



Pexip Infinity

Management API

Software Version 33

Document Version 33.a

October 2023

]pexip[

Contents

Introduction	5
About Pexip Infinity	5
About this guide	5
Definitions	5
Intended audience and references	6
Using the management API	7
Obtaining complete API information (schemas)	7
Authentication	7
Examples used in the guide	7
Configuration replication delay	7
API performance	8
API design guidelines and best practices	8
Security	8
Configuration API	9
Configuration resources	9
Resource details	12
Resource methods	12
Getting resource details	12
Creating a single resource object	13
Updating a single resource object	13
Updating multiple resource objects	13
Deleting all resource objects	13
Examples	14
Creating a Virtual Meeting Room, Virtual Auditorium, Virtual Reception, or Test Call Service	14
Getting a Virtual Meeting Room's configuration	15
Changing an existing Virtual Meeting Room	16
Adding a Virtual Meeting Room alias	16
Deleting a Virtual Meeting Room	16
Creating a Virtual Meeting Room with aliases	17
Getting all Virtual Meeting Rooms, Virtual Auditoriums and Virtual Receptions	17
Getting all Virtual Meeting Rooms only	17
Creating multiple Virtual Meeting Rooms	18
Deleting multiple Virtual Meeting Rooms	18
Creating a Virtual Auditorium	18
Creating a Virtual Reception	19
Creating Automatically Dialed Participants	19
Modifying an Automatically Dialed Participant	19
Removing an Automatically Dialed Participant from a Virtual Meeting Room	20
Deleting an Automatically Dialed Participant	20
Creating Call Routing Rules	20
Deleting a Call Routing Rule	21

Downloading a Conferencing Node for deployment	21
Deploying a Conferencing Node using a VM template and configuration file	22
Uploading a service theme	23
Returning a license	23
Adding a license	24
Using the Google Meet Gateway Token Management API	24
Status API	26
Status resources	26
Specifying the object ID in the path	27
Resource details	27
Resource methods	27
Pagination and filtering	28
Examples	28
Getting all active conference instances	28
Getting all active Virtual Meeting Room conferences	28
Getting all participants for a conference	29
Getting the media statistics for a participant	31
Getting the status of a Conferencing Node	32
Getting the load for a system location	32
Getting all registered aliases	32
Listing all cloud overflow Conferencing Nodes	32
Listing all locations monitored for dynamic bursting	33
Listing all locations containing dynamic bursting Conferencing Nodes	34
History API	35
History resources	35
Resource details	35
Resource methods	35
Pagination and filtering	36
Examples	36
Getting all conference instances	36
Getting all participants for a conference instance	36
Getting all participants for a time period	39
Getting all participants with packet loss	39
Command API	40
Command resources	40
Resource details	41
Resource methods	41
Response format	41
Examples	42
Dialing a participant into a conference	42
Disconnecting a participant	42
Muting a participant	43
Muting all Guest participants	43
Unmuting a participant	43

Unmuting all Guest participants	43
Locking a conference instance	44
Unlocking a conference instance	44
Unlocking a participant	44
Transferring a participant	44
Changing a participant's role	45
Changing a conference's layout	45
Creating a system backup	45
Restoring a system backup	45
Starting an overflow Conferencing Node	46
Taking a system snapshot	47
Retrieving, paginating, filtering and ordering resource details	48
Dealing with resources	48
Getting a single resource object	49
Getting multiple resource objects	49
Global settings	50
Pagination	50
Filtering	50
Ordering	51
Using the API with SNMP	52
Examples	52
Retrieving the SNMP sysName	52
Retrieving CPU load average	52

Introduction

About Pexip Infinity

Pexip Infinity is a self-hosted, virtualized and distributed multipoint conferencing platform. It can be deployed in an organization's own datacenter, or in a private or public cloud such as Microsoft Azure, Amazon Web Services (AWS), Google Cloud Platform (GCP) or Oracle Cloud Infrastructure, as well as in any hybrid combination. It enables scaling of video, voice and data collaboration across organizations, enabling everyone to engage in high definition video, web, and audio conferencing.

