

Dante Managed API

User Guide



Document version: 1.8

Document name: AUD-MAN_Managed_API_v1.8

Published: Wednesday, April 2, 2025

Copyright

© 2025 Audinate Pty Ltd. All Rights Reserved.

Audinate®, the Audinate logo and Dante® are registered trademarks of Audinate Pty Ltd.

All other trademarks are the property of their respective owners.

Audinate products are protected by one or more of US Patents 7747725, 8005939, 7978696, 8171152, European Patent 2255541, Chinese Patent ZL200780026677.0, and other patents pending or issued. See

www.audinate.com/patents.

Legal Notice and Disclaimer

Audinate retains ownership of all intellectual property in this document.

The information and materials presented in this document are provided as an information source only. While effort has been made to ensure the accuracy and completeness of the information, no guarantee is given nor responsibility taken by Audinate for errors or omissions in the data.

Audinate is not liable for any loss or damage that may be suffered or incurred in any way as a result of acting on information in this document. The information is provided solely on the basis that readers will be responsible for making their own assessment, and are advised to verify all relevant representation, statements and information with their own professional advisers.

Software Licensing Notice

Audinate distributes products which are covered by Audinate license agreements and third-party license agreements.

For further information and to access copies of each of these licenses, please visit our website:

www.audinate.com/software-licensing-notice

Contacts

Audinate Pty Ltd

Level 7, 64 Kippax Street
Surry Hills NSW 2010
Australia
Tel. +61 2 8090 1000
info@audinate.com
www.audinate.com

European Office

Audinate Ltd
Future Business Centre
Kings Hedges Rd
Cambridge CB4 2HY
United Kingdom
Tel. +44 (0) 1273 921695

Audinate Inc

4380 S Macadam Avenue
Suite 255
Portland, OR 97239
USA
Tel: +1.503.224.2998
Fax. +1.503.360.1155

Asia Pacific Office

Audinate Limited
Suite 1106-08, 11/F Tai Yau Building
No 181 Johnston Road
Wanchai, Hong Kong
澳迪耐特有限公司
香港灣仔莊士敦道181號
大有大廈11樓1106-8室
Tel. +(852)-3588 0030
+(852)-3588 0031
Fax. +(852)-2975 8042

Contents

Copyright	1
About Audinate	4
About Dante	4
One Connection. Endless Possibilities.	4
What is the Dante Managed API?	5
Accessing the API via Dante Domain Manager	6
Enabling API Access	6
API Keys	6
Enabling API Key Management	6
API Key Creation	6
Limitations of the API from DDM 1.5	8
Legacy Devices Not Supported	8
Virtual Devices Not Recognized	8
AES67 and SMPTE2110 Channel Subscriptions Not Supported	8
Accessing the API via Dante Director	9
API Keys	9
API Key Generation	9
API Usage	12
Setting up a Dante Managed API Postman Collection	12
To Use the Shared Workspace	12
To Use a Downloaded Collection	13
Performing Your First Request	14
Requests via cURL	15
Getting Device and Subscription Status	16
Getting Device and Subscription Status via cURL	17
Setting (or Clearing) a Channel Subscription	18
Clearing a Subscription	20
Setting a Channel Subscription via cURL	20
Appendix 1 - DDM GraphQL Schema Reference	22
DDM 1.8 GraphQL Schema Reference	22
DDM 1.7 GraphQL Schema Reference	37
DDM 1.5 GraphQL Schema Reference	51
Index	61

About Audinate

Audinate® is the leading provider of professional AV networking technologies globally. Audinate's Dante platform distributes digital audio and video signals over computer networks, and is designed to bring the benefits of IT networking to the professional AV industry. AV-over-IP (AVoIP) using Dante-enabled products ensures interoperability between AV devices and allows end users to enjoy high quality, flexible solutions – typically with a lower total cost of ownership.

About Dante

One Connection. Endless Possibilities.

Dante replaces all audio and video connections with a computer network, effortlessly sending video or hundreds of channels of audio over slender Ethernet cables with perfect digital fidelity.

Adopted by hundreds of manufacturers in thousands of products, Dante is the de facto standard for modern AV connectivity.

For more information, please visit the Audinate website at getdante.com.

What is the Dante Managed API?

Dante Managed API provides a way to programmatically interact with your Dante devices when enrolled in Dante Domain Manager or Dante Director. This will allow you to integrate Dante functionality into custom solutions.

Currently it is possible to query the status of a device or domain, set audio and video subscriptions for a device and enrol or unenroll devices.

Additional functionality is planned for future releases.

Accessing the API via Dante Domain Manager

The Dante Managed API was released in Dante Domain Manager v1.5.

Enabling API Access

API access can be enabled and disabled in the DDM Administration Menu by DDM system administrators. Please refer to the [DDM User Guide](#) for more information. API Access is enabled by default when upgrading from DDM v1.4 or earlier or for fresh installations.

API Keys

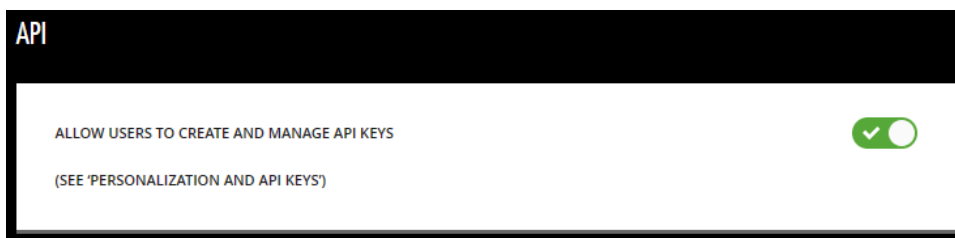
Authentication and authorization for the Dante Managed API is controlled via API keys, which can be generated in the Dante Domain Manager web UI.

API keys do not expire but they can be revoked via the Dante Domain Manager web UI.

Enabling API Key Management

Before any API keys can be created, a user with the role of Site Control must enable API Key Management:

1. Go to **Settings > Network & Security** in the Dante Domain Manager web UI.
2. Under the API heading ensure ALLOW USERS TO CREATE AND MANAGE API KEYS is selected:



This setting is disabled by default when upgrading from DDM 1.4 or earlier, or for fresh installations.

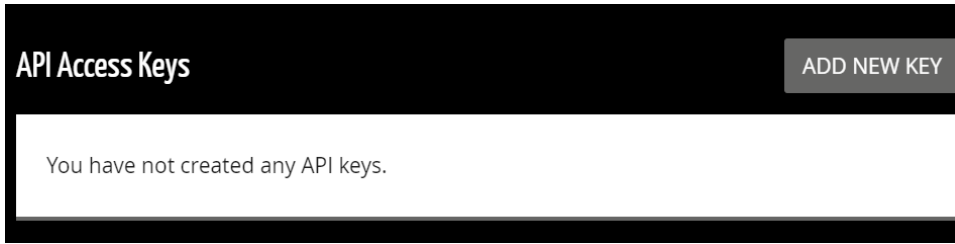
If API Access has been disabled, this setting is not available. Please contact your DDM system administrator or consult the DDM User Guide for more information.

API Key Creation

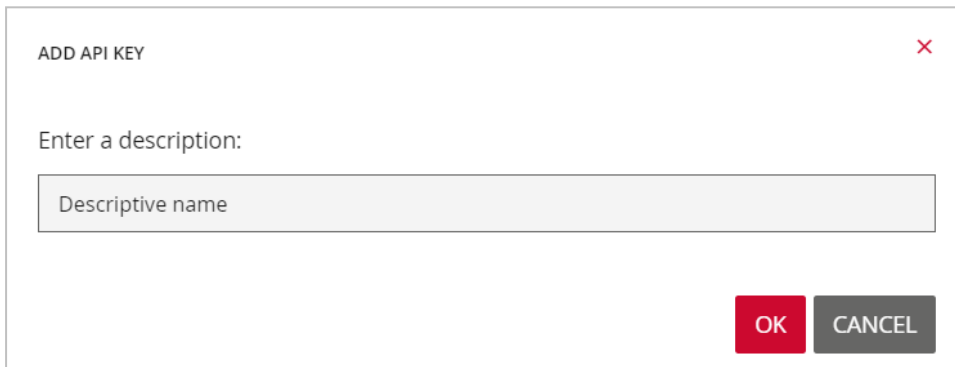
Once API Key Management is enabled, an API key can be created in the Dante Domain Manager web UI by any user. The API key will share the user's privileges. If, as an Administrator, you wish to create an API key with restricted access, create a normal user account for this purpose then log in with it to generate the API key.

To create an API key:

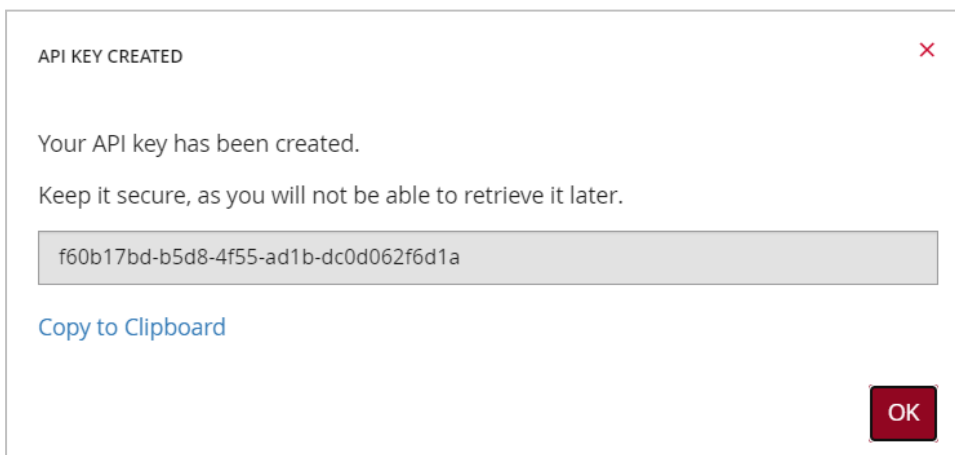
1. Go to **Settings > Personalization & API Keys** in the Dante Domain Manager web UI.
2. Next to the heading 'API Access Keys' click **ADD NEW KEY**:



3. Provide a descriptive name for the key to help identify its use later. This name will be shown in the Event Log when the API is used with this key.



4. A dialog will appear with the newly generated key. The key must be copied and retained by the user at this point, it will not be recoverable at a later time. If a key is lost, it must be removed via the web UI and a new one created. Do not share your API key with anyone as this will give them access to your Dante Domain Manager account.



5. The newly created key will appear in the API Access Keys list on the **Settings > Personalization & API Keys** page of the Dante Domain Manager web UI. This list includes the name given to the key and a partially obscured version of the key string, to aid identification and the key's permissions. There is also a button

available to remove the key.



Limitations of the API from DDM1.5

Legacy Devices Not Supported

Devices operating in 'legacy mode' (in general, Dante devices with software version prior to 4.0) are not supported by the Managed API. Whilst they will continue to work normally, they will not be shown in API queries, and they cannot be controlled via the API.

Virtual Devices Not Recognized

'Virtual devices' used for Shared Audio Groups are not yet recognized by the Managed API. Whilst they will continue to work normally, they will not be shown in API queries, and they cannot be controlled via the API.

AES67 and SMPTE2110 Channel Subscriptions Not Supported

Only native Dante flows are supported by the API at this time.

Accessing the API via Dante Director

Dante Director provides early access to part of the Dante Managed API.

API Keys

Authentication and authorization for the Dante Managed API is controlled via an API key. API keys can be generated in the Dante Director web UI. When generating a key, access restrictions for the key can also be defined.

API keys do not expire but their access restrictions can be changed, and they can be revoked via the Dante Director UI.

API Key Generation

An API key can be generated in the Dante Director web UI by any user with administrator privileges.

To create an API key:

1. Go to **Settings > API Keys** in the Dante Director web UI.
2. Click **Add Service Key**.
3. Provide a descriptive name for the key to help identify its use later. This name will be shown in the Event Log when the API is used with this key.
4. Access level can be specified as full access to all sites (domains) or restricted to select sites.

Add Service Key

Add service key with access permission to sites

Name:

Privileges
Only administrator Keys can manage users, sites and Dante Cloud settings.

Site Access:

- Full Administrator Access to All Sites
- Restricted Access...
 - Revesby
 - Wahroonga
 - Hurlstone Park
 - Surry Hills
 - Croydon


Cancel **Add Key**

5. Click **Add Key**.
6. A dialog will appear with the newly generated key. The key must be copied and retained by the user at this point, it will not be recoverable at a later time. If a key is lost, it must be deleted via the web UI and a new one created. Do not share your API key with anyone as this will give them access to your Dante Director account.

Success

Please copy your key now.

Once you press OK, it will not be recoverable and if lost, a new key will have to be created.

Key: **HFuEaUPUaXPONNWMWkqsQ** 

OK

7. The newly created key will appear in the Service API Keys list on the **Settings > API Keys** page of the Dante Director web UI. This list includes the name given to the key, a partially blanked out version of the key string to aid identification as well as the key's permissions. There are also buttons available to edit key permissions and delete the key.

