay before triggering the timeout. An additional delay of at least 500 ms (for a total timeout of 1000 ms) is recommended.

There are two types of requests:

- Queries, such as getting a value or a set of values. They use the GET HTTP method. They are guaranteed to have no impact on the state of the controlled devices.
- Commands, such as setting a value or triggering an action. They use the POST HTTP method. They may have an impact on the state of the controlled devices.

General format and future compatibility considerations

GET requests

In queries, the request bodies must be empty. In other words, there are no parameters for queries (indeed, they are part of the request URL).

The response bodies are valid JSON objects, encoded in UTF-8.

The response may contain additional undocumented fields. They are not part of the officially supported API and can evolve without notice. The undocumented fields must be ignored by the client application.

POST requests

POST requests must have the HTTP header `"Content-Type: application/json"`.

In commands, the request bodies must be valid JSON objects, encoded in UTF-8. For commands with no parameters, the request body can be either empty or contain an empty JSON object ({}).

The request bodies must not contain any undocumented parameters. If they do, the device behaviour is undefined.

However, the request bodies may contain documented parameters from the future revisions of the API. Devices incorporating older revisions of the API are guaranteed to ignore them.

Similarly, performing a request on an undocumented endpoint will result in undefined behaviour, but the requests sent to the documented endpoints from the future revisions of the API are guaranteed to result in the "404 Not Found" HTTP status code on devices incorporating older revisions of the API (unless an older revision of the API already supported the endpoint in question, albeit in an undocumented way). Note: this only applies to the endpoints under `/ipcontrol` path.

Unless an error is encountered (see the corresponding section), the response bodies for POST requests contain empty JSON objects.

Common parameters and fields

In the following documentation, if a value or a setting can be both queried and set, the corresponding POST and GET requests are documented together. In this case, they will have the same request parameters and response fields, respectively.

If in some cases certain fields are only available in the responses to the GET request, it is documented explicitly.

All documented GET response fields can be present in the POST request as well. However, if they are marked as "`(GET only)`" or listed in the "GET response" section only, they will be ignored by the device.

Subscriptions

Asynchronous access to the information (notifications) is not available yet.

Devices, systems, and groups

Devices are physically separate Devialet products that are connected to the local network. It includes individual speakers and accessories, such as Arch and Dialog.

Systems are the sets of one or more speakers that always share the playback state. They are configured in the Devialet companion app and are typically not reconfigured unless some of the speakers are physically moved to another location(s). Currently, there exist two types of systems:

- solo systems (one speaker),
- stereo systems (two speakers).

At any given moment, all speakers belong to exactly one system each.

Different audio sources may be hosted on different devices of the system. For example, Bluetooth audio source may be hosted on the left speaker, and Spotify Connect audio source on the right speaker. Physical input audio sources, if any, are hosted on the

corresponding devices (for example, “Optical - Left” audio source on the left speaker, and “Optical - Right” audio source on the right speaker).

Moreover, system settings are hosted by a single device (belonging to the system) called “system leader”. Reading/querying system settings can be done by accessing any device in the system even in the absence of the system leader, but changing a system setting requires the system leader to be reachable.

The selection of the source hosts and the system leader is done automatically by the embedded firmware and can not be modified.

Groups are the sets of one or more systems that play the same content according to the current multi-room configuration. They can be configured on-the-fly in the Devialet companion app.

For some features, there is a notion of a “group master” which is a device belonging to the group. The selection of the group master is done automatically by the embedded firmware and can not be modified.

At any given moment, all speakers belong to exactly one system each, and all systems belong to exactly one group each.

The accessories such as Arch or Dialog are not part of any system or group. However, audio sources they host are visible by all groups.

All Devialet devices in a given local network are collectively referred to as an installation.

The dispatcher

The device that receives the API command from the client application is called the dispatcher. In many cases, it forwards the API command to other devices in the installation.

Volume

All devices in the same system share the same volume. Indeed, one can only create systems of the devices of the same family and power rating.

Different systems in the same group, while playing the same content, may have different volumes. The group volume is an aggregate of individual system volumes. When setting the volume on one system, the volumes of other systems are not modified, but the group volume level may change. When setting the volume of a group, the volumes of all systems comprising this group may change simultaneously. The only exception is the case where a system has zero volume, in which case changing the group volume will have no impact on the volume of this system.