It provides any number of users with their own personal Virtual Meeting Rooms (VMRs), which they can use to hold conferences, share presentations, and chat. Participants can join over audio or video from any location using the endpoint or client of their choice, including:

- Professional video conferencing room systems (SIP and H.323 devices)
- Desktop/mobile (with the Pexip Connect app suite of clients)
- Web browsers (WebRTC - no downloads required)
- Traditional audio conferencing (PSTN dialing)

About this guide

Pexip Infinity includes a management API that allows third parties to control, configure, and obtain status information on the Pexip Infinity platform. Typical tasks include:

- dynamic creation of Virtual Meeting Rooms
- controlling outbound calls to endpoints and recording solutions
- controlling conference participants
- deploying new Conferencing Nodes
- real-time and historic status monitoring

This guide describes how to use the Pexip Infinity version 33 API. It comprises the following sections:

- [Using the management API](#): an overview of the RESTful web API used to communicate with a Pexip InfinityManagement Node.
- [Configuration API](#): how to configure the Pexip Infinity platform and services (Virtual Meeting Rooms, Virtual Auditoriums, Virtual Receptions and the Infinity Gateway).
- [Status API](#): how to obtain status information on the Pexip Infinity platform and services.
- [History API](#): how to obtain historical information on the Pexip Infinity platform and services.
- [Command API](#): how to control aspects of Pexip Infinity conference instances.
- [Retrieving, paginating, filtering and ordering resource details](#): how to control which resource details are retrieved.

Definitions

In the context of this API guide:

- A **service** is a Virtual Meeting Room, Virtual Auditorium, Virtual Reception or Infinity Gateway.
- A **conference instance** is a unique conference that is created when one or more participants accesses a **service**, and exists only until the last participant leaves.

Intended audience and references

It is assumed that readers are familiar with the concepts of HTTP(S), JSON and REST. The following links provide more information about the technologies used in relation to the management API.

HTTP(S)

- http://en.wikipedia.org/wiki/HTTP_Secure
- <https://tools.ietf.org/html/rfc2616>
- <https://tools.ietf.org/html/rfc2818>

JSON

- <http://en.wikipedia.org/wiki/JSON>
- <http://www.json.org/>
- <https://tools.ietf.org/html/rfc4627>

REST

- http://en.wikipedia.org/wiki/Representational_state_transfer
- <http://www.ibm.com/developerworks/webservices/library/ws-restful/>

Python

- [http://en.wikipedia.org/wiki/Python_\(programming_language\)](http://en.wikipedia.org/wiki/Python_(programming_language))
- <http://www.python.org/about/gettingstarted/>
- <http://docs.python-requests.org/en/latest/>
- <http://pysnmp.sourceforge.net>

Using the management API

This section describes how to use the Pexip Infinity management API.

Obtaining complete API information (schemas)

A summary of the schemas is available from within the Management Node web interface via <https://<manageraddress>/admin/platform/schema/>. If you require the schemas but do not have access to a Management Node running v33, please contact your Pexip authorized support representative.

Full information for each configuration, command and resource is available from within the Management Node by downloading the resource's schema. The information in each schema includes, where applicable:

- all the available fields
- whether each field is optional or required
- the type of data each field must contain
- any restrictions on the minimum or maximum length of each field
- help text with information on usage

See the [Command API](#), [Configuration API](#) and [Status API](#) sections for a list of all resources and how to access the schemas for each.

Authentication

All access to the management API is authenticated over HTTPS.

If you are not using an LDAP database for authentication, access is via the credentials for the web admin user. The default username for this account is `admin`.

If you are using an LDAP database, we recommend you create an account specifically for use by the API.

Examples used in the guide

This guide uses Python 3 and the requests library (<https://pypi.python.org/pypi/requests>) for examples. However, many other modern programming languages and libraries exist which could be used equally successfully for driving the Pexip Infinity RESTful APIs.