API Keys

Add Service Key

API keys are used to access data through API

Service API Keys

Service API keys are not associated with a user. Will show up in logs as the key name and have permissions explicitly set.

Name ^	Key Hint	Permissions	
Descriptive Name	HFuE*****kqsQ	Revesby, Surry Hills	Edit Key Delete Key

API Usage

Once an API key has been generated, it can be used to access the Dante Managed API.

Dante Managed API implements a GraphQL interface. An introduction to GraphQL and how to get started can be found at <https://graphql.org/learn/>.

GraphQL can be accessed from many different clients. In this document we will use Postman and cURL as examples, but the same principles apply for any client-side implementation. Postman can be downloaded for free from <https://www.postman.com/>.

Setting up a Dante Managed API Postman Collection

A shared Postman collection is available to help get started with the Dante Managed API. The download link for this shared collection is available on the **Settings > Personalization & API Keys** page in the Dante Domain Manager web UI, and on the **Settings > API Keys** page in the Dante Director web UI.


In Dante Domain Manager the link will download the Postman collection to your machine. In Dante Director the link will open a shared workspace in Postman.

To Use the Shared Workspace

The link should open Postman in your browser. If you are already signed into Postman, it will be added to your Workspaces.

To start using the Dante Managed API collection, you will need to set the API key that is specific to your account.

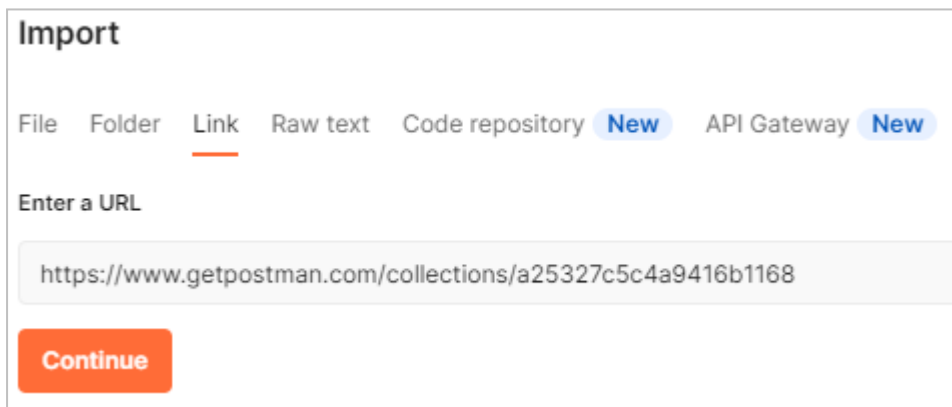
1. Click the collection name and then navigate to the **Variables** tab.
2. In the 'Variables' tab, Set the APIKEY variable 'Current Value' to the previously-generated API key.
3. Save the changes.

 **Note:** The URL should already be set for Dante Director, <https://api.director.dante.cloud/graphql>

	Variable	Initial value	Current value	...
<input checked="" type="checkbox"/>	URL	https://api.director.dante.cloud/graphql	https://api.director.dante.cloud/graphql	
<input checked="" type="checkbox"/>	APIKEY		*****	
	Add new variable			

To Use a Downloaded Collection

After opening the Postman desktop application, you will be prompted to create a new workspace. Once the workspace is created, click the **Import** button, select **File** and import the postman collection file you downloaded.



You should see a confirmation dialog indicating that you are about to import the Dante Managed API collection. Click the **Import** button.

Once the collection is imported into your workspace, you will need to set two variables that are specific to your account.

1. Click the collection name and then navigate to the **Variables** tab.
2. Set the URL variable 'Current Value' to your API URL:
 - For Dante Domain Manager, this is the base URL of your Dante Domain Manager with the path `/graphql`, e.g. `https://ddm.orgname.com/graphql`. This URL can be copied from the section 'Getting Started with the Dante Managed API' on the **Settings > Personalization & API Keys** page.
 - For Dante Director, this is `https://api.director.dante.cloud/graphql`. This URL should already be set, and can also be copied from the **Settings > API Keys** page.
3. Set the APIKEY variable 'Current Value' to the previously-generated API key.
4. Save the changes.

Performing Your First Request

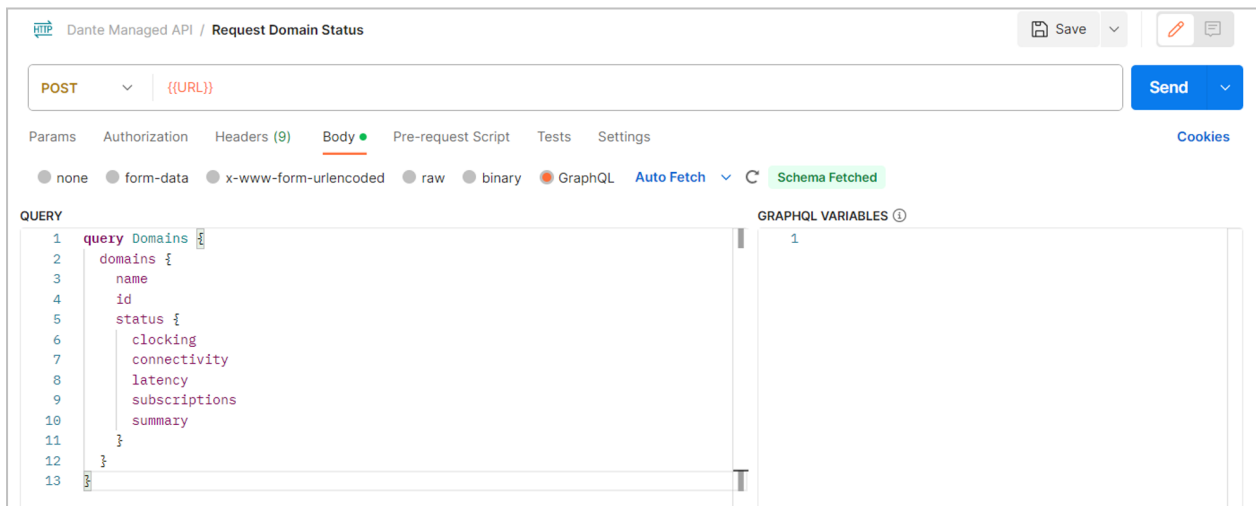
For our first request we will be obtaining the status of all accessible domains.

Domain status is obtained by the query shown below - this is the 'Request Domain Status' query from the shared Postman collection. It will get the name and collective status of all accessible domains.

```
query Domains {
  domains {
    name
    status {
      clocking
      connectivity
      latency
      subscriptions
      summary
    }
  }
}
```

 **Note:** 'Domains' are referred to as 'Sites' in the Dante Director user interface.

The resultant request in Postman should look like this:



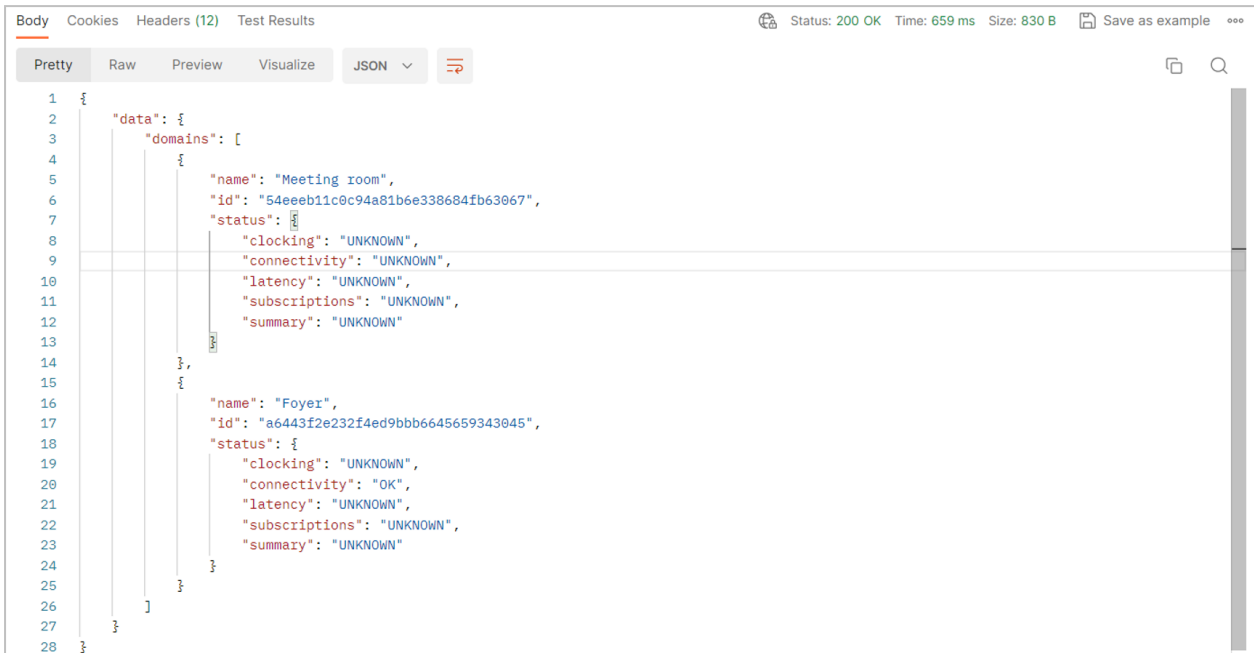
The screenshot shows the Postman interface for a GraphQL request. The request method is set to POST and the URL is {{URL}}. The request body is a GraphQL query:

```
query Domains {
  domains {
    name
    id
    status {
      clocking
      connectivity
      latency
      subscriptions
      summary
    }
  }
}
```

 The GraphQL variables section is empty. The interface also shows tabs for Params, Authorization, Headers (9), Body, Pre-request Script, Tests, and Settings. A 'Send' button is visible in the top right corner.

Click the **Send** button to make the request.

You should receive a JSON object in the response with the names and summary statuses of all domains that the API key has permission to access, similar to the one below.



```

1  {
2    "data": {
3      "domains": [
4        {
5          "name": "Meeting room",
6          "id": "54eeeb11c0c94a81b6e338684fb63067",
7          "status": {
8            "clocking": "UNKNOWN",
9            "connectivity": "UNKNOWN",
10           "latency": "UNKNOWN",
11           "subscriptions": "UNKNOWN",
12           "summary": "UNKNOWN"
13         }
14       },
15       {
16         "name": "Foyer",
17         "id": "a6443f2e232f4ed9bbb6645659343045",
18         "status": {
19           "clocking": "UNKNOWN",
20           "connectivity": "OK",
21           "latency": "UNKNOWN",
22           "subscriptions": "UNKNOWN",
23           "summary": "UNKNOWN"
24         }
25       }
26     ]
27   }
28 }

```

If you receive an authentication error, check to make sure the collection authorization settings from the previous step have been saved.

Requests via cURL

The same request can be performed using cURL with the request shown below.

```

#!/bin/bash

URL="https://api.director.dante.cloud:4000/graphql"
API_KEY=""

query="query Domains {
  domains {
    name
    status {
      clocking
      connectivity
      latency
      subscriptions
      summary
    }
  }
}"

# the 'query' needs to be a one-liner, without newlines
queryOneLine="$(echo $query)"

curl \
  -H 'Content-Type: application/json' \
  -H "Authorization: ${API_KEY}" \
  --data-raw "{ \"query\": \"${queryOneLine}\" }" \
  $URL

```

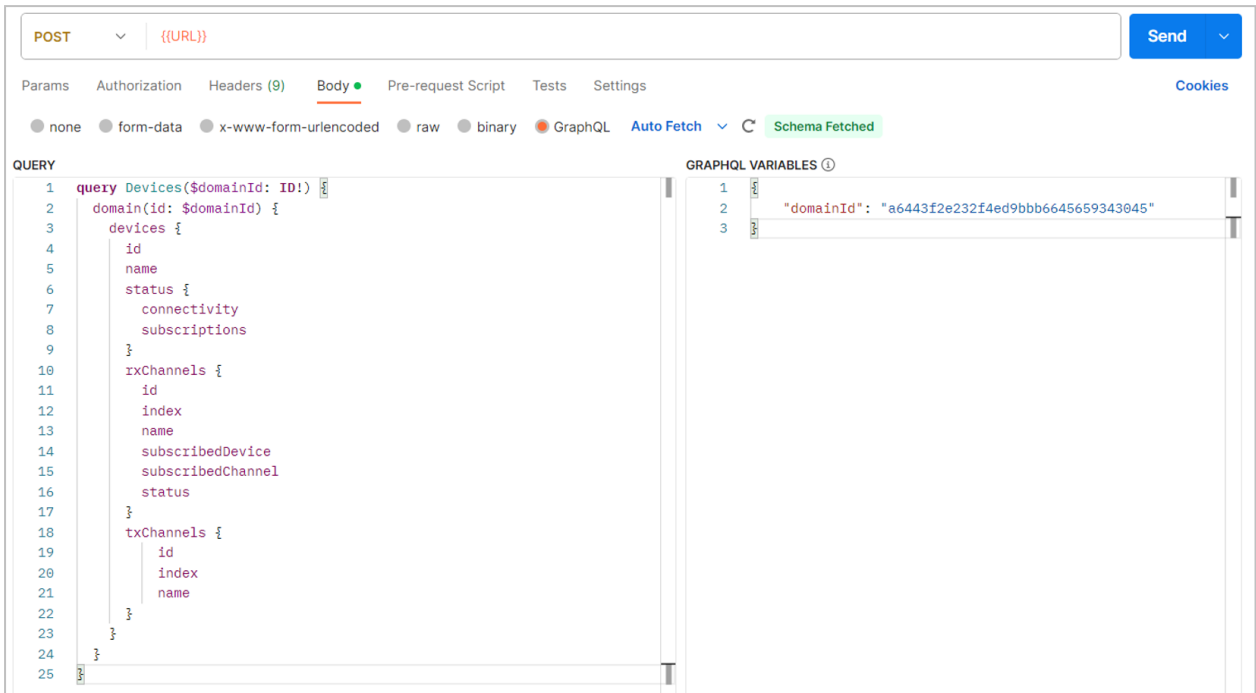
Getting Device and Subscription Status

Device status and subscription information for all devices in a given domain can be obtained using 'Request Devices Status' in the shared Postman collection. This request performs the following query.