All volume commands unmute the current source. On the other hand, they do not change the `playingState`.

Requests in /devices namespace

About devices

The URLs in /devices namespace must have the following form:

```
/devices/{deviceId}/PATH/TO/ENDPOINT
```

The commands in `/devices` namespace are applied to a single device only. Similarly, the queries in `/devices` namespace represent the state of a single device only.

The only supported `{deviceId}` today is `"current"`. `"current"` device refers to the dispatcher.

General information

/devices/{deviceId}

GET, NOTIFICATION

Query general information pertaining to the designated device.

Minimal version:

- DOS >= 2.14

GET response:

```
{
  deviceId: <String>,
  systemId: <String> (speakers only),
  groupId: <String> (speakers only),
  model: <String>,
  release:
  {
    version: <String>
  },
  serial: <String>,
  role: <String> (speakers only),
  deviceName: <String>
}
```

`deviceId` is a string containing a unique device identifier in UUIDv4 format.

`systemId` is a string containing a unique system identifier in UUIDv4 format. This is the system the designated device belongs to.

Note: `systemId` is only present for speakers. Accessories such as Arch or Dialog do not belong to any system.

`groupId` is a string containing a unique group identifier in UUIDv4 format. This is the group the designated device belongs to (via its system).

Note: `groupId` is only present for speakers. Accessories such as Arch or Dialog do not belong to any group.

`model` is a human-readable string representing the device's model name. Note: even though the model name language is English, some non-ASCII characters may be present (for example, "Phantom I Opéra de Paris").

`release.version` is a human-readable string representing the device's current firmware version. Note: Although it is usually in the form of "MAJOR.MINOR.PATCH", such as "2.14.0", for example, it can contain additional non-numerical information as well. It should only contain ASCII characters.

`serial` is a string representing the device's serial number. It is usually 13 characters long, however, this may evolve in the future. It should only contain ASCII characters.

`role` is a string representing the device's role in the system.

It is only present for speakers. Possible values are: `"FrontLeft"`, `"FrontRight"`, and `"Mono"`.

`deviceName` is a human-readable non-empty string in UTF-8 format. Typically, it describes the physical location of the speaker system, such as `"Dining room"` or `"Kitchen"`, or the source device of an accessory, such as `"CD player"`.

GET example for an accessory:

```
{
  "deviceId": "f42cf307-f5bb-4311-a917-1e06d404f595",
  "model": "Arch",
  "release":
  {
    "version": "2.14.2"
  },
  "serial": "P35V12345UX02",
  "deviceName": "💿 CD Player"
}
```

GET example for a speaker:

```
{
  "deviceId": "5b35aa24-e4c9-4942-a501-7b0cf5c1e892",
  "systemId": "44a53d02-c69f-4a01-a0ce-1b6588b1d5b1",
  "groupId": "0e985d77-8212-4b48-842b-9e102d52887e",
  "model": "Phantom II 98 dB",
  "release":
  {
    "version": "2.14.2"
  },
  "role": "Mono",
  "serial": "P35V12345TQ9A",
  "deviceName": "Kitchen"
}
```


Device manipulation

/devices/{deviceId}/powerOff

POST

Turn off the device.

Minimal version:

- DOS >= 2.16

Parameters:
none

Note: Exiting OFF mode is only possible by pressing a physical button on the device.

/devices/{deviceId}/restart

POST

Reboot the device.

Minimal version:

- DOS >= 2.16

Parameters:
none

The device will become unresponsive during the reboot.

/devices/{deviceId}/resetToFactorySettings

POST

Reset the device to factory settings and reboot the device.

Minimal version:
- DOS >= 2.16

Parameters:
none

The device will become unresponsive during the reboot.

Beware, this will also erase all network credentials. Unless the device is connected to the local network by an Ethernet cable, it will become inaccessible.

Note: this will not roll back the firmware version.

Requests in /systems namespace

About systems

The commands in `/systems` namespace are applied to all accessible devices that are part of the designated system.