The examples used are self-contained programs rather than pseudo code. These are given in an effort to help illustrate programming possibilities.

The following example data is used in the examples:

`<manageraddress>`: the IP address or FQDN of the Management Node.

`<password1>`: the admin web password for accessing the Management Node.

`<user1>`: the admin web username for accessing the Management Node. On a default installation this is `admin`.

Configuration replication delay

It may take some time for any configuration changes to take effect across the Conferencing Nodes. In typical deployments, configuration replication is performed approximately once per minute. However, in very large deployments (more than 60 Conferencing Nodes), configuration replication intervals are extended, and it may take longer for configuration changes to be applied to all Conferencing Nodes (the administrator log shows when each node has been updated).

API performance

Accessing the REST API on the Management Node requires that the Management Node lock access to its database to ensure that configuration remains consistent and coherent. This therefore limits the rate at which requests can be serviced (to 1,000 requests every 60 seconds).

When configuration is modified, the modified configuration is replicated to the various Conferencing Nodes. This means that configuration changes cause some CPU usage on both the Management Node and the Conferencing Nodes.

Heavy or frequent API usage will consume resources across the Pexip Infinity deployment.

API design guidelines and best practices

We recommend you follow these guidelines and best practices when designing your application, to minimize resource usage and network latency when using the API:

- Use bulk requests, rather than individual requests. For example, the PATCH method can be used to create or update multiple objects at once. For more information, see [Updating multiple resource objects](#); for an example of this in use, see [Creating multiple Virtual Meeting Rooms](#).
- Avoid duplication. Avoid designs that perform large amounts of polling for information that is not needed, or that result in repeatedly writing the same configuration values.
- Consider application designs that avoid the need to make large numbers of requests concurrently. Although concurrent API requests will work they may not be processed in the order that they commence. Consider waiting for a request to complete before making another one.
- In cases where multiple REST requests will be made sequentially in quick succession, consider using HTTP 1.1 connection keep alives and reusing your HTTPS connection in order to avoid the not insignificant handshake overhead of re-establishing separate HTTPS connections for every request.
- Use a client with a TCP connection pool, to avoid repeatedly setting up TLS connections.
- Be aware that other systems and applications in your deployment could also require access to the Management API, so consume the API as little as possible to ensure that they can also make use of it.
- In situations where the Pexip Infinity is heavily loaded, API performance may be impaired. We therefore do **not** recommend using the API for real-time call admission control.
- Very heavy usage of the API might cause the web-based Administrator interface performance to be impeded. Conversely, heavy Administrator interface usage might impair API performance.
- If you are using the [Status API](#) heavily, consider using [Event sinks](#) instead, to reduce the load on the Management Node.
- All API URL paths must end with a / character. For example: `https://<manageraddress>/api/admin/history/v1/participant/`
If the API URL path is appended with a query parameter, then the URL path does not require a / character at the end of the query parameter. For example: `https://<manageraddress>/api/admin/configuration/v1/conference_alias/?offset=20`

Security

We recommend that you upload a HTTPS certificate to your Management Node and each Conferencing Node, and that client applications verify those certificates when connecting to those servers.

Configuration API

Configuration of the Pexip Infinity platform and services can be performed through the REST API, allowing you to create, view, edit and delete items.

Configuration resources

A summary of the schema for configuration resources is available from the:

- Management Node web interface via <https://<manageraddress>/admin/platform/schema/>
- REST API via <https://<manageraddress>/api/admin/configuration/v1/>

The following configuration resources are available via the REST API:

Component	Path
System configuration	
DNS server	/api/admin/configuration/v1/dns_server/
NTP server	/api/admin/configuration/v1/ntp_server/
Web proxy server	/api/admin/configuration/v1/http_proxy/
Syslog server	/api/admin/configuration/v1/syslog_server/
SNMP NMS	/api/admin/configuration/v1/snmp_network_management_system/
SMTP server	/api/admin/configuration/v1/smtp_server/
Static route	/api/admin/configuration/v1/static_route/
Exchange servers †	/api/admin/configuration/v1/ms_exchange_connector/
Exchange domains †	/api/admin/configuration/v1/exchange_domain/
Event sinks	/api/admin/configuration/v1/event_sink/
One-Touch Join Endpoints	/api/admin/configuration/v1/mjx_endpoint/
One-Touch Join Endpoint Groups	/api/admin/configuration/v1/mjx_endpoint_group/
One-Touch Join Exchange Autodiscover URLs	/api/admin/configuration/v1/mjx_exchange_autodiscover_url/
One-Touch Join Exchange Integrations	/api/admin/configuration/v1/mjx_exchange_deployment/
One-Touch Join O365 Graph Integrations	/api/admin/configuration/v1/mjx_graph_deployment/
One-Touch Join Google Workspace Integrations	/api/admin/configuration/v1/mjx_google_deployment/
One-Touch Join Profiles	/api/admin/configuration/v1/mjx_integration/
One-Touch Join Meeting Processing Rules	/api/admin/configuration/v1/mjx_meeting_processing_rule/
Platform configuration	
System location	/api/admin/configuration/v1/system_location/
Management Node	/api/admin/configuration/v1/management_vm/

Component	Path
Conferencing Node	/api/admin/configuration/v1/worker_vm/
Licensing	/api/admin/configuration/v1/licence/
License request	/api/admin/configuration/v1/licence_request/
CA certificate	/api/admin/configuration/v1/ca_certificate/
TLS certificate	/api/admin/configuration/v1/tls_certificate/
Certificate signing request (CSR)	/api/admin/configuration/v1/certificate_signing_request/
Global settings	/api/admin/configuration/v1/global/
Diagnostic graphs	/api/admin/configuration/v1/diagnostic_graphs/
Call control	
H.323 Gatekeeper	/api/admin/configuration/v1/h323_gatekeeper/
SIP credentials	/api/admin/configuration/v1/sip_credential/
SIP proxy	/api/admin/configuration/v1/sip_proxy/
Microsoft SIP proxy	/api/admin/configuration/v1/mssip_proxy/
TURN server	/api/admin/configuration/v1/turn_server/
STUN server	/api/admin/configuration/v1/stun_server/
Policy server	/api/admin/configuration/v1/policy_server/
Google Meet access token	/api/admin/configuration/v1/gms_access_token/
Google Meet gateway token	/api/admin/configuration/v1/gms_gateway_token/
Microsoft Azure tenant	/api/admin/configuration/v1/azure_tenant/
Microsoft Teams Connector	/api/admin/configuration/v1/teams_proxy/
Epic telehealth profile	/api/admin/configuration/v1/telehealth_profile/
Break-in resistance allow list	/api/admin/configuration/v1/break_in_allow_list_address/
Service configuration	
Virtual Meeting Room, Virtual Auditorium, Virtual Reception, scheduled conference, Media Playback Service, or Test Call Service	/api/admin/configuration/v1/conference/
Alias for a service	/api/admin/configuration/v1/conference_alias/
Automatically Dialed Participant	/api/admin/configuration/v1/automatic_participant/
IVR theme	/api/admin/configuration/v1/ivr_theme/
Gateway routing rule	/api/admin/configuration/v1/gateway_routing_rule/
Registration settings	/api/admin/configuration/v1/registration/
Device	/api/admin/configuration/v1/device/
Conference sync template	/api/admin/configuration/v1/conference_sync_template/