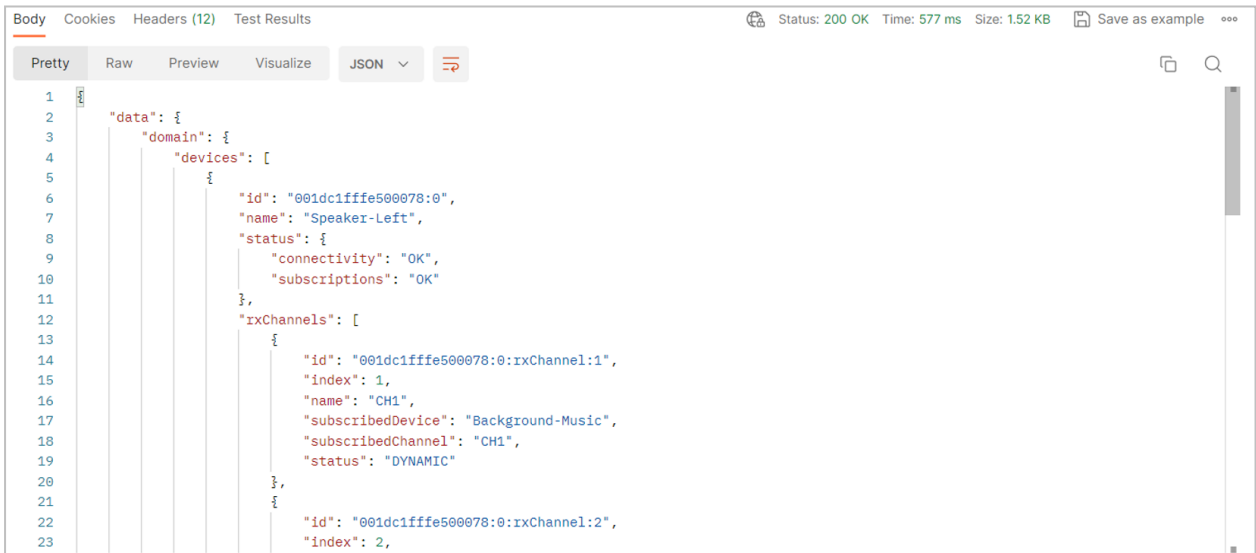
```
query Devices($domainId: ID!) {  
  domain(id: $domainId) {  
    devices {  
      id  
      name  
      status {  
        connectivity  
        subscriptions  
      }  
      rxChannels {  
        id  
        index  
        name  
        subscribedDevice  
        subscribedChannel  
        status  
      }  
      txChannels {  
        id  
        index  
        name  
      }  
    }  
  }  
}
```

For this request, a variable must be provided, which is the name of the domain of the devices you wish to query.

The complete Postman request should look like this:



This will return a response containing a list of all the devices in the specified domain, including the device status and receive channel subscriptions and subscription status. An example response is shown below.



A receive channel status of 'DYNAMIC' indicates a successfully connected unicast subscription, and 'STATIC' indicates a successful multicast subscription.

Getting Device and Subscription Status via cURL

The equivalent request for device and subscription status via cURL is shown below.

```
#!/bin/bash
URL="https://api.director.dante.cloud/graphql"
API_KEY=""

# note the '$' symbol needs to be escaped
query="query Devices(\$domainId: ID!) {
  domain(id: \$domainId) {
    devices {
      id
      name
      status {
        connectivity
        subscriptions
      }
      rxChannels {
        id
        index
        name
        subscribedDevice
        subscribedChannel
        status
      }
      txChannels {
        id
        index
        name
      }
    }
  }
}"

# the 'query' needs to be a one-liner, without newlines
queryOneLine="$(echo $query)"

variables="{\"domainId\": \"a6443f2e232f4ed9bbb6645659343045\"}"

curl \
  -H 'Content-Type: application/json' \
  -H "Authorization: ${API_KEY}" \
  --data-raw "{\"query\": \"${queryOneLine}\", \"variables\": $variables}" \
  $URL
```

Setting (or Clearing) a Channel Subscription

As well as querying the status of a device's receive channel subscriptions, the Dante Managed API can also be used to set (and clear) subscriptions.

The request below uses a GraphQL mutation to change the subscriptions for a device.

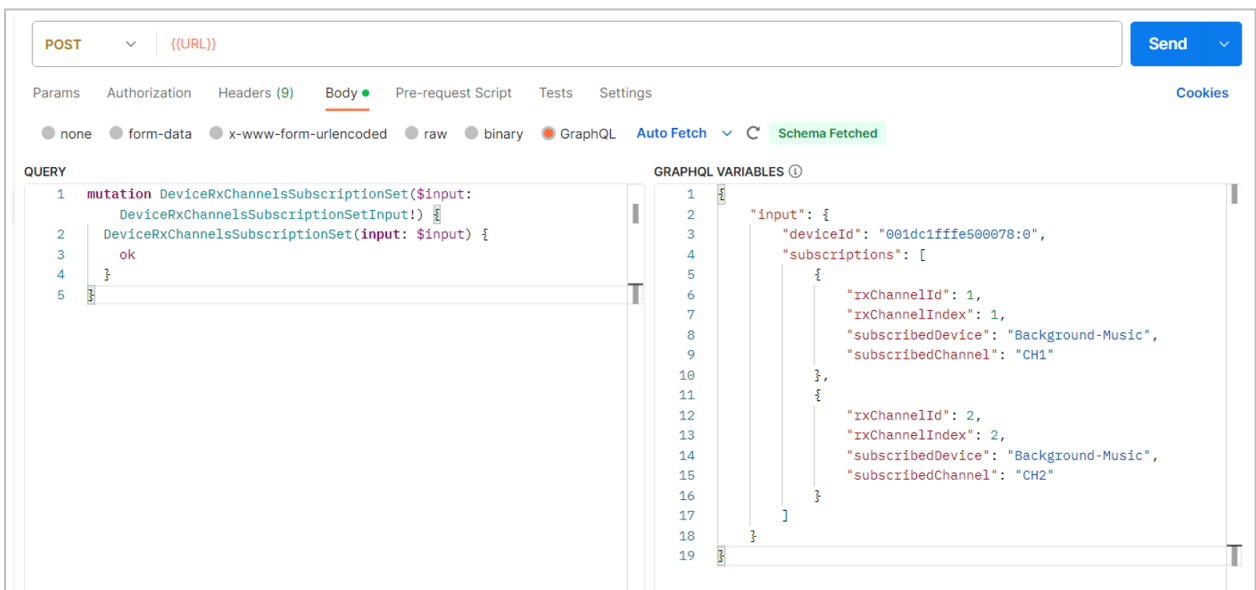
```
mutation DeviceRxChannelsSubscriptionSet
  ($input: DeviceRxChannelsSubscriptionSetInput!) {
  DeviceRxChannelsSubscriptionSet (input: $input) {
    ok
  }
}
```

This mutation requires an input variable to specify what change to make. Below is an example. The values for all the input variable fields can be obtained from the response of the devices request shown previously.

```

{
  "input": {
    "deviceId": "001dc1fffe51f9e9:0",
    "subscriptions": [
      {
        "rxChannelIndex": 1,
        "subscribedDevice": "Mixer",
        "subscribedChannel": "CH2"
      },
      {
        "rxChannelIndex": 2,
        "subscribedDevice": "Mixer",
        "subscribedChannel": "CH1"
      }
    ]
  }
}
    
```

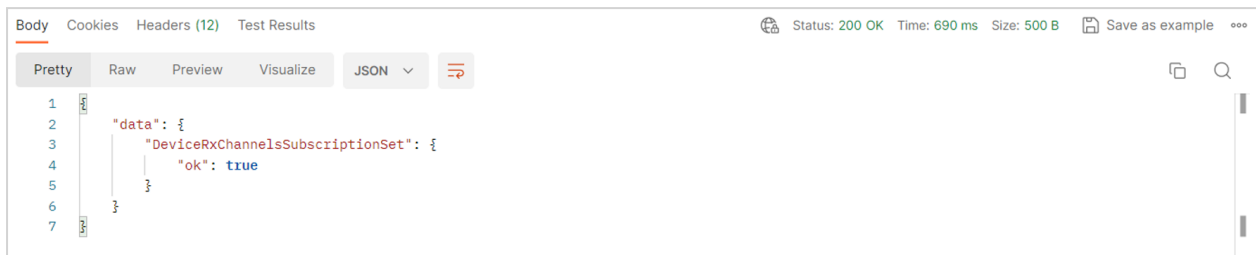
- `deviceId` is the ID of the device that will be receiving the subscriptions. An array of subscriptions you wish to set follows. Any channels on the receive device that are not specified in this array remain unchanged.
- `rxChannelIndex` is the index of the receive device channel for this subscription.
- `subscribedDevice` is the name of the transmitting device.
- `subscribedChannel` is the name of the channel on the transmitting device to subscribe to.



The screenshot shows a REST client interface with a POST request to a URL. The request body is a GraphQL mutation: `mutation DeviceRxChannelsSubscriptionSet($input: DeviceRxChannelsSubscriptionSetInput!) { DeviceRxChannelsSubscriptionSet(input: $input) { ok } }`. The GraphQL variables are: `{ "input": { "deviceId": "001dc1fffe50078:0", "subscriptions": [{ "rxChannelId": 1, "rxChannelIndex": 1, "subscribedDevice": "Background-Music", "subscribedChannel": "CH1" }, { "rxChannelId": 2, "rxChannelIndex": 2, "subscribedDevice": "Background-Music", "subscribedChannel": "CH2" }] } }`.

Why are the receiving device and channel identified by ID / index, and the transmitting device and channel identified by name? This is how Dante devices manage their channel subscriptions: the receiver stores the transmitter details by name, allowing transmitters to be substituted with other devices if desired.

This mutation should return an `"ok": true` result (as shown below) to indicate that the subscription request has been sent.



```

Body Cookies Headers (12) Test Results
Pretty Raw Preview Visualize JSON
1
2   "data": {
3     "DeviceRxChannelsSubscriptionSet": {
4       "ok": true
5     }
6   }
7
Status: 200 OK Time: 690 ms Size: 500 B Save as example

```

To determine if the device has been able to successfully establish the subscription, the receive channel status for the receiving device should be queried using the devices query shown previously. It may take a few seconds for the device to establish the connection.

Subscription status for devices can also be confirmed using Dante Controller (after logging Dante Controller in to Dante Domain Manager or Dante Director, and selecting the relevant domain).

Clearing a Subscription

To clear a subscription, provide empty strings for the `txDeviceName` and `txChannelName` input fields.