If some devices are switched off or inaccessible for another reason (network issues etc.), several scenarios are possible:

- If the command requires a system leader and the leader is...
  - ...present, the command is carried out correctly. It will be applied to other devices as soon as the other device(s) become accessible again.
  - ...absent, the `"SystemLeaderAbsent"` error code is reported and the system state is not modified.
- If the command requires the current or a designated source and the device hosting that source is…
  - …present, the command is carried out correctly. It will be applied to other devices as soon as the other device(s) become accessible again.
  - ...absent, the `"UnreachableSource"` error code is reported and the system state is not modified.
- In other cases, the command is carried out on accessible devices only, except the dispatcher, and an `"UnreachableDevices"` error code is reported. The state of the devices other than the dispatcher is undefined and must be re-queried.

The queries in `/systems` namespace represent the state of the system as a whole.

The URLs in `/systems` namespace must have the following form:

`/systems/{systemId}/PATH/TO/ENDPOINT`

The only supported `{systemId}` today is `"current"`. In this case, the request will apply to the system of the dispatcher.

General information

/systems/{systemId}

GET, NOTIFICATION

Query general information pertaining to the designated system.

Note: for (non-speaker) accessories such as Dialog or Arch, one should use `/devices/{deviceId}` instead.

Minimal version:

- DOS >= 2.14
- DOS >= 2.16 (certain fields)

GET response:

```
{
  systemId: <String>,
  groupId: <String>,
  systemName: <String>,
  availableFeatures: [DOS >= 2.16]
  [
    <String>, ...
  ]
}
```

`systemId` is a string containing a unique system identifier in UUIDv4 format.

`groupId` is a string containing a unique group identifier in UUIDv4 format. This is the group the designated system belongs to.

`systemName` is a human-readable non-empty string in UTF-8 format. Typically, it describes the physical location of the speaker system, such as `"Dining room"` or `"Kitchen"`.

`availableFeatures` [DOS >= 2.16] is an array containing strings that identify various features available on this system. Possible string values are: `"equalizer"`, `"nightMode"`.

GET example:

```
{
  "systemId": "13531594-b1c1-42c7-8d5a-18fa9e5d7cd4",
  "groupId": "0e985d77-8212-4b48-842b-9e102d52887e",
```

```
  "systemName": "Dining room 🍴",
  "availableFeatures":
  [
    "equalizer",
    "nightMode"
  ]
}
```

Sound control

/systems/{systemId}/sources/current/soundControl/volume

GET, POST, NOTIFICATION

Query or set the current volume for the designated system.

Minimal version:

- DOS >= 2.14

Parameters / fields:

```
{
  volume: <Integer>
}
```

`volume` is the current volume of the current system, in percent (0-100).

See “Volume” section for additional details.

Example:

```
{
  "volume": 35
}
```

/systems/{systemId}/sources/current/soundControl/volumeUp

POST

Increase the current volume for the designated system.

Minimal version:

- DOS >= 2.14

Parameters:
none

The volume is increased by one step. The step value is 5% of the volume range and it is not configurable. If the current volume is closer to 100% than the step value, then the current volume is set to 100%. If the current volume is already 100%, the request succeeds.

See “Volume” section for additional details.

/systems/{systemId}/sources/current/soundControl/volumeDown

POST

Decrease the current volume for the designated system.

Minimal version:

- DOS >= 2.14

Parameters:
none

The volume is decreased by one step. The step value is 5% of the volume range and it is not configurable. If the current volume is closer to 0% than the step value, then the current volume is set to 0%. If the current volume is already 0%, the request succeeds.

See “Volume” section for additional details.

Audio settings

/systems/{systemId}/settings/audio/equalizer

GET, POST, NOTIFICATION

Query or set equalizer gains/preset for the designated system.

Minimal version:

- DOS >= 2.16

Parameters / fields:

```
{
  enabled: <Boolean>, (GET only)
  preset: <String>,
  currentEqualization: (GET only)
  {
    BAND_LABEL:
    {
      frequency: <Integer> (optional),
      gain: <Number>
    }, ...
```

```
  },
  customEqualization: (optional for POST)
  {
    BAND_LABEL:
    {
      gain: <Number>
    }, ...
  },
  gainRange: (GET only)
  {
    min: <Number>,
    max: <Number>,
    stepPrecision: <Number>
  },
  availablePresets: (GET only)
  [
    <String>, ...
  ]
}
```

`enabled` indicates if the equalizer is active. It can become inactive in certain special audio processing modes of the product. When it is inactive, the client application may manipulate all the settings as usual, but there will be no audible impact. However, for better user experience, it is usually recommended to disable equalizer controls in this case. This field is read-only (the equalizer can not be re-enabled via this endpoint).