Component	Path
Conference LDAP sync source	/api/admin/configuration/v1/ldap_sync_source/
Conference LDAP sync field	/api/admin/configuration/v1/ldap_sync_field/
Recurring conference †	/api/admin/configuration/v1/recurring_conference/
Scheduled conference †	/api/admin/configuration/v1/scheduled_conference/
Scheduled conference aliases †	/api/admin/configuration/v1/scheduled_alias/
Media Playback Service library entry	/api/admin/configuration/v1/media_library_entry/
Media Playback Service playlist	/api/admin/configuration/v1/media_library_playlist/
Media Playback Service playlist entry	/api/admin/configuration/v1/media_library_playlist_entry/
Users	
Authentication	/api/admin/configuration/v1/authentication/
Account role	/api/admin/configuration/v1/role/
LDAP role	/api/admin/configuration/v1/ldap_role/
Permission	/api/admin/configuration/v1/permission/
AD FS servers	/api/admin/configuration/v1/adfs_auth_server/
AD FS domains	/api/admin/configuration/v1/adfs_auth_server_domain/
Google OAuth 2.0 Credential	/api/admin/configuration/v1/google_auth_server/
Google OAuth domains	/api/admin/configuration/v1/google_auth_server_domain/
Users	/api/admin/configuration/v1/end_user/
Identity Providers	/api/admin/configuration/v1/identity_provider/
Identity Provider Groups	/api/admin/configuration/v1/identity_provider_group/
User groups*	/api/admin/configuration/v1/user_group/
User group mappings*	/api/admin/configuration/v1/user_group_entity_mapping/
Web app	
Web app path	/api/admin/configuration/v1/webapp_alias/
Web app branding package	/api/admin/configuration/v1/webapp_branding/
Utilities	
Upgrade	/api/admin/configuration/v1/upgrade/
Software bundle	/api/admin/configuration/v1/software_bundle/
Software bundle revision	/api/admin/configuration/v1/software_bundle_revision/
System backup	/api/admin/configuration/v1/system_backup/
Automatic backup	/api/admin/configuration/v1/autobackup/
Teams scheduled scaling	/api/admin/configuration/v1/scheduled_scaling/

* This is new in version 33.

† We do not currently recommend creation, deletion or modification of these resources directly; this should only be done by the VMR Scheduling for Exchange service.

Resource details

More information can be obtained for each resource by downloading the resource schema in a browser. You may want to install a JSON viewer extension to your browser in order to view the JSON strings in a readable format.

For example, to view the schema for **aliases** the URI would be:

```
https://<manageraddress>/api/admin/configuration/v1/conference_alias/schema/?format=json
```

Each schema contains information on the available fields including:

- whether the field is optional (`nullable: true`) or required (`nullable: false`)
- the default value
- the type of data the field must contain
- the choices for the field, e.g. `valid_choices: ["audio", "video", "video-only"]`
- which criteria can be used for [filtering](#) and [ordering](#) searches
- help text with additional information on usage.

Resource methods

Each configuration resource supports the following HTTP methods:

Method	Action
GET	Retrieves the current configuration for a resource
POST	Creates a new object for the resource
PATCH	Updates an existing resource object, or creates the object if it does not already exist
DELETE	Deletes an existing resource object

Getting resource details

See [Retrieving, paginating, filtering and ordering resource details](#) for information about how to retrieve the current configuration for a resource.

Creating a single resource object

To create a new object resource, a POST request is submitted to the root URI for the resource. The data should be a JSON object whose attributes match the field values for the resource.

Fields with default values may be omitted from the data.

The response to the POST method contains a Location header which contains the REST URI of the newly created resource.

For example, the URI of a VMR resource takes the format: `/api/admin/configuration/v1/conference/<object ID>/`

Note that Pexip Infinity always allocates a new object ID when creating a new resource.

Updating a single resource object

To update a single resource object, a PATCH request is submitted to the URI created by concatenating the object ID with the root URI of the resource.

For example, to make changes to the alias with ID **1** the URI would be:

```
https://<manageraddress>/api/admin/configuration/v1/conference_alias/1/
```

Updating multiple resource objects

The PATCH method can be used to create, update or delete multiple objects at once. To update multiple objects a PATCH request is submitted to the root URI for the resource. The data must be formatted as a JSON object with a single attribute `objects` whose value is a list of the new objects.