Setting a Channel Subscription via cURL

The same request to set a channel subscription can also be performed via cURL. The example above via cURL is shown below.

```

#!/bin/bash

URL="https://api.director.dante.cloud/graphql"
API_KEY=""

# note the '$' symbol needs to be escaped
query="mutation DeviceRxChannelsSubscriptionSet (\$input: deviceRxChannelsSubscriptionSetInput!) {
  DeviceRxChannelsSubscriptionSet(input: \$input) {
    ok
  }
}"

# the 'query' needs to be a one-liner, without newlines
queryOneLine="$(echo $query)"

variables="{
  \"input\": {
    \"deviceId\": \"001dc106e9c80000:0\",
    \"subscriptions\": [
      {
        \"rxChannelIndex\": 2,
        \"subscribedDevice\": \"Mixer\",
        \"subscribedChannel\": \"CH2\"
      }
    ]
  }
}"

curl \

```

```
-H 'Content-Type: application/json' \  
-H "Authorization: ${API_KEY}" \  
--data-raw "{ \"query\": \"${queryOneLine}\", \"variables\": $variables}" \  
$URL
```

Appendix 1 - DDM GraphQL Schema Reference

DDM 1.8 GraphQL Schema Reference

```

"""Controls the rate of traffic."""
directive @sustainedRateLimit(
  """Number of occurrences allowed over duration."""
  limit: Int! = 60

  """Number of seconds before limit is reset."""
  duration: Int! = 60
) on OBJECT | FIELD_DEFINITION

"""Controls the rate of traffic."""
directive @loginRateLimit(
  """Number of occurrences allowed over duration."""
  limit: Int! = 60

  """Number of seconds before limit is reset."""
  duration: Int! = 60
) on OBJECT | FIELD_DEFINITION

directive @hidden on OBJECT | INPUT_OBJECT | FIELD_DEFINITION | INPUT_FIELD_DEFINITION

"""
Indicates exactly one field must be supplied and this field must not be `null`.
"""
directive @oneOf on INPUT_OBJECT

"""
A date-time string at UTC, such as 2007-12-03T10:15:30Z, compliant with the `date-time` format
outlined in section 5.6 of the RFC 3339 profile of the ISO 8601 standard for representation of dates
and times using the Gregorian calendar.
"""
scalar DateTime

type Query {
  domains: [Domain!]!
  domain(id: ID, name: String @deprecated(reason: "Dante Director could have multiple domains in
different accounts with the same name. Please use domain id to query")): Domain
  deviceEntitlements: [DeviceEntitlement!]!
  unenrolledDevices: [Device!]!
}

type Mutation {
  DevicesEnroll(input: DevicesEnrollInput!): DevicesEnrollPayload
  DevicesUnenroll(input: DevicesUnenrollInput!): DevicesUnenrollPayload
  DeviceRxChannelsSubscriptionSet(input: DeviceRxChannelsSubscriptionSetInput!):
DeviceRxChannelsSubscriptionSetPayload
  DomainAdd(input: DomainAddInput!): DomainAddPayload!
  DomainRemove(input: DomainRemoveInput!): DomainRemovePayload!
  DomainUpdate(input: DomainUpdateInput!): DomainUpdatePayload!
  ClockingGroupAutomaticallyConfigure(input: ClockingGroupAutomaticallyConfigureInput!):
ClockingGroupAutomaticallyConfigurePayload
  ClockingGroupSet(input: ClockingGroupSetInput!): ClockingGroupSetPayload

```

```

    DeviceClockingUnicastSet(input: DeviceClockingUnicastSetInput!): DeviceClockingUnicastSetPayload
    DeviceClockingPreferredLeaderSet(input: DeviceClockingPreferredLeaderSetInput!):
DeviceClockingPreferredLeaderSetPayload
    DeviceClockingSyncToExternalSet(input: DeviceClockingSyncToExternalSetInput!):
DeviceClockingSyncToExternalSetPayload
    DeviceClockingPTPV1UnicastDelayRequestSet(input: DeviceClockingPTPV1UnicastDelayRequestSetInput!):
DeviceClockingPTPV1UnicastDelayRequestSetPayload
    DeviceClockingPreferencesOverridesSet(input: DeviceClockingPreferencesOverridesSetInput!):
DeviceClockingPreferencesOverridesSetPayload
    DeviceNameSet(input: DeviceNameSetInput!): DeviceNameSetPayload
    DomainClockingGroupAssign(input: DomainClockingGroupAssignInput!): DomainClockingGroupAssignPayload
}

interface MutationResponse {
  ok: Boolean!
}

type ClockingGroup {
  id: ID!
  mode: ClockingGroupMode
  ptp: ClockingGroupPtp
  rtp: ClockingGroupRtp
}

enum ClockingGroupMode {
  DEFAULT
  AES67
  SMPTE
}

type ClockingGroupPtp {
  id: ID!
  dantePriorities: Boolean
  followerOnly: Boolean
  v1: Boolean
  v1SubdomainName: String
  v2: Boolean
  v2AnnounceInterval: Int
  v2DomainNumber: Int
  v2MulticastTtl: Int
  v2Priority1: Int
  v2Priority2: Int
  v2SyncInterval: Int
}

type ClockingGroupRtp {
  id: ID!
  prefixV4: Int
  rxLatency: Int
  systemPacketTime: Int
  transmitPort: Int
}

type Domain {
  clockingGroup: ClockingGroup
  id: ID!
  name: String
  legacyInterop: Boolean
  clockingGroupId: ID
  icon: DomainIcon
  accountId: ID
  mediaEncryptionPolicy: DomainMediaEncryptionPolicy
  devices: [Device]
  device(id: ID!): Device
  status: DomainStatus
}

```

```
type Device {
  domain: Domain
  id: ID!
  name: String!
  identity: Identity
  manufacturer: Manufacturer
  domainId: ID
  interfaces: [Interface]
  enrolmentState: EnrolmentState
  discovery: DeviceDiscovery
  connection: DeviceConnection
  clockPreferences: ClockPreferences
  capabilities: Capabilities
  compatibilityResults: [DeviceCompatibilityResult!]
  picture: String
  location: String
  description: String
  comments: String
  platform: Platform
  product: Product
  clockingState: ClockingState
  rxChannels: [RxChannel]
  txChannels: [TxChannel]
  status: DeviceStatus
}

enum DomainIcon {
  BROADCAST
  RECORDING_STUDIO
  HOUSE_OF_WORSHIP_1
  HOUSE_OF_WORSHIP_2
  HIGHER_EDUCATION
  PRIMARY_EDUCATION
  GOVERNMENT
  TRANSPORTATION_1
  TRANSPORTATION_2
  HEALTH_CARE
  COMMERCIAL
  OFFICE
  MANUFACTURING
  RETAIL
  ARTS_VENUE
  THEATRE_CINEMA
  RESTAURANT
  BAR_CLUB
  HOTEL_CASINO
  THEMED_ENTERTAINMENT
  SPORTS_VENUE_1
  SPORTS_VENUE_2
  RESIDENTIAL_1
  RESIDENTIAL_2
  OTHER
}

enum DomainMediaEncryptionPolicy {
  NONE
  COMPATIBLE
  STRICT
}

type DeviceOverrides {
  id: ID!
  ptp: DevicePtp
  rtp: DeviceRtp
}
```

```

type DevicePtp {
  id: ID!
  v2Priority1: Int
  v2Priority2: Int
}

type DeviceRtp {
  id: ID!
  prefixV4: Int
}

"""
type DeviceDiscovery {
  id: ID!
  fqdn: String
  type: DiscoveryType!
}

enum DiscoveryType {
  UNICAST_DNS
  MULTICAST_DNS
  STATIC_CONFIG
  UNKNOWN
}

"""
The Manufacturer of a device or software.

Examples: "Acme Incorporated"

See also: Product
"""
type Manufacturer {
  id: ID!
  name: String!
}

type Capabilities {
  id: ID!
  CAN_WRITE_PREFERRED_MASTER: Boolean
  CAN_WRITE_EXT_WORD_CLOCK: Boolean
  CAN_WRITE_SLAVE_ONLY: Boolean
  CAN_WRITE_UNICAST_DELAY_REQUESTS: Boolean
  CAN_UNICAST_CLOCKING: Boolean
  CAN_ENCRYPT_MEDIA: Boolean
  mediaTypes: Int
}

type ClockPreferences {
  id: ID!
  externalWordClock: Boolean
  leader: Boolean
  unicastClocking: Boolean
  v1UnicastDelayRequests: Boolean
  overrides: DeviceOverrides
}

type DeviceCompatibilityResult {
  id: ID!
  name: String!
  count: Int!
  passes: Int!
  failures: Int!
}

```

```

type Identity {
  id: ID!
  instanceId: ID!
  """
  The name that the device will have when it is unboxed or factory reset. Typically contains the MAC
  address. Example: 'AVIOA02-51f9e7'
  """
  defaultName: String
  """The name given to the device by the user. Example: 'Foyer'"""
  actualName: String
  """
  The name given to the device (actualName) which may be appended with a numeric identifier if a
  device of the same actualName already exists. Example: 'Foyer (2)'
  """
  advertisedName: String @deprecated(reason: "Not used in managed environments. Use device.name")
  manufacturerId: String @deprecated(reason: "Use manufacturer.id")
  manufacturerName: String @deprecated(reason: "Use manufacturer.name")
  productModelId: String
  productModelName: String
  productVersion: String
  productSoftwareVersion: String
  danteVersion: String
  danteHardwareVersion: String
}

type Interface {
  id: ID!
  macAddress: String
  address: String
  netmask: Int
  subnet: String
}

type DeviceConnection {
  id: ID!
  state: ConnectionState
  lastChanged: DateTime
}

enum ConnectionState {
  DISCONNECTED
  ESTABLISHED
  READY
}

enum EnrolmentState {
  UNENROLLED
  ENROLLED
  ENROLLING
  UNENROLLING
  CHANGING_DOMAIN
}

"""
The device clocking information in the network
for example: whether the device is leader in current network
"""
type ClockingState {
  id: ID!
  locked: ClockSyncServo
  grandLeader: Boolean
  followerWithoutLeader: Boolean
  multicastLeader: Boolean
}
    
```

```

unicastLeader: Boolean
unicastFollower: Boolean
muteStatus: MuteStatus
frequencyOffset: Int
}

"""
The device Platform describes the Dante Platform that the device is built upon.

This is the Audinate product used *inside* the manufacturers Product. See also: Product

Examples: Brooklyn 2, Ultimo, DEP, etc.

This section will be extended with additional information about the Platform as required. Examples
include
- Icon
- Latest available firmware version
- Links to release notes
- Capabilities
"""
type Platform {
  id: ID!
  name: String!
}

"""
The device Product describes information about the manufactured product (that is, the device the end-
user interacts with)

This is the OEM product (ie. the outside of the box). See also: Platform and Manufacturer

Examples include: "Amplifier 5000", "Mixer 900"

This section will be extended with additional information about the Platform as required. Examples
include
- Icon
- Latest available firmware version
- Links to release notes
- Capabilities
"""
type Product {
  """The name of the product from the OEM"""
  id: ID!
  name: String!
}

type RxChannel {
  """ A unique identifier for the channel, used for caching purposes. Use 'index' when displaying to
users
  """
  id: ID!

  """
  The channel number, starting at 1.

  Examples:
  - 1
  - 19

  This index is unique on the given device and is not media specific.
  """
  index: Int!
  enabled: Boolean

  """ The name of the channel """
  name: String

```

```

    """ The device name of the device that this channel is subscribed to """
    subscribedDevice: String
}

    """ The Tx channel name on the the device that this channel is subscribed to
    """
    subscribedChannel: String
    status: RxChannelStatus
    statusMessage: String
    summary: RxChannelSummary
    mediaType: ChannelMediaType
    encryptionScheme: MediaEncryptionScheme
}

enum ChannelMediaType {
    AUDIO
    VIDEO
    ANCILLARY
    UNDEFINED
    ALL
    GENERIC
}

enum RxChannelStatus {
    """ No subscription for this channel """
    NONE

    """ Error: Channel Name not yet found on network """
    UNRESOLVED

    """ Channel Name has been found, but not yet processed. This is an transient state before the flow
    is created
    """
    RESOLVED

    """ Error: an error occurred while trying to resolve the channel name """
    RESOLVE_FAIL

    """ Channel is successfully subscribed to own TX channels (local loopback mode)
    """
    SUBSCRIBE_SELF

    """ Error: Channel Name explicitly does not exist on this network """
    RESOLVED_NONE

    """
    A flow has been configured but does not have sufficient information to
    establish an audio connection.

    For example, configuring a template with no associations.
    """
    IDLE

    """ Channel Name has been found and processed; setting up flow. This is an transient state
    """
    IN_PROGRESS

    """ Active subscription to an automatically configured source flow """
    DYNAMIC

    """ Active subscription to a manually configured source flow """
    STATIC

    """ Manual flow configuration bypassing the standard subscription process
    """
    MANUAL
}

```

```

""
Error: The name was found but the connection process failed
(the receiver could not communicate with the transmitter)
""
NO_CONNECTION

"" Error: Channel formats do not match ""
CHANNEL_FORMAT

"" Error: Flow formats do not match ""
BUNDLE_FORMAT

"" Error: Receiver is out of resources (e.g. flows) ""
NO_RX

"" Error: Receiver couldn't set up the flow ""
RX_FAIL

"" Error: Transmitter is out of resources (e.g. flows) ""
NO_TX

"" Error: Transmitter couldn't set up the flow ""
TX_FAIL

"" Error: Receiver got a QoS failure (too much data) when setting up the flow.
""
QOS_FAIL_RX

"" Error: Transmitter got a QoS failure (too much data) when setting up the flow.
""
QOS_FAIL_TX

"" Error: Tx rejected the address given by rx (usually indicates an ARP failure)
""
TX_REJECTED_ADDR

"" Error: Transmitter rejected the bundle request as invalid ""
INVALID_MSG

"" Error: Tx channel latency higher than maximum supported Rx latency ""
CHANNEL_LATENCY

"" Error: Tx and Rx and in different clock subdomains ""
CLOCK_DOMAIN

"" Error: Attempt to use an unsupported feature ""
UNSUPPORTED

"" Error: All Rx links are down ""
RX_LINK_DOWN

"" Error: All Tx links are down ""
TX_LINK_DOWN

"" Error: can't find suitable protocol for dynamic connection ""
DYNAMIC_PROTOCOL

"" Error: Channel does not exist (eg no such local channel) ""
INVALID_CHANNEL

"" Error: Tx Scheduler failure ""
TX_SCHEDULER_FAILURE

"" Error: The given subscription to self was disallowed by the device ""
SUBSCRIBE_SELF_POLICY