`preset` is the selection of one of the available presets. Possible values are `"flat"`, `"custom"`, and `"voice"`.

`currentEqualization` provides gains and other information for all bands for the currently used equalizer preset. Different systems may have different *`BAND_LABEL`* values. *`BAND_LABEL`* examples include: "low", "high".

`currentEqualization.frequency` provides a user-facing "central frequency" (it actually may be rounded) for the given band, in hertz. This value may be absent.

`currentEqualization.gain` provides a positive or negative gain for the given band, in steps (may be fractional).

`customEqualization` provides gains and other information for all bands for the `"custom"` equalizer preset. The *`BAND_LABEL`* values are the same as in `currentEqualization` above.

`customEqualization.gain` provides a positive or negative gain for the given band, in steps (may be fractional). For POST requests, it will be rounded to the nearest authorized value by the device (in this case, the value read back by the client application may be

slightly different). If the value is out of authorized bounds, `"InvalidValue"` error code is reported.

Although these gains only have an effect when the `"custom"` preset is selected, it is possible to change their values at any time. The stored custom gain values can be modified independently from the preset. I.e., one can modify custom gain values while using a different (non-`"custom"`) preset. Also, modifying just a preset does not trigger any changes to the stored custom gain values.

The gains may be provided either simultaneously for all or for some bands (in which case the device guarantees to apply the new settings for all provided bands at the same moment) or one by one.

Both `preset` and `customEqualization` are present in the response. However, `customEqualization` can be omitted in the POST request, in which case the values of the gains corresponding to the `"custom"` preset will not be modified.

If provided equalizer parameters already correspond to the current state, the call will succeed.

`gainRange.min` provides the minimal authorized value for the gain when using the `"custom"` preset, in steps. It applies to all bands.

`gainRange.max` provides the maximal authorized value for the gain when using the `"custom"` preset, in steps. It applies to all bands.

`gainRange.stepPrecision` provides the step precision, in arbitrary units. This value determines how the gain rounding operates: the gain values are rounded to the nearest positive or negative multiple of the step precision. This value can be used by the client application to adapt the UI so as to match the device gain precision as closely as possible. Typical values are `0.1`, `0.25`, `0.5`, and `1`.

`availablePresets` provides a full list of available equalizer presets on the designated system.

For POST requests, if the system leader is absent, the `"SystemLeaderAbsent"` error code is reported.

GET example:

```
{
  "preset": "flat",
  "currentEqualization":
  {
    "low":
    {
      "frequency": 400,
```

```
      "gain": 0
    },
    "high":
    {
      "frequency": 2000,
      "gain": 0
    }
  },
  "customEqualization":
  {
    "low":
    {
      "gain": -0.5
    },
    "high":
    {
      "gain": 2.25
    }
  },
  "gainRange":
  {
    "min": -6,
    "max": 6,
    "stepPrecision": 1
  },
  "availablePresets":
  [
    "flat",
    "custom",
    "voice"
  ]
}
```

POST example 1:

```
{
  "preset": "custom",
  "customEqualization":
  {
    "low":
    {
      "gain": 3.0
    },
    "high":
    {
      "gain": 3.0
    }
```

```
  }
}
```

POST example 2:

```
{
  "preset": "flat"
}
```

/systems/{systemId}/settings/audio/nightMode

GET, POST, NOTIFICATION

Query or set the night mode audio rendering for the designated system.

Minimal version:

- DOS >= 2.16

Parameters / fields:

```
{
  nightMode: <String>
}
```

`nightMode` is the night mode selection. Possible values are `"on"` and `"off"`. If the selection already corresponds to the current state, the call will succeed.

For POST requests, if the system leader is absent, the `"SystemLeaderAbsent"` error code is reported.

Example:

```
{
  "nightMode": "on"
}
```

Other operations

/systems/{systemId}/bluetooth/startAdvertising

POST

Start bluetooth advertising.