Note: for any objects in the list that already exist, the `resource_uri` field must be included in the request. For any objects that are to be created by the PATCH request, the `resource_uri` field should be omitted.

Deleting all resource objects

If a DELETE operation is invoked on the root resource URI, then all the resource objects will be deleted. You should therefore use this operation with caution, and ensure you include the resource ID in the URI unless your intention actually is to delete all objects.

For example, to delete ALL Virtual Meeting Rooms, the URI is:

```
"https://<manageraddress>/api/admin/configuration/v1/conference/"
```

whereas to delete the single Virtual Meeting Room with ID **1** only, the URI is:

```
"https://<manageraddress>/api/admin/configuration/v1/conference/1/"
```

Examples

Creating a Virtual Meeting Room, Virtual Auditorium, Virtual Reception, or Test Call Service

To create a new Virtual Meeting Room, Virtual Auditorium, Virtual Reception, or Test Call Service, you must submit a POST to the URI for this resource. You must specify the `service_type` as follows:

- `conference` for a Virtual Meeting Room
- `lecture` for a Virtual Auditorium
- `two_stage_dialing` for a Virtual Reception
- `media_playback` for a Media Playback Service
- `test_call` for a Test Call Service

If you do not specify a `service_type`, the service will by default be created as a Virtual Meeting Room.

The following example creates a new Virtual Meeting Room with the name **VMR_1**. It has no aliases.

```
import json
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/conference/",
    auth=('<user1>', '<password1>'),
    verify=True,
    json={'name': 'VMR_1', 'service_type': 'conference'})
print("Created new Virtual Meeting Room:", response.headers['location'])
```

Getting a Virtual Meeting Room's configuration

By submitting a GET to the resource URI of a Virtual Meeting Room, you can get its configuration:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/configuration/v1/conference/1/",
    auth=('<user1>', '<password1>'),
    verify=True
)
print("Virtual Meeting Room:", response.json())
```

Example output

```
Virtual Meeting Room: {
  'aliases': [
    {
      'alias': 'meet@example.com',
      'conference': '/api/admin/configuration/v1/conference/1/',
      'creation_time': '2017-05-11T22:02:05.855877',
      'description': '',
      'id': 1
    }
  ]
  'allow_guests': False,
  'automatic_participants': [],
  'call_type': 'video',
  'creation_time': '2017-05-11T22:02:05.848612',
  'crypto_mode': '',
  'description': '',
  'enable_active_speaker_indication': false,
  'enable_chat': 'default',
  'enable_overlay_text': false,
  'force_presenter_into_main': false,
  'guest_identity_provider_group': '',
  'guest_pin': '',
  'guest_view': None,
  'guests_can_present': true,
  'host_identity_provider_group': '',
  'host_view': 'one_main_seven_pips',
  'id': 1,
  'ivr_theme': None,
  'match_string': '',
  'max_callrate_in': None,
  'max_callrate_out': None,
  'max_pixels_per_second': None,
  'mssip_proxy': null,
  'mute_all_guests': false,
  'name': 'VMR_1',
  'non_idp_participants': 'disallow_all',
  'participant_limit': None,
  'pin': '',
  'primary_owner_email_address': '',
  'replace_string': '',
  'resource_uri': '/api/admin/configuration/v1/conference/1/',
  'service_type': 'conference',
  'sync_tag': '',
  'system_location': null,
  'tag': ''
}
```

Changing an existing Virtual Meeting Room

Submitting a PATCH to an existing Virtual Meeting Room URI allows the data for that Virtual Meeting Room to be modified.

The following example updates the Virtual Meeting Room PIN to **1234**:

```
import json
import requests
response = requests.patch(
    "https://<manageraddress>/api/admin/configuration/v1/conference/1/",
    auth=('<user1>', '<password1>'),
    verify=True,
    json={'pin': '1234'}
)
```

Adding a Virtual Meeting Room alias

A Virtual Meeting Room must already exist before you can add an alias to it. To do this you submit a POST to the resource URI for Virtual Meeting Room aliases and add the partial URI of the Virtual Meeting Room to the data POSTed.

The following example creates a new alias **meet@example.com** for the Virtual Meeting Room with ID **1**.