```

```

"""" Warning: There is an external issue with Tx """"
TX_NOT_READY

"""" Warning: There is an external issue with Rx """"
RX_NOT_READY

"""" Error: Tx device cannot support additional unicast flows """"
TX_FANOUT_LIMIT_REACHED

"""" Error: Rx device does not support the signal encryption """"
TX_CHANNEL_ENCRYPTED

"""" Error: Unexpected response from TX device """"
TX_RESPONSE_UNEXPECTED

"""" Error: Template-based subscription failed: template and subscription device names don't match
""""
TEMPLATE_MISMATCH_DEVICE

"""" Error: Template-based subscription failed: flow and channel formats don't match
""""
TEMPLATE_MISMATCH_FORMAT

"""" Error: Template-based subscription failed: the channel is not part of the given multicast flow
""""
TEMPLATE_MISSING_CHANNEL

""""
Error: Template-based subscription failed: something else about the template configuration
made it impossible to complete the subscription using the given flow
""""
TEMPLATE_MISMATCH_CONFIG

"""" Error: Template-based subscription failed: the unicast template is full
""""
TEMPLATE_FULL

"""" Error: Tx access control denied the request """"
TX_ACCESS_CONTROL_DENIED

"""" Tx access control request is in progress """"
TX_ACCESS_CONTROL_PENDING

"""" Error: Rx device does not have a supported subscription mode (unicast/multicast) available
""""
RX_UNSUPPORTED_SUB_MODE

"""" Error: Tx device does not have a supported subscription mode (unicast/multicast) available
""""
TX_UNSUPPORTED_SUB_MODE

"""" Error: HDCP key negotiation failed """"
HDCP_NEGOTIATION_FAILED

"""" Error: Unexpected system failure """"
SYSTEM_FAIL
}

enum RxChannelSummary {
    NONE
    CONNECTED
    IN_PROGRESS
    WARNING
    ERROR
}
    
```

```

enum MediaEncryptionScheme {
    NONE
    HDCP
    AES_CTR256
}

enum TxChannelMediaEncryptionPolicy {
    COMPATIBLE
    STRICT
}

""""A transmit channel of media (either audio or video) from a device""""
type TxChannel {
    id: ID!
    index: Int!
    name: String!
    mediaType: ChannelMediaType
    encryptionPolicy: TxChannelMediaEncryptionPolicy
}

enum ClockSyncServo {
    """"if a device's servo is in sync or a grand leader for uuidV2""""
    LOCKED

    """"
    if a device is online and servo is not in ( sync or a grand leader for uuidV2 )
    """"
    NOT_LOCKED

    """"
    if a device is offline (this UNKNOWN is not the same as unknown enum in clockServo or servo state,
    we cannot distinguish unknown and none in ddm side, dante model cpp api has NONE value
    )
    """"
    UNKNOWN
}

enum MuteStatus {
    """"PTP clocking is operating normally and audio is not muted""""
    NOT_MUTED

    """"
    PTP clocking is not operating normally, the device is using an external clock source, and audio is
    muted
    """"
    MUTED_EXTERNAL_CLOCK

    """"
    PTP clocking is not operating normally, the device is using an internal clock source, and audio is
    muted
    """"
    MUTED_INTERNAL_CLOCK

    """"user requested audio mute""""
    MUTED_USER

    """"clocking mute status is unknown""""
    UNKNOWN
}

enum Status {
    OK
    WARNING
    ERROR
    UNKNOWN
}

```

```

}

type DeviceStatus {
  id: ID!
  clocking: Status
  connectivity: Status
  latency: Status
  subscriptions: Status
  summary: Status
  alertMessage: AlertMessage
}

type DomainStatus {
  id: ID!
  clocking: Status
  connectivity: Status
  latency: Status
  subscriptions: Status
  domainAlertMessage: DomainAlertMessages
  summary: Status
}

type DomainAlertMessages {
  id: ID!
  connectivity: DomainAlertMessage
  clocking: DomainAlertMessage
  latency: DomainAlertMessage
  subscriptions: DomainAlertMessage
}

type DomainAlertMessage {
  id: ID!
  message: String!
  messageSeverity: Status!
}

type AlertMessage {
  id: ID!
  connectivity: String
  clocking: String
  latency: String
  subscriptions: String
}

type DeviceEntitlement {
  id: ID!
  name: String!

  """
  Limit is the number of devices that can be licensed using this entitlement.
  """
  limit: Int!

  """
  EntitledDevices are devices that are already licensed.
  If the entitledDevices' length is longer than the limit, it is over the limit for this Device's
  Entitlement
  """
  entitledDevices: [Device]!

  """
  A list of Matching device product Ids for this entitlement. There could be more than 1 types of
  products use this entitlement
  """
  matchIds: [EntitlementMatchIds]!
}

```

```

    """If current entitlement is over the limit"""
    overLimit: Boolean
  }

type EntitlementMatchIds {
  id: ID!
  name: String

  """
  manufacturer Id for the license (same as manufacturer.Id in the Device type)
  """
  manufacturerId: String!

  """
  Product model Id for the license (same as productModelId in the Device.identity)
  """
  productModelId: String!
}

input DevicesEnrollInput {
  domainId: ID!
  deviceIds: [ID!]!

  """
  Clear Configuration will reset the following configuration settings to the device defaults:
  - Device Name
  - Channel labels
  - Latency
  - Sample rate
  - Encoding
  - Subscriptions
  Note: Clear Configuration is not supported for legacy devices.
  """
  clearConfig: Boolean = false
}

type DevicesEnrollPayload implements MutationResponse {
  ok: Boolean!
}

input DevicesUnenrollInput {
  deviceIds: [ID!]!

  """
  Clear Configuration will reset the following configuration settings to the device defaults:
  - Device Name
  - Channel labels
  - Latency
  - Sample rate
  - Encoding
  - Subscriptions
  Note: Clear Configuration is not supported for legacy devices.
  """
  clearConfig: Boolean = false
}

type DevicesUnenrollPayload implements MutationResponse {
  ok: Boolean!
}

input DeviceRxChannelsSubscriptionSetInput {
  """
  The ID of the device which will receive the subscription

  Examples:
  "0ae3b2edf1374c0c836c96649e879c2f" (for software devices like DVS)
  """
}

```

```

    "001dc1ffffe501c25:0" (for hardware devices like AVIO)
    ""
    deviceId: ID!
    subscriptions: [DeviceRxChannelsSubscriptionInput!]!
    ""
    Allows setting a subscription to a device name when that device does not currently exist, typically
    because it is offline.

    Subscriptions to unknown devices may be helpful when restoring a preset.

    Dante resolves subscriptions by name, so this is a valid configuration. However, the subscription
    will remain unresolved until the transmitter device is online.

    If set, 'allowSubscriptionToNonExistentChannel' must also be set since the channel name cannot
    resolve when the device is offline
    ""
    allowSubscriptionToNonExistentDevice: Boolean = false
    ""
    Allows setting a subscription to a channel name even when the transmitter device does not have a
    channel by that name

    Subscriptions to unknown devices or channels may be helpful when restoring a preset.

    Dante resolves subscriptions by name, so this is a valid configuration. However, the subscription
    will remain unresolved until the transmitter device has a channel by this name.

    If this value is set to true, then additional checks (such as checking the media types of each
    channel match) will not be performed.
    ""
    allowSubscriptionToNonExistentChannel: Boolean = false
}

input DeviceRxChannelsSubscriptionInput {
  rxChannelId: Int @deprecated(reason: "Use rxChannelIndex instead")

  ""
  The channel number on the device which will receive the subscription

  Examples:
  - 1
  - 19

  Channel numbers start at 1

  This index is unique on the given device and is not media specific.
  The index to use should be determined by first querying the channels on the device, finding the
  channel you want to modify, and then use that index.
  ""
  rxChannelIndex: Int!

  ""
  The name of the device which is transmitting the media that you wish to subscribe to

  Device *names* are used in Dante subscriptions (rather than IDs) so that the the subscription will
  follow a device of that name, rather than a specific device of that ID.
  This allows the transmitting device to be swapped out for a device of the same name.

  Example: "AVIOA02-51f9e7"

  An empty subscribedDevice and subscribedChannel will unsubscribe the channel
  ""
  subscribedDevice: String!
  ""

```

The channel name on the device which is transmitting the media

Channel *labels* are used in Dante (rather than IDs) so that the the subscription will follow the channel of that label, rather than a specific channel of that ID. This allows the transmitting device rearrange it's transmit channels, without needing to update the subscriptions on receiving devices.

Examples:

```
"Audio L"
"CH1"
"01"
```

An empty subscribedDevice and subscribedChannel will unsubscribe the channel

```
""
```

```
subscribedChannel: String!
```

```
}
```

```
type DeviceRxChannelsSubscriptionSetPayload implements MutationResponse {
```

```
  ok: Boolean!
```

```
}
```

```
input DomainAddInput {
```

```
  name: String!
```

```
  icon: DomainIcon!
```

```
  accountId: ID
```

```
}
```

```
type DomainAddPayload implements MutationResponse {
```

```
  ok: Boolean!
```

```
  domain: Domain
```

```
}
```

```
input DomainRemoveInput {
```

```
  id: ID!
```

```
}
```

```
type DomainRemovePayload implements MutationResponse {
```

```
  ok: Boolean!
```

```
}
```

```
input DomainUpdateInput {
```

```
  id: ID!
```

```
  name: String
```

```
  icon: DomainIcon
```

```
}
```

```
type DomainUpdatePayload implements MutationResponse {
```

```
  ok: Boolean!
```

```
  domain: Domain
```

```
}
```

```
input ClockingGroupAutomaticallyConfigureInput {
```

```
  clockingGroupId: ID!
```

```
}
```

```
type ClockingGroupAutomaticallyConfigurePayload implements MutationResponse {
```

```
  ok: Boolean!
```

```
}
```

```
input ClockingGroupSetInput {
```

```
  clockingGroupId: ID!
```

```
  mode: ClockingGroupMode
```

```
  ptp: ClockingGroupPtpInput
```

```
  rtp: ClockingGroupRtpInput
```

```
}
```

```

input ClockingGroupPtpInput {
  dantePriorities: Boolean
  followerOnly: Boolean
  v1: Boolean
  v1SubdomainName: String
  v2: Boolean
  v2AnnounceInterval: Int
  v2DomainNumber: Int
  v2MulticastTtl: Int
  v2Priority1: Int
  v2Priority2: Int
  v2SyncInterval: Int
}

input ClockingGroupRtpInput {
  prefixV4: Int
  rxLatency: Int
  systemPacketTime: Int
  transmitPort: Int
}

type ClockingGroupSetPayload implements MutationResponse {
  ok: Boolean!
}

input DeviceClockingUnicastSetInput {
  deviceId: ID!
  enabled: Boolean!
}

type DeviceClockingUnicastSetPayload implements MutationResponse {
  ok: Boolean!
}

input DeviceClockingPreferredLeaderSetInput {
  deviceId: ID!
  enabled: Boolean!
}

type DeviceClockingPreferredLeaderSetPayload implements MutationResponse {
  ok: Boolean!
}

input DeviceClockingSyncToExternalSetInput {
  deviceId: ID!
  enabled: Boolean!
}

type DeviceClockingSyncToExternalSetPayload implements MutationResponse {
  ok: Boolean!
}

input DeviceClockingPTPV1UnicastDelayRequestSetInput {
  deviceId: ID!
  enabled: Boolean!
}

type DeviceClockingPTPV1UnicastDelayRequestSetPayload implements MutationResponse {
  ok: Boolean!
}

input DeviceClockingPreferencesOverridesSetInput {
  clockingGroupId: ID!
  deviceId: ID!
  ptp: DeviceClockPreferencesOverridesPtp
  rtp: DeviceClockPreferencesOverridesRtp
}

```

```

}

input DeviceClockPreferencesOverridesPtp {
  v2Priority1: Int
  v2Priority2: Int
}

input DeviceClockPreferencesOverridesRtp {
  prefixV4: Int
}

type DeviceClockingPreferencesOverridesSetPayload implements MutationResponse {
  ok: Boolean!
}

input DeviceNameSetInput {
  """
  The ID of the device to rename

  Examples:
  "0ae3b2edf1374c0c836c96649e879c2f" (for software devices like DVS)
  "001dc1fffe501c25:0" (for hardware devices like AVIO)
  """
  deviceId: ID!

  """
  New name for the device
  Maximum of 31 characters and must contain only letters, numbers and -.
  The first and last characters cannot be -.
  Provide an empty string to reset to default.
  """
  name: String!
}

type DeviceNameSetPayload implements MutationResponse {
  ok: Boolean!
}

input DomainClockingGroupAssignInput {
  domainId: ID
  clockingGroupId: ID
}

type DomainClockingGroupAssignPayload implements MutationResponse {
  ok: Boolean!
}

```

DDM 1.7 GraphQL Schema Reference

```

"""Controls the rate of traffic."""
directive @sustainedRateLimit(
  """Number of occurrences allowed over duration."""
  limit: Int! = 60

  """Number of seconds before limit is reset."""
  duration: Int! = 60
) on OBJECT | FIELD_DEFINITION

"""Controls the rate of traffic."""