Minimal version:

- DOS >= 2.16

Parameters:

none

While advertising, the system can be discovered and paired with by bluetooth devices trying to pair with a speaker.

The advertising will automatically turn off after 1 minute or upon a successful BT connection.

If this command is issued while advertising is ongoing, the timeout will be set to 1 minute after the moment of the latest call.

If the system contains several devices, and the dispatcher is not the device elected to perform the advertising, this endpoint can report the `"UnreachableDevice"` error code in the case the advertising device can not be reached by the dispatcher.

Device manipulation

/systems/{systemId}/powerOff

POST

Turn off all the devices of the designated system.

Minimal version:

- DOS >= 2.16

Parameters:
none

Note: Exiting OFF mode is only possible by pressing a physical button on each device.

/systems/{systemId}/restart

POST

Reboot all the devices of the designated system.

Minimal version:

- DOS >= 2.16

Parameters:
none

The devices will become unresponsive during the reboot.

/systems/{systemId}/resetToFactorySettings

POST

Reset to factory settings and reboot all the devices of the designated system.

Minimal version:
- DOS >= 2.16

Parameters:
none

The devices will become unresponsive during the reboot.

Beware, this will also erase all network credentials. Unless the devices are connected to the local network by an Ethernet cable, they will become inaccessible.

Note: this will not roll back the firmware version.

Requests in /groups namespace

About groups

The commands in `/groups` namespace are applied to all accessible devices that are part of the systems comprising the designated group.

The URLs in `/groups` namespace must have the following form:

```
/groups/{groupId}/PATH/TO/ENDPOINT
```

The only supported `{groupId}` today is `"current"`. In this case, the request will apply to the group of the dispatcher (i.e. the device that receives the request).

General information

Source types and stream sensing

The following source types exist:

- Physical sources
  - `"phono"` (Arch only)
  - `"line"` (Arch only)
  - `"digital_left"` (Arch only)
  - `"digital_right"` (Arch only)
  - `"optical"` (Phantom I, Dialog)
  - `"opticaljack"` (Phantom II only)
- Non-physical sources
  - `"spotifyconnect"`

    - `"airplay2"`
    - `"bluetooth"`
    - `"upnp"`
    - `"raat"`

Moreover, there are two subcategories of physical sources:

- Physical sources that do not allow the detection of the stream.
- Physical sources that allow the detection ("sensing") of the stream:
    - `"digital_left"`
    - `"digital_right"`
    - `"optical"` (including Dialog)
    - `"opticaljack"`

Note 1: despite `"digital_left"` and `"digital_right"` being physically separate connectors, it is impossible to perform the sensing on the input which is not currently configured as the active input of Arch.

All non-physical sources, as well as the physical sources that allow the stream sensing support the autoswitch feature (unless it is disabled from the Devialet companion app).

/groups/{groupId}/sources

GET, NOTIFICATION

Query the list of currently available sources for the designated group.

Minimal version:

- DOS >= 2.14

Request parameters:
none

Response:

```
{
  sources:
  [
    {
      sourceId: <String>,
      deviceId: <String>,
      type: <String>
    }, ...
  ]
}
```

- `sourceId` is a string containing a unique identifier of the source in UUIDv4 format.

- `deviceId` is a string containing a unique identifier of the host device in UUIDv4 format. The host may be either a speaker device, or a (non-speaker) accessory such as Dialog or Arch.

- `type` is one of the predefined source types (see above). For speaker stereo pairs, use `deviceId` to differentiate between the inputs of the left and the right speakers.

The list can change if the group is reconfigured (via Devialet companion app) or if certain devices become turned off, disconnected from the network or otherwise unreachable. It can also change if Arch's physical input configuration is changed (see below).

Physical sources that do not allow the detection of the stream are always present. Physical sources that allow the detection ("sensing") of the stream (see the "Stream sensing" chapter) are also always present, whether there is a stream or not.

For each Arch, there are four possible situations, depending on its current configuration of the physical inputs:

- Only the `"phono"` source is available
- Only the `"line"` source is available
- Only the `"digital_left"` source is available
- Only the `"digital_right"` source is available

To change Arch configuration, one should use the Devialet companion app.