```
import json
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/conference_alias/",
    auth=('<user1>', '<password1>'),
    verify=True,
    json={
        'alias': 'meet@example.com',
        'conference': '/api/admin/configuration/v1/conference/1/',
    }
)
print("Created new alias:", response.headers['location'])
```


Note that the request is rejected if the alias already exists.

Deleting a Virtual Meeting Room

Deleting a Virtual Meeting Room is achieved by submitting a DELETE request to an existing Virtual Meeting Room URI.

The following example deletes the Virtual Meeting Room with ID **1**:

```
import requests
response = requests.delete(
    "https://<manageraddress>/api/admin/configuration/v1/conference/1/",
    auth=('<user1>', '<password1>'),
    verify=True
)
```

 Deleting a Virtual Meeting Room will also delete all aliases associated with it.

Creating a Virtual Meeting Room with aliases

To simplify the creation of a Virtual Meeting Room and the aliases that belong to it, you can submit a single POST with all the information.

The following example creates a new Virtual Meeting Room with the name **VMR_1** and two aliases: **meet** and **meet@example.com**:

```
import json
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/conference/",
    auth=(<user1>, <password1>),
    verify=True,
    json={
        'name': 'VMR_1',
        'service_type': 'conference',
        'aliases': [{ 'alias' : 'meet'}, { 'alias' : 'meet@example.com'}]
    }
)
print("Created new Virtual Meeting Room:", response.headers['location'])
```

Note that if there is an existing VMR with the same alias, that alias is removed from the previous VMR and assigned to the new VMR.

Getting all Virtual Meeting Rooms, Virtual Auditoriums and Virtual Receptions

Retrieving all the configured Virtual Meeting Rooms, Virtual Auditoriums and Virtual Receptions is achieved by submitting a GET request to the resource URI they all share:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/configuration/v1/conference/",
    auth=(<user1>, <password1>),
    verify=True
)
print("Virtual Meeting Rooms:", response.json()['objects'])
```

Getting all Virtual Meeting Rooms only

To retrieve all the configured Virtual Meeting Rooms but not Virtual Auditoriums or Virtual Receptions, you submit a GET request to the resource URI as above but filter it by `service_type`:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/configuration/v1/conference/?service_type=conference",
    auth=(<user1>, <password1>),
    verify=True
)
print("Virtual Meeting Rooms:", response.json()['objects'])
```

Creating multiple Virtual Meeting Rooms

Multiple Virtual Meeting Rooms can be created using a PATCH request.

The following example creates two Virtual Meeting Rooms; the first with the name **VMR_1** and an alias **meet1@example.com**, and the second with the name **VMR_2** and an alias **meet2@example.com**:

```
import json
import requests
data = { 'objects' : [
    { 'name' : 'VMR_1', 'service_type': 'conference', 'aliases' : [{ 'alias' : 'meet1@example.com' } ] },
    { 'name' : 'VMR_2', 'service_type': 'conference', 'aliases' : [{ 'alias' : 'meet2@example.com' } ] },
  ] }
response = requests.patch(
    "https://<manageraddress>/api/admin/configuration/v1/conference/",
    auth=(<user1>, <password1>),
    verify=True,
    json=data
)
```

Deleting multiple Virtual Meeting Rooms

Multiple Virtual Meeting Rooms can be deleted using a PATCH request.

The following example deletes four Virtual Meeting Rooms. The `deleted_objects` list should refer to existing VMR URIs; in this example it refers to the VMRs with IDs of **5**, **46**, **47** and **65**.