```

```

directive @loginRateLimit(
  """Number of occurrences allowed over duration."""
  limit: Int! = 60

  """Number of seconds before limit is reset."""
  duration: Int! = 60
) on OBJECT | FIELD_DEFINITION

"""Exposes a URL that specifies the behavior of this scalar."""
directive @specifiedBy(
  """The URL that specifies the behavior of this scalar."""
  url: String!
) on SCALAR

type AlertMessage {
  id: ID!
  connectivity: String
  clocking: String
  latency: String
  subscriptions: String
}

type Capabilities {
  id: ID!
  CAN_WRITE_PREFERRED_MASTER: Boolean
  CAN_WRITE_EXT_WORD_CLOCK: Boolean
  CAN_WRITE_SLAVE_ONLY: Boolean
  CAN_WRITE_UNICAST_DELAY_REQUESTS: Boolean
  CAN_UNICAST_CLOCKING: Boolean
  mediaTypes: Int
}

enum ChannelMediaType {
  AUDIO
  VIDEO
  ANCILLARY
  UNDEFINED
  ALL
  GENERIC
}

type ClockingGroup {
  id: ID!
  mode: ClockingGroupMode
  ptp: ClockingGroupPtp
  rtp: ClockingGroupRtp
}

input ClockingGroupAutomaticallyConfigureInput {
  clockingGroupId: ID!
}

type ClockingGroupAutomaticallyConfigurePayload implements MutationResponse {
  ok: Boolean!
}

enum ClockingGroupMode {
  DEFAULT
  AES67
  SMPTE
}

type ClockingGroupPtp {
  id: ID!
  dantePriorities: Boolean
  followerOnly: Boolean
}

```

```

v1: Boolean
v1SubdomainName: String
v2: Boolean
v2AnnounceInterval: Int
v2DomainNumber: Int
v2MulticastTtl: Int
v2Priority1: Int
v2Priority2: Int
v2SyncInterval: Int
}

input ClockingGroupPtpInput {
  dantePriorities: Boolean
  followerOnly: Boolean
  v1: Boolean
  v1SubdomainName: String
  v2: Boolean
  v2AnnounceInterval: Int
  v2DomainNumber: Int
  v2MulticastTtl: Int
  v2Priority1: Int
  v2Priority2: Int
  v2SyncInterval: Int
}

type ClockingGroupRtp {
  id: ID!
  prefixV4: Int
  rxLatency: Int
  systemPacketTime: Int
  transmitPort: Int
}

input ClockingGroupRtpInput {
  prefixV4: Int
  rxLatency: Int
  systemPacketTime: Int
  transmitPort: Int
}

input ClockingGroupSetInput {
  clockingGroupId: ID!
  mode: ClockingGroupMode
  ptp: ClockingGroupPtpInput
  rtp: ClockingGroupRtpInput
}

type ClockingGroupSetPayload implements MutationResponse {
  ok: Boolean!
}

type ClockingState {
  id: ID!
  locked: ClockSyncServo
  grandLeader: Boolean
  followerWithoutLeader: Boolean
  multicastLeader: Boolean
  unicastLeader: Boolean
  unicastFollower: Boolean
  muteStatus: String
  frequencyOffset: String
}

type ClockPreferences {
  id: ID!
  externalWordClock: Boolean

```

```

    leader: Boolean
    unicastClocking: Boolean
    v1UnicastDelayRequests: Boolean
    overrides: DeviceOverrides
}

enum ClockSyncServo {
    ""if a device's servo is in sync or a grand leader for uuidV2""
    LOCKED

    ""
    if a device is online and servo is not in ( sync or a grand leader for uuidV2 )
    ""
    NOT_LOCKED

    ""
    if a device is offline (this UNKNOWN is not the same as unknown enum in clockServo or servo state,
    we cannot distinguish unknown and none in ddm side, dante model cpp api has NONE value
    )
    ""
    UNKNOWN
}

enum ConnectionState {
    DISCONNECTED
    ESTABLISHED
    READY
}

""
A date-time string at UTC, such as 2007-12-03T10:15:30Z, compliant with the
`date-time` format outlined in section 5.6 of the RFC 3339 profile of the ISO
8601 standard for representation of dates and times using the Gregorian calendar.
""
scalar DateTime

type Device {
    domain: Domain
    id: ID!
    name: String!
    identity: Identity
    manufacturer: Manufacturer
    domainId: ID
    interfaces: [Interface]
    enrolmentState: EnrolmentState
    discovery: DeviceDiscovery
    connection: DeviceConnection
    clockingState: ClockingState
    clockPreferences: ClockPreferences
    capabilities: Capabilities
    picture: String
    location: String
    description: String
    comments: String
    platform: Platform
    product: Product
    rxChannels: [RxChannel]
    txChannels: [TxChannel]
    status: DeviceStatus
}

input DeviceClockingPreferencesOverridesSetInput {
    clockingGroupId: ID!
    deviceId: ID!
    ptp: DeviceClockPreferencesOverridesPtp
    rtp: DeviceClockPreferencesOverridesRtp
}

```

```
}  
  
type DeviceClockingPreferencesOverridesSetPayload implements MutationResponse {  
  ok: Boolean!  
}  
  
input DeviceClockingPreferredLeaderSetInput {  
  deviceId: ID!  
  enabled: Boolean!  
}  
  
type DeviceClockingPreferredLeaderSetPayload implements MutationResponse {  
  ok: Boolean!  
}  
  
input DeviceClockingPTPV1UnicastDelayRequestSetInput {  
  deviceId: ID!  
  enabled: Boolean!  
}  
  
type DeviceClockingPTPV1UnicastDelayRequestSetPayload implements MutationResponse {  
  ok: Boolean!  
}  
  
input DeviceClockingSyncToExternalSetInput {  
  deviceId: ID!  
  enabled: Boolean!  
}  
  
type DeviceClockingSyncToExternalSetPayload implements MutationResponse {  
  ok: Boolean!  
}  
  
input DeviceClockingUnicastSetInput {  
  deviceId: ID!  
  enabled: Boolean!  
}  
  
type DeviceClockingUnicastSetPayload implements MutationResponse {  
  ok: Boolean!  
}  
  
input DeviceClockPreferencesOverridesPtp {  
  v2Priority1: Int  
  v2Priority2: Int  
}  
  
input DeviceClockPreferencesOverridesRtp {  
  prefixV4: Int  
}  
  
type DeviceConnection {  
  id: ID!  
  state: ConnectionState  
  lastChanged: DateTime  
}  
  
type DeviceDiscovery {  
  id: ID!  
  fqdn: String  
  type: DiscoveryType!  
}  
  
type DeviceEntitlement {  
  id: ID!  
  name: String!
```

```

"""
Limit is the number of devices that can be licensed using this entitlement.
"""
limit: Int!

"""
EntitledDevices are devices that are already licensed.
If the entitledDevices' length is longer than the limit, it is over the limit for this Device's
Entitlement
"""
entitledDevices: [Device]!

"""
manufacturer Id for the license (same as manufacturer.Id in the Device type)
"""
manufacturerId: String!

"""
Product model Id for the license (same as productModelId in the Device.identity)
"""
productModelId: String!
}

type DeviceOverrides {
  id: ID!
  ptp: DevicePtp
  rtp: DeviceRtp
}

type DevicePtp {
  id: ID!
  v2Priority1: Int
  v2Priority2: Int
}

type DeviceRtp {
  id: ID!
  prefixV4: Int
}

input DeviceRxChannelsSubscriptionInput {
  """
  The channel number on the device which will receive the subscription

  Examples:
  - 1
  - 19

  Channel numbers start at 1

  This index is unique on the given device and is not media specific.
  The index to use should be determined by first querying the channels on the
  device, finding the channel you want to modify, and then use that index.
  """
  rxChannelIndex: Int!

  """
  The name of the device which is transmitting the media that you wish to subscribe to

  Device *names* are used in Dante subscriptions (rather than IDs) so that the the subscription will
  follow a device of that name, rather than a specific device of that ID.
  This allows the transmitting device to be swapped out for a device of the same name.

  Example: "AVIOAO2-51f9e7"
  """

```

```

    An empty subscribedDevice and subscribedChannel will unsubscribe the channel
    """
    subscribedDevice: String!
    """
    The channel name on the device which is transmitting the media

    Channel *labels* are used in Dante (rather than IDs) so that the the subscription will
    follow the channel of that label, rather than a specific channel of that ID.
    This allows the transmitting device rearrange it's transmit channels,
    without needing to update the subscriptions on receiving devices.

    Examples:
    "Audio L"
    "CH1"
    "01"

    An empty subscribedDevice and subscribedChannel will unsubscribe the channel
    """
    subscribedChannel: String!
}

input DeviceRxChannelsSubscriptionSetInput {
    """
    The ID of the device which will receive the subscription

    Examples:
    "0ae3b2edf1374c0c836c96649e879c2f" (for software devices like DVS)
    "001dc1fffe501c25:0" (for hardware devices like AVIO)
    """
    deviceId: ID!
    subscriptions: [DeviceRxChannelsSubscriptionInput!]!
    """
    Allows setting a subscription to a device name when that device does not currently exist, typically
    because it is offline.

    Subscriptions to unknown devices may be helpful when restoring a preset.

    Dante resolves subscriptions by name, so this is a valid configuration.
    However, the subscription will remain unresolved until the transmitter device is online.

    If set, 'allowSubscriptionToNonExistentChannel' must also be set since the
    channel name cannot resolve when the device is offline
    """
    allowSubscriptionToNonExistentDevice: Boolean = false
    """
    Allows setting a subscription to a channel name even when the transmitter device does not have a
    channel by that name

    Subscriptions to unknown devices or channels may be helpful when restoring a preset.

    Dante resolves subscriptions by name, so this is a valid configuration.
    However, the subscription will remain unresolved until the transmitter device
    has a channel by this name.

    If this value is set to true, then additional checks (such as checking the
    media types of each channel match) will not be performed.
    """
    allowSubscriptionToNonExistentChannel: Boolean = false
}

type DeviceRxChannelsSubscriptionSetPayload implements MutationResponse {
    ok: Boolean!
}

```

```

input DevicesEnrollInput {
  domainId: ID!
  deviceIds: [ID!]!

  """
  Clear Configuration will reset the following configuration settings to the device defaults:
  - Device Name
  - Channel labels
  - Latency
  - Sample rate
  - Encoding
  - Subscriptions
  Note: Clear Configuration is not supported for legacy devices.
  """
  clearConfig: Boolean = false
}

type DevicesEnrollPayload implements MutationResponse {
  ok: Boolean!
}

type DeviceStatus {
  id: ID!
  clocking: Status
  connectivity: Status
  latency: Status
  subscriptions: Status
  summary: Status
  alertMessage: AlertMessage
}

input DevicesUnenrollInput {
  deviceIds: [ID!]!

  """
  Clear Configuration will reset the following configuration settings to the device defaults:
  - Device Name
  - Channel labels
  - Latency
  - Sample rate
  - Encoding
  - Subscriptions
  Note: Clear Configuration is not supported for legacy devices.
  """
  clearConfig: Boolean = false
}

type DevicesUnenrollPayload implements MutationResponse {
  ok: Boolean!
}

enum DiscoveryType {
  UNICAST_DNS
  MULTICAST_DNS
  STATIC_CONFIG
  UNKNOWN
}

type Domain {
  clockingGroup: ClockingGroup
  id: ID!
  name: String
  legacyInterop: Boolean
  clockingGroupId: ID
  icon: DomainIcon
}