All accessories are available for all groups all the time. However, an accessory can only play in one group at a given moment of time.

/groups/{groupId}/sources/current

GET, NOTIFICATION

Query the current state of the playback pertaining to the designated group, as well as the current capabilities of the designated source.

Minimal version:

- DOS >= 2.14

Parameters:

```
{
  source:
  {
    sourceId: <String>,
    deviceId: <String>,
    type: <String>
```

```
  },
  playingState: <String>,
  muteState: <String>,
  metadata: (optional)
  {
    artist: <String>,
    album: <String>,
    title: <String>,
    coverArtUrl: <String> (optional)
  },
  availableOperations:
  [
    <String>, ...
  ]
}
```

`source` object describes the current source. If there is no current source, this parameter is absent. When present, it contains the following fields:

- `source.sourceId` is a string containing a unique identifier of the source in UUIDv4 format.
- `source.deviceId` is a string containing a unique identifier of the host device in UUIDv4 format. The host may be either a speaker device or a (non-speaker) accessory such as Dialog or Arch.
- `source.type` is one of the predefined source types (see above). For speaker stereo pairs, use `source.deviceId` to differentiate between the inputs of the left and the right speakers.

`playingState` is one of the following values: `"playing"`, `"paused"`.

`muteState` is one of the following values: `"muted"`, `"unmuted"`.

`metadata` object describes the current track. If the current source does not provide metadata, the object may be absent. For sources that provide `metadata` while playing, the metadata will be kept (and reported via this endpoint) even when the source is paused. When present, it contains the following sub-parameters (some of which may be empty strings):

- `metadata.artist` provides the artist name. It may be empty if the data are not available, but the field is always present.
- `metadata.album` provides the album name. It may be empty if the data are not available, but the field is always present.

- `metadata.title` provides the track name. It may be empty if the data are not available, but the field is always present.

- `metadata.coverArtUrl` provides an URL that can be used by the client application to download the image of the cover art. This field may be absent if the data are not available as an URL.

  Reminder: All names are encoded in UTF-8.

`availableOperations` is an array of strings providing the list of currently available operations on the current source. The possible values are:

- `"play"`, `"pause"`, `"next"`, `"previous"`, `"seek"`

Note: `"mute"` and `"unmute"` operations are always available. They are not present in this list.

Note: the list of available operations can change even if the source is not changed. For example, `"next"` command may become unavailable when the last song in the playlist is played or when the next operation is not allowed (for example, free Spotify accounts).

Example:

```
{
  "source":
  {
    "sourceId": "213a3ed0-1fb9-4da2-bcf4-066da0f7b27e",
    "deviceId": "13531594-b1c1-42c7-8d5a-18fa9e5d7cd4",
    "type": "spotifyconnect"
  },
  "playingState": "playing",
  "muteState": "unmuted",
  "metadata":
  {
    "artist": "Michael Jackson",
    "album": "Thriller",
    "track": "Billie Jean",
    "coverArtUrl": "https://cdn.spotify.com/covers/4729028427.png"
  },
  "availableOperations":
  [
    "play",
    "pause",
    "seek"
  ]
}
```

Playback

About playback commands and states

The mute/unmute commands and the corresponding playback states (muted/unmuted) are independent from the play/pause commands and their states (playing/paused). All four combinations are possible. Modifying one state does not modify the other one (i.e., play/pause commands have no impact on the muted/unmuted state and vice versa).

On the other hand, all volume related commands unmute the current source (but they have no impact on the playing/paused state).

/groups/{groupId}/sources/{sourceId}/playback/play

POST

Start or resume playback of the designated source on the system and change its `playingState` to `"playing"`.

Minimal version:

- DOS >= 2.14

Parameters:
none

If the designated source is not the current source of the group, it will be selected first. The previous current source (if any) will change its `playingState` to `"paused"` (if it is not already the case) and will be unselected.

It should be possible to switch to and resume the playback ("pull the playback") for most sources. In other words, all paused sources maintain their context even while another source is playing. Of course, certain sources also support resuming the playback via an external action ("push the playback" + autoswitch).

For sources hosted by an accessory, this call will also have an effect of setting the current target for the accessory in question to the dispatcher's system (and, by extension, to the group the said system currently belongs to).

If the group's current source is already playing and it matches the designated source, the call will succeed.

Important note: this call may have asynchronous effects, depending on the source. Even when this call reports success, the client application should use the output of the `/groups/{groupId}/sources/current` in general and `playingState` in particular in order to update its UI state. If the "delayed" `play` command fails, there will be no specific

notification. On the other hand, additional `play` commands are accepted (for the same or for a different source) even if there still is an "ongoing" play command in the group.

If the play request fails, `"PlaybackNoStream"` error is reported. This can happen, for example:

- If there is no cable on an input with cable sensing (`"opticaljack"`).
- If there is no lock on a digital input (`"digital_left"`, `"digital_right"`, `"optical"`. Does not apply to `"opticaljack"` (Phantom II)).
- If the audio session has been interrupted and could not be re-established (torn down AirPlay session or unreachable AirPlay source, unreachable Bluetooth source, unreachable UPnP server, unreachable Roon Ready controller, expired, revoked or otherwise invalidated Spotify Connect token or unreachable Spotify server).
- If the detected format on a digital input is not supported.

If this happens, the designated source will still become / remain current (with no sound).

Warning: Because of an independent implementation of multi-room grouping by Apple AirPlay 2 and Roon Ready, selecting an AirPlay2 or Roon Ready source will automatically exclude the source's host system from its current group. If the system is alone in its group, it will create a new group (and change its groupId), and do so on every new audio session. If the group master was in the system in question, the group is destroyed completely (all its accessible constituent systems will move to its own separate groups).

/groups/{groupId}/sources/current/playback/pause

POST

Pause playback of the current source on the system and change its `playingState` to `"paused"`, or mute the current source if pausing is not supported semantically.

Minimal version:

- DOS >= 2.14

Parameters:
none

If the system is already paused, the call will succeed.

All sources support the "Pause" command.

Note: although certain sources can not semantically support the "Pause" command (example: `"optical"`), they will accept the "Pause" command and mute the sound output instead. Note that in this case, the `playingState` will remain `"playing"`, and the `muteState` will be changed to `"muted"`.

/groups/{groupId}/sources/current/playback/mute

POST

Mute the playback of the current source on the system and change its `muteState` to `"muted"`.

Minimal version:

- DOS >= 2.14

Parameters:
none

If the system is already muted, the call will succeed.

/groups/{groupId}/sources/current/playback/unmute

POST

Unmute the playback of the current source on the system and change its `muteState` to `"unmuted"`.

Minimal version:

- DOS >= 2.14

Parameters:
none

If the system is already unmuted, the call will succeed.

/groups/{groupId}/sources/current/playback/next

POST

Start playback of the next track.

Minimal version:

- DOS >= 2.14

Parameters:
none

If the current source does not have the “Next” command, `"PlaybackOperationNotAvailable"` error code is reported.

/groups/{groupId}/sources/current/playback/previous

POST

Start playback of the previous track.

Minimal version:

- DOS >= 2.14

Parameters:
none

If the current source does not have the "Previous" command, `"PlaybackOperationNotAvailable"` error code is reported.

Some client applications may want to implement a special "restart current track" logic when the user presses the "Previous" button in the UI while being in the middle of the track. Such higher level logic is not supported by this endpoint and must be implemented in the client application.

Sample implementation

These instructions apply to the latest version of the IP Control. For availability of individual endpoints or fields, please refer to the corresponding sections.

The client application may perform the following steps to obtain a full state of the devices in the installation:

1. Perform network discovery. Construct a list of all visible Devialet devices and their IP addresses.
2. Iterate over all discovered devices:
    a. Perform `/devices/current` call on each device.
    b. For each device, memorize its system and group ids, and (optionally) their firmware version and serial numbers.
    c. For accessories only, memorize their names.
3. Construct a list of discovered systems and groups.
4. Iterate over all discovered systems:
    a. Perform `/systems/current` call on one device from each system.
    b. For each system, memorize its name such as "Dining Room" etc.
5. Iterate over all discovered groups:
    a. Perform `/groups/current/sources` call on one device from each group. All available sources for this group will be listed.
    b. Perform `/groups/current/sources/current` call on one device from each group. The current playback state will be received.
6. On groups or systems of interest:

a. Perform `/{groups,systems}/current/sources/current/soundControl/volume` call to get the current volume.