```

```

    accountId: ID
    devices: [Device]
    device(id: ID!): Device
    status: DomainStatus
  }

type DomainAlertMessage {
  id: ID!
  message: String!
  messageSeverity: Status!
}

type DomainAlertMessages {
  id: ID!
  connectivity: DomainAlertMessage
  clocking: DomainAlertMessage
  latency: DomainAlertMessage
  subscriptions: DomainAlertMessage
}

input DomainClockingGroupAssignInput {
  domainId: ID
  clockingGroupId: ID
}

type DomainClockingGroupAssignPayload implements MutationResponse {
  ok: Boolean!
}

enum DomainIcon {
  BROADCAST
  RECORDING_STUDIO
  HOUSE_OF_WORSHIP_1
  HOUSE_OF_WORSHIP_2
  HIGHER_EDUCATION
  PRIMARY_EDUCATION
  GOVERNMENT
  TRANSPORTATION_1
  TRANSPORTATION_2
  HEALTH_CARE
  COMMERCIAL
  OFFICE
  MANUFACTURING
  RETAIL
  ARTS_VENUE
  THEATRE_CINEMA
  RESTAURANT
  BAR_CLUB
  HOTEL_CASINO
  THEMED_ENTERTAINMENT
  SPORTS_VENUE_1
  SPORTS_VENUE_2
  RESIDENTIAL_1
  RESIDENTIAL_2
  OTHER
}

type DomainStatus {
  id: ID!
  clocking: Status
  connectivity: Status
  latency: Status
  subscriptions: Status
  domainAlertMessage: DomainAlertMessages
  summary: Status
}

```

```

enum EnrolmentState {
  UNENROLLED
  ENROLLED
  ENROLLING
  UNENROLLING
  CHANGING_DOMAIN
}

type Identity {
  id: ID!
  instanceId: ID!

  """
  The name that the device will have when it is unboxed or factory reset.
  Typically contains the MAC address. Example: 'AVIOA02-51f9e7'
  """
  defaultName: String

  """The name given to the device by the user. Example: 'Foyer'"""
  actualName: String

  """
  The name given to the device (actualName) which may be appended with a numeric
  identifier if a device of the same actualName already exists. Example: 'Foyer (2)'
  """
  advertisedName: String @deprecated(reason: "Not used in managed environments. Use device.name")
  manufacturerId: String @deprecated(reason: "Use manufacturer.id")
  manufacturerName: String @deprecated(reason: "Use manufacturer.name")
  productModelId: String
  productModelName: String
  productVersion: String
  productSoftwareVersion: String
  danteVersion: String
  danteHardwareVersion: String
}

type Interface {
  id: ID!
  macAddress: String
  address: String
  netmask: Int
  subnet: String
}

"""
The Manufacturer of a device or software.

Examples: "Acme Incorporated"

See also: Product
"""
type Manufacturer {
  id: ID!
  name: String!
}

type Mutation {
  DevicesEnroll(input: DevicesEnrollInput!): DevicesEnrollPayload
  DevicesUnenroll(input: DevicesUnenrollInput!): DevicesUnenrollPayload
  DeviceRxChannelsSubscriptionSet(input: DeviceRxChannelsSubscriptionSetInput!):
  DeviceRxChannelsSubscriptionSetPayload
  ClockingGroupAutomaticallyConfigure(input: ClockingGroupAutomaticallyConfigureInput!):
  ClockingGroupAutomaticallyConfigurePayload
  ClockingGroupSet(input: ClockingGroupSetInput!): ClockingGroupSetPayload
  DeviceClockingUnicastSet(input: DeviceClockingUnicastSetInput!): DeviceClockingUnicastSetPayload

```

```

    DeviceClockingPreferredLeaderSet(input: DeviceClockingPreferredLeaderSetInput!):
DeviceClockingPreferredLeaderSetPayload
    DeviceClockingSyncToExternalSet(input: DeviceClockingSyncToExternalSetInput!):
DeviceClockingSyncToExternalSetPayload
    DeviceClockingPTPV1UnicastDelayRequestSet(input: DeviceClockingPTPV1UnicastDelayRequestSetInput!):
DeviceClockingPTPV1UnicastDelayRequestSetPayload
    DeviceClockingPreferencesOverridesSet(input: DeviceClockingPreferencesOverridesSetInput!):
DeviceClockingPreferencesOverridesSetPayload
    DomainClockingGroupAssign(input: DomainClockingGroupAssignInput!): DomainClockingGroupAssignPayload
}

interface MutationResponse {
  ok: Boolean!
}

"""
The device Platform describes the Dante Platform that the device is built upon.

This is the Audinate product used *inside* the manufacturers Product. See also: Product

Examples: Brooklyn 2, Ultimo, DEP, etc.

This section will be extended with additional information about the Platform as required. Examples
include
- Icon
- Latest available firmware version
- Links to release notes
- Capabilities
"""
type Platform {
  id: ID!
  name: String!
}

"""
The device Product describes information about the manufactured product (that is, the device the end-
user interacts with)

This is the OEM product (ie. the outside of the box). See also: Platform and Manufacturer

Examples include: "Amplifier 5000", "Mixer 900"

This section will be extended with additional information about the Platform as required. Examples
include
- Icon
- Latest available firmware version
- Links to release notes
- Capabilities
"""
type Product {
  """The name of the product from the OEM"""
  id: ID!
  name: String!
}

type Query {
  domains: [Domain]!
  domain(id: ID, name: String!): Domain
  deviceEntitlements: [DeviceEntitlement]!
  unenrolledDevices: [Device]!
}

type RxChannel {
  """ A unique identifier for the channel, used for caching purposes. Use 'index' when displaying to
users
"""

```

```

id: ID!

"""
The channel number, starting at 1.

Examples:
- 1
- 19

This index is unique on the given device and is not media specific.
"""
index: Int!
enabled: Boolean

""" The name of the channel """
name: String

""" The device name of the device that this channel is subscribed to """
subscribedDevice: String

""" The Tx channel name on the the device that this channel is subscribed to
"""
subscribedChannel: String
status: RxChannelStatus
statusMessage: String
summary: RxChannelSummary
mediaType: ChannelMediaType
}

enum RxChannelStatus {
    """ No subscription for this channel """
    NONE

    """ Channel Name not yet found on network """
    UNRESOLVED

    """ Channel Name has been found, but not yet processed. This is an transient state before the flow
is created
"""
    RESOLVED

    """ Error: an error occurred while trying to resolve the channel name """
    RESOLVE_FAIL

    """ Channel is successfully subscribed to own TX channels (local loopback mode)
"""
    SUBSCRIBE_SELF

    """ Error: Channel Name explicitly does not exist on this network """
    RESOLVED_NONE

    """
A flow has been configured but does not have sufficient information to
establish an audio connection.

For example, configuring a template with no associations.
"""
    IDLE

    """ Channel Name has been found and processed; setting up flow. This is an transient state
"""
    IN_PROGRESS

    """ Active subscription to an automatically configured source flow """
    DYNAMIC

```

```

"" Active subscription to a manually configured source flow ""
STATIC

"" Manual flow configuration bypassing the standard subscription process
""
MANUAL

""
Error: The name was found but the connection process failed
(the receiver could not communicate with the transmitter)
""
NO_CONNECTION

"" Error: Channel formats do not match ""
CHANNEL_FORMAT

""
Error: Flow formats do not match,
e.g. Multicast flow with more slots than receiving device can handle
""
BUNDLE_FORMAT

"" Error: Receiver is out of resources (e.g. flows) ""
NO_RX

"" Error: Receiver couldn't set up the flow ""
RX_FAIL

"" Error: Transmitter is out of resources (e.g. flows) ""
NO_TX

"" Error: Transmitter couldn't set up the flow ""
TX_FAIL

"" Error: Receiver got a QoS failure (too much data) when setting up the flow.
""
QOS_FAIL_RX

"" Error: Transmitter got a QoS failure (too much data) when setting up the flow.
""
QOS_FAIL_TX

"" Error: Tx rejected the address given by rx (usually indicates an ARP failure)
""
TX_REJECTED_ADDR

"" Error: Transmitter rejected the bundle request as invalid ""
INVALID_MSG

"" Error: Tx channel latency higher than maximum supported Rx latency ""
CHANNEL_LATENCY

"" Error: Tx and Rx and in different clock subdomains ""
CLOCK_DOMAIN

"" Error: Attempt to use an unsupported feature ""
UNSUPPORTED

"" Error: All Rx links are down ""
RX_LINK_DOWN

"" Error: All Tx links are down ""
TX_LINK_DOWN

"" Error: can't find suitable protocol for dynamic connection ""
DYNAMIC_PROTOCOL

```

```

"" Error: Channel does not exist (eg no such local channel) ""
INVALID_CHANNEL

"" Error: Tx Scheduler failure ""
TX_SCHEDULER_FAILURE

"" Error: The given subscription to self was disallowed by the device ""
SUBSCRIBE_SELF_POLICY

"" Error: There is an external issue with Tx ""
TX_NOT_READY

"" Error: There is an external issue with Rx ""
RX_NOT_READY

"" Error: Tx device cannot support additional unicast flows ""
TX_FANOUT_LIMIT_REACHED

"" Error: Rx device does not support the signal encryption ""
TX_CHANNEL_ENCRYPTED

"" Error: Unexpected response from TX device ""
TX_RESPONSE_UNEXPECTED

"" Error: Template-based subscription failed: template and subscription device names don't match
""
TEMPLATE_MISMATCH_DEVICE

"" Error: Template-based subscription failed: flow and channel formats don't match
""
TEMPLATE_MISMATCH_FORMAT

"" Error: Template-based subscription failed: the channel is not part of the given multicast flow
""
TEMPLATE_MISSING_CHANNEL

""
Error: Template-based subscription failed: something else about the template configuration
made it impossible to complete the subscription using the given flow
""
TEMPLATE_MISMATCH_CONFIG

"" Error: Template-based subscription failed: the unicast template is full
""
TEMPLATE_FULL

"" Error: Tx access control denied the request ""
TX_ACCESS_CONTROL_DENIED

"" Tx access control request is in progress ""
TX_ACCESS_CONTROL_PENDING

"" Error: Rx device does not have a supported subscription mode (unicast/multicast) available
""
RX_UNSUPPORTED_SUB_MODE

"" Error: Tx device does not have a supported subscription mode (unicast/multicast) available
""
TX_UNSUPPORTED_SUB_MODE

"" Error: HDCP key negotiation failed ""
HDCP_NEGOTIATION_FAILED

"" Error: Unexpected system failure ""
SYSTEM_FAIL
    
```

```

}
enum RxChannelSummary {
  NONE
  CONNECTED
  IN_PROGRESS
  WARNING
  ERROR
}
enum Status {
  OK
  WARNING
  ERROR
  UNKNOWN
}
"""A transmit channel of media (either audio or video) from a device"""
type TxChannel {
  id: ID!
  index: Int!
  name: String!
  mediaType: ChannelMediaType
}

```

DDM 1.5 GraphQL Schema Reference

```

type Query {
  """ accounts: Director Only - get all accounts accessible using this API key """
  accounts: [Account]!
  """ account: Director Only - get a single account """
  account(id: ID!): Account!
  """ domains: get all domains (Sites) accessible using this API key """
  domains: [Domain]!
  """ domain: get a single domains (Site) """
  domain(id: ID): Domain
}

type Mutation {
  """ DeviceRxChannelsSubscriptionSet: subscribe or unsubscribe a receiver from a transmitter """
  DeviceRxChannelsSubscriptionSet(input: DeviceRxChannelsSubscriptionSetInput!):
  DeviceRxChannelsSubscriptionSetPayload
  """ DevicesEnroll: enrol an (unenrolled) device into a domain (Site) in this account """
  DevicesEnroll(input: DevicesEnrollInput!): DevicesEnrollPayload
  """ DevicesUnenroll: enrol a device from a domain (Site) and return to unamanged mode """
  DevicesUnenroll(input: DevicesUnenrollInput!): DevicesUnenrollPayload
  """ DeviceNameSet: Director Only - set the friendly name of a device """
  DeviceNameSet(input: DeviceNameSetInput!): DeviceNameSetPayload
}

"""
An Account is a
1. collection of domains
2. roll up of billable features
3. scope for a default role
4. scope for Service API keys (but not User API keys)
"""

```

```

type Account {
  id: ID!
  """
  The name of the account, typically the name of the business unit that pays for the account
  """
  name: String!
  """Domains which are part of this account"""
  domains: [Domain]!
}

""" Domain (also referred to as Site) is a container for enrolled devices """
type Domain {
  devices: [Device]
  device(id: ID!): Device
  id: ID!
  name: String
  status: DomainStatus
  account: Account
}

""" Device is a Dante device that is associated with this account """
type Device {
  id: ID!
  name: String!
  identity: Identity
  domainId: ID
  interfaces: [Interface]
  enrolmentState: EnrolmentState
  location: String
  description: String
  comments: String
  rxChannels: [RxChannel]
  txChannels: [TxChannel]
  domain: Domain
  status: DeviceStatus
}

""" Identity contains identifying attributes of a Device """
type Identity {
  id: ID!
  instanceId: ID!
  """
  The name that the device will have when it is unboxed or factory reset. Typically contains the MAC
  address. Example: 'AVIOA02-51f9e7'
  """
  defaultName: String
  """The name given to the device by the user. Example: 'Foyer'"""
  actualName: String
  """
  The name given to the device (actualName) which may be appended with a numeric identifier if a
  device of the same actualName already exists. Example: 'Foyer (2)'
  """
  productModelId: String
  productModelName: String
  productVersion: String
  productSoftwareVersion: String
  danteVersion: String
  danteHardwareVersion: String
}

""" Interface contains network interface details of a Device """