After that, the client application can present a full installation state to the user. In order to control various devices, the following operations can be performed:

- Playback commands on a group
- Sound control (volume) commands on a system
- [DOS >= 2.16] Audio setting (EQ, night mode) commands on a system

In order to show up to date information, the client application may subscribe to or perform regular polling on the following endpoints:

- `/groups/current/sources` (the list of all available sources)
- `/groups/current/sources/current` (the current playback state)
- `/systems/current/sources/current/soundControl/volume` (the current volume for the system)
- `/groups/current/sources/current/playback/position` (the position within the current track)

Error handling

Code 200 errors (regular errors)

In case of a regular error, the response will have the "200 OK" status code. The response body will be a JSON:

```
{
  error:
  {
    code: <String>,
    details: <Object> (optional),
    message: <String> (optional)
  }
}
```

`error.code` is one of the predefined error identifiers. If the client application encounters an unknown code (this may happen when the firmware is updated, but not the application), it must process it and show a generic error message.

`error.details`'s content, if present, depends on the error identifier. Typically, these are additional structured data that can be used for formatting a user-facing error message in any language supported by the front end application.

`error.message`, if present, is provided for debug purposes only. It is not recommended to show it to the user or process it in any way other than logging for debugging purposes.

The common errors are:

- `"Error"` is a generic error with no available details.
- `"UnreachableDevices"` for all requests that require communication with other devices. Depending on the request, it can be either completely abandoned, or carried out partially (on certain devices only).
- `"Timeout"` if a request could not be carried out in the allotted time. The resulting state is undefined and should be re-queried.
- If a request starting by `/groups/{groupId}/sources/current` or `/systems/{systemId}/sources/current` is performed, but there is no current source, the `"NoCurrentSource"` error code is reported.
- When setting a value, if the provided value has incorrect type, format, is not part of the predefined list of allowed values, or is out of range, `"InvalidValue"` error code is reported, unless there is an explicitly defined error code (e.g., too-long strings). Note: when providing a fractional number when an integer is expected, the device will round the value to the nearest integer and will not report an error.

Code 500 errors

In case of an exception (unexpected error), the response will have the "500 Internal Server Error" status code. The message body, if present, will be a JSON following the same format as the one for the "200 OK" status code presented above. However, its content is not part of the officially supported API and can evolve without notice.

Such errors should not occur on a correctly functioning device.

Code 400 errors

In case of a malformed JSON request, the response will have the "400 Bad Request" status code. The message body will be empty.

Such errors should not occur if the client application is implemented correctly.

Code 404 errors

In case of a non-existing endpoint, the response will have the "404 Not Found" status code. The message body will be empty.

Such errors should not occur if the client application is implemented correctly.

Note: incorrect `{sourceId}` value will result in a code 200 error, see above.

Examples:

- A request starting by `/systems/current` is performed on an accessory
- A request starting by `/groups/current` is performed on an accessory

Code 415 errors

The "415 Unsupported Media Type" status code will be issued if an invalid `"Content-Type"` header value is provided in the POST request. The only allowed value is `"application/json"`. The same error will occur if this header is omitted.

Such errors should not occur if the client application is implemented correctly.

Errors with other codes

Any other status codes must be interpreted as an unexpected internal error, similar to the "500 Internal Server Error" status code. They are not part of the officially supported API and can evolve without notice.

Such errors should not occur on a correctly functioning device.

Known issues

DOS 2.14

The following issues have been identified in DOS 2.14.x.

- `availableOperations` property of `/groups/{groupId}/sources/current` is not correct for the UPnP source (listing "Previous" and "Next" operations when they may not be actually available). Moreover, the "Previous" and "Next" operations for the UPnP source in `/groups/{groupId}/sources/current/playback` do not work as expected.
- `release.version` property of `/devices/{deviceId}` is incorrect.
- When a POST uses an invalid numerical value, the `"InvalidValue"` error code is not implemented.

These issues are fixed in DOS 2.16.x.

Document history

Revision 1 - December 2021

The initial version, covering DOS 2.14 and upcoming DOS 2.16.