```

```

type Interface {
  id: ID!
  macAddress: String
  address: String
  netmask: Int
  subnet: String
}

type DeviceConnection {
  id: ID!
  state: ConnectionState
  lastChanged: DateTime
}

enum ConnectionState {
  DISCONNECTED
  ESTABLISHED
  READY
}

""" EnrolmentState of a Device in the account """
enum EnrolmentState {
  UNENROLLED
  ENROLLED
  ENROLLING
  UNENROLLING
  CHANGING_DOMAIN
}

""" RxChannel is a receive channel of a device, which can be subscribed to a transmitter using the
DeviceRxChannelsSubscriptionSet mutation """
type RxChannel {
  """ A unique identifier for the channel, used for caching purposes. Use 'index' when displaying to
users """
  id: ID!

  """
  The channel number, starting at 1.

  Examples:
  - 1
  - 19

  This index is unique on the given device and is not media specific.
  """
  index: Int!
  enabled: Boolean

  """ The name of the channel """
  name: String

  """ The device name of the device that this channel is subscribed to """
  subscribedDevice: String

  """ The Tx channel name on the the device that this channel is subscribed to
  """
  subscribedChannel: String
  status: RxChannelStatus
  statusMessage: String
  summary: RxChannelSummary
  mediaType: ChannelMediaType
}

enum ChannelMediaType {
  AUDIO

```

```

VIDEO
ANCILLARY
UNDEFINED
ALL
GENERIC
}

enum RxChannelStatus {
    "" No subscription for this channel ""
    NONE

    "" Channel Name not yet found on network ""
    UNRESOLVED

    "" Channel Name has been found, but not yet processed. This is an transient state before the flow
is created
    ""
    RESOLVED

    "" Error: an error occurred while trying to resolve the channel name ""
    RESOLVE_FAIL

    "" Channel is successfully subscribed to own TX channels (local loopback mode)
    ""
    SUBSCRIBE_SELF

    "" Error: Channel Name explicitly does not exist on this network ""
    RESOLVED_NONE

    ""
    A flow has been configured but does not have sufficient information to
establish an audio connection.

    For example, configuring a template with no associations.
    ""
    IDLE

    "" Channel Name has been found and processed; setting up flow. This is an transient state
    ""
    IN_PROGRESS

    "" Active subscription to an automatically configured source flow ""
    DYNAMIC

    "" Active subscription to a manually configured source flow ""
    STATIC

    "" Manual flow configuration bypassing the standard subscription process
    ""
    MANUAL

    ""
    Error: The name was found but the connection process failed
(the receiver could not communicate with the transmitter)
    ""
    NO_CONNECTION

    "" Error: Channel formats do not match ""
    CHANNEL_FORMAT

    ""
    Error: Flow formats do not match,
e.g. Multicast flow with more slots than receiving device can handle
    ""
    BUNDLE_FORMAT

```

```
"" Error: Receiver is out of resources (e.g. flows) ""
NO_RX

"" Error: Receiver couldn't set up the flow ""
RX_FAIL

"" Error: Transmitter is out of resources (e.g. flows) ""
NO_TX

"" Error: Transmitter couldn't set up the flow ""
TX_FAIL

"" Error: Receiver got a QoS failure (too much data) when setting up the flow.
""
QOS_FAIL_RX

"" Error: Transmitter got a QoS failure (too much data) when setting up the flow.
""
QOS_FAIL_TX

"" Error: Tx rejected the address given by rx (usually indicates an ARP failure)
""
TX_REJECTED_ADDR

"" Error: Transmitter rejected the bundle request as invalid ""
INVALID_MSG

"" Error: Tx channel latency higher than maximum supported Rx latency ""
CHANNEL_LATENCY

"" Error: Tx and Rx and in different clock subdomains ""
CLOCK_DOMAIN

"" Error: Attempt to use an unsupported feature ""
UNSUPPORTED

"" Error: All Rx links are down ""
RX_LINK_DOWN

"" Error: All Tx links are down ""
TX_LINK_DOWN

"" Error: can't find suitable protocol for dynamic connection ""
DYNAMIC_PROTOCOL

"" Error: Channel does not exist (eg no such local channel) ""
INVALID_CHANNEL

"" Error: Tx Scheduler failure ""
TX_SCHEDULER_FAILURE

"" Error: The given subscription to self was disallowed by the device ""
SUBSCRIBE_SELF_POLICY

"" Error: There is an external issue with Tx ""
TX_NOT_READY

"" Error: There is an external issue with Rx ""
RX_NOT_READY

"" Error: Tx device cannot support additional unicast flows ""
TX_FANOUT_LIMIT_REACHED

"" Error: Rx device does not support the signal encryption ""
TX_CHANNEL_ENCRYPTED
```

```

""" Error: Unexpected response from TX device """
TX_RESPONSE_UNEXPECTED

""" Error: Template-based subscription failed: template and subscription device names don't match
"""
TEMPLATE_MISMATCH_DEVICE

""" Error: Template-based subscription failed: flow and channel formats don't match
"""
TEMPLATE_MISMATCH_FORMAT

""" Error: Template-based subscription failed: the channel is not part of the given multicast flow
"""
TEMPLATE_MISSING_CHANNEL

"""
Error: Template-based subscription failed: something else about the template configuration
made it impossible to complete the subscription using the given flow
"""
TEMPLATE_MISMATCH_CONFIG

""" Error: Template-based subscription failed: the unicast template is full
"""
TEMPLATE_FULL

""" Error: Tx access control denied the request """
TX_ACCESS_CONTROL_DENIED

""" Tx access control request is in progress """
TX_ACCESS_CONTROL_PENDING

""" Error: Rx device does not have a supported subscription mode (unicast/multicast) available
"""
RX_UNSUPPORTED_SUB_MODE

""" Error: Tx device does not have a supported subscription mode (unicast/multicast) available
"""
TX_UNSUPPORTED_SUB_MODE

""" Error: HDCP key negotiation failed """
HDCP_NEGOTIATION_FAILED

""" Error: Unexpected system failure """
SYSTEM_FAIL
}

enum RxChannelSummary {
  NONE
  CONNECTED
  IN_PROGRESS
  WARNING
  ERROR
}

"""A transmit channel of media (either audio or video) from a device"""
type TxChannel {
  id: ID!
  index: Int!
  name: String!
  mediaType: ChannelMediaType
}

""" Status: of a single property of DomainStatus or DeviceStatus """
enum Status {
  OK
  WARNING

```

```

    ERROR
    UNKNOWN
  }

  """ DeviceStatus is a summary of the current status of a device """
  type DeviceStatus {
    id: ID!
    clocking: Status
    connectivity: Status
    latency: Status
    subscriptions: Status
    summary: Status
    alertMessage: AlertMessage
  }

  """ DomainStatus is a summary of the current status of a domain (site) """
  type DomainStatus {
    id: ID!
    clocking: Status
    connectivity: Status
    latency: Status
    subscriptions: Status
    domainAlertMessage: DomainAlertMessages
    summary: Status
  }

  input DevicesEnrollInput {
    domainId: ID!
    deviceIds: [ID!]!

    """
    Clear Configuration will reset the following configuration settings to the device defaults:
    - Device Name
    - Channel labels
    - Latency
    - Sample rate
    - Encoding
    - Subscriptions
    Note: Clear Configuration is not supported for legacy devices.
    """
    clearConfig: Boolean = false
  }

  type DevicesEnrollPayload implements MutationResponse {
    ok: Boolean!
  }

  input DevicesUnenrollInput {
    deviceIds: [ID!]!

    """
    Clear Configuration will reset the following configuration settings to the device defaults:
    - Device Name
    - Channel labels
    - Latency
    - Sample rate
    - Encoding
    - Subscriptions
    Note: Clear Configuration is not supported for legacy devices.
    """
    clearConfig: Boolean = false
  }

  type DevicesUnenrollPayload implements MutationResponse {
    ok: Boolean!
  }

```

```

input DeviceNameSetInput {
    """
    The ID of the device to rename

    Examples:
    "0ae3b2edf1374c0c836c96649e879c2f" (for software devices like DVS)
    "001dc1fffe501c25:0" (for hardware devices like AVIO)
    """
    deviceId: ID!

    """
    New name for the device
    Maximum of 31 characters and must contain only letters, numbers and -.
    First and last characters cannot be -.
    Provide an empty string to reset to default.
    """
    name: String!
}

type DeviceNameSetPayload implements MutationResponse {
    ok: Boolean!
}

input DeviceRxChannelsSubscriptionSetInput {
    """
    The ID of the device which will receive the subscription

    Examples:
    "0ae3b2edf1374c0c836c96649e879c2f" (for software devices like DVS)
    "001dc1fffe501c25:0" (for hardware devices like AVIO)
    """
    deviceId: ID!
    subscriptions: [DeviceRxChannelsSubscriptionInput!]!

    """
    Allows setting a subscription to a device name when that device does not currently exist, typically
    because it is offline.

    Subscriptions to unknown devices may be helpful when restoring a preset.

    Dante resolves subscriptions by name, so this is a valid configuration. However, the subscription
    will remain unresolved until the transmitter device is online.

    If set, 'allowSubscriptionToNonExistentChannel' must also be set since the channel name cannot
    resolve when the device is offline
    """
    allowSubscriptionToNonExistentDevice: Boolean = false

    """
    Allows setting a subscription to a channel name even when the transmitter device does not have a
    channel by that name

    Subscriptions to unknown devices or channels may be helpful when restoring a preset.

    Dante resolves subscriptions by name, so this is a valid configuration. However, the subscription
    will remain unresolved until the transmitter device has a channel by this name.

    If this value is set to true, then additional checks (such as checking the media types of each
    channel match) will not be performed.
    """
    allowSubscriptionToNonExistentChannel: Boolean = false
}

input DeviceRxChannelsSubscriptionInput {
    rxChannelId: Int @deprecated(reason: "Use rxChannelIndex instead")
}
    
```

```

"""
The channel number on the device which will receive the subscription

Examples:
- 1
- 19

Channel numbers start at 1
"""
rxChannelIndex: Int!

"""
The name of the device which is transmitting the media that you wish to subscribe to

Device *names* are used in Dante subscriptions (rather than IDs) so that the the subscription will
follow a device of that name, rather than a specific device of that ID.
This allows the transmitting device to be swapped out for a device of the same name.

Example: "AVIOA02-51f9e7"

An empty subscribedDevice and subscribedChannel will unsubscribe the channel
"""
subscribedDevice: String!

"""
The channel name on the device which is transmitting the media

Channel *labels* are used in Dante (rather than IDs) so that the the subscription will
follow the channel of that label, rather than a specific channel of that ID.
This allows the transmitting device rearrange it's transmit channels,
without needing to update the subscriptions on receiving devices.

Examples:
"Audio L"
"CH1"
"01"

An empty subscribedDevice and subscribedChannel will unsubscribe the channel
"""
subscribedChannel: String!
}

type DeviceRxChannelsSubscriptionSetPayload implements MutationResponse {
  ok: Boolean!
}

input EnrolDevicesInput {
  domainId: ID!
  deviceIds: [ID!]!
}

"""
Clear Configuration will reset the following configuration settings to the device defaults:
- Device Name
- Channel labels
- Latency
- Sample rate
- Encoding
- Subscriptions
Note: Clear Configuration is not supported for legacy devices.
"""
clearConfig: Boolean = false
}

type EnrolDevicesPayload implements MutationResponse {
  ok: Boolean!
}

```

```
}  
  
input UnenrollDevicesInput {  
  deviceIds: [ID!]!  
  
  """  
  Clear Configuration will reset the following configuration settings to the device defaults:  
  - Device Name  
  - Channel labels  
  - Latency  
  - Sample rate  
  - Encoding  
  - Subscriptions  
  Note: Clear Configuration is not supported for legacy devices.  
  """  
  clearConfig: Boolean = false  
}  
  
type UnenrollDevicesPayload implements MutationResponse {  
  ok: Boolean!  
}
```

Index

A

Accessing the API via Dante Director 9
Accessing the API via Dante Domain Manager 6
AES67 8
API Key Creation 6
API Key Generation 9
API Keys 6
API Usage 12

C

Channel Subscription 18
Clearing a Subscription 20
cURL 15, 17, 20

D

Dante Director 9
DDM 1.5 GraphQL Schema Reference 51
DDM 1.7 GraphQL Schema Reference 37
DDM 1.8 GraphQL Schema Reference 22
Device and Subscription Status 16

E

Enabling API Access 6
Enabling API Key Management 6

G

Getting Device and Subscription Status via
cURL 17

K

Key Generation 9

L

Legacy Devices 8
Limitations 8

P

Performing Your First Request 14
Postman Collection 12

R

Requests via cURL 15

S

Setting (or Clearing) a Channel Subscription 18

Setting a Channel Subscription via cURL 20

Setting up a Dante Managed API Postman
Collection 12

SMPTE2110 8

Subscription 18

Subscription Status 16

V

Virtual Devices 8