```
import json
import requests
data = {
    'deleted_objects' : [
        "/api/admin/configuration/v1/conference/5/",
        "/api/admin/configuration/v1/conference/46/",
        "/api/admin/configuration/v1/conference/47/",
        "/api/admin/configuration/v1/conference/65/",
    ],
    "objects": []
}
response = requests.patch(
    "https://<manageraddress>/api/admin/configuration/v1/conference/",
    auth=(<user1>, <password1>),
    verify=True,
    json=data
)
```

Creating a Virtual Auditorium

The following example creates a new Virtual Auditorium with the name **Lecture** and a single alias **lecture@example.com**:

```
import json
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/conference/",
    auth=(<user1>, <password1>),
    verify=True,
    json={
        'name': 'Lecture',
        'service_type': 'lecture',
        'aliases': [{ 'alias' : 'lecture@example.com' } ]
    }
)
print("Created new Virtual Auditorium:", response.headers['location'])
```

Creating a Virtual Reception

The following example creates a new Virtual Reception with the name **Reception** and a single alias **reception@example.com**:

```
import json
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/conference/",
    auth=(<user1>, <password1>),
    verify=True,
    json={
        'name': 'Reception',
        'service_type': 'two_stage_dialing',
        'aliases': [{'alias': 'reception@example.com'}]
    }
)
print("Created new Virtual Reception:", response.headers['location'])
```

Creating Automatically Dialed Participants

You can create a new Automatically Dialed Participant by submitting a POST request to the resource URI for Automatically Dialed Participants. Automatically dialed participants are a little different from other resources as they may be associated with many different Virtual Meeting Rooms and Virtual Auditoriums.

The following example creates an Automatically Dialed Participant and associates them with an already created Virtual Meeting Room with ID 1:

```
import json
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/automatic_participant/",
    auth=(<user1>, <password1>),
    verify=True,
    json={
        'alias': 'myendpoint@mydomain.com',
        'remote_display_name': 'My Name',
        'description': "Dial myendpoint@mydomain.com whenever a related conference starts",
        'protocol': 'sip',
        'role': 'guest',
        'conference': ['/api/admin/configuration/v1/conference/1/']
    }
)
print("Created new automatically dialed participant:", response.headers['location'])
```

Modifying an Automatically Dialed Participant

The following examples use PATCH to modify an existing Automatically Dialed Participant.

The example below associates the Automatically Dialed Participant (created above) with two (already created) Virtual Meeting Rooms — one with ID 1, one with ID 2:

```
import json
import requests
response = requests.patch(
    "https://<manageraddress>/api/admin/configuration/v1/automatic_participant/1/",
    auth=(<user1>, <password1>),
    verify=True,
    json={
        'conference': ['/api/admin/configuration/v1/conference/1/',
                       '/api/admin/configuration/v1/conference/2/']
    }
)
print("added existing automatic participant to an additional meeting room:", response)
```

Removing an Automatically Dialed Participant from a Virtual Meeting Room

The example below removes the association between the Automatically Dialed Participant and the Virtual Meeting Rooms:

```
import json
import requests
response = requests.patch(
    "https://<manageraddress>/api/admin/configuration/v1/automatic_participant/1/",
    auth=(<user1>, <password1>),
    verify=True,
    json={
        'conference' : []
    }
)
print("Removed automatic participant from all meeting rooms:", response)
```

Deleting an Automatically Dialed Participant

The following example deletes the Automatically Dialed Participant with ID 1:

```
import requests
response = requests.delete(
    "https://<manageraddress>/api/admin/configuration/v1/automatic_participant/1/",
    auth=(<user1>, <password1>),
    verify=True
)
```

Creating Call Routing Rules

To create a new Call Routing Rule you submit a POST to the URI for that resource.

The following example creates a Call Routing Rule with the name **Route to Teams**, which routes incoming Infinity Gateway calls (via H.323 or SIP) to Microsoft Teams:

```
import json
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/gateway_routing_rule/",
    auth=(<user1>, <password1>),
    verify=True,
    json={
        'name': 'Route to Teams',
        'priority': 50,
        'match_incoming_calls': True,
        'match_outgoing_calls': False,
        'match_incoming_h323': True,
        'match_incoming_mssip': False,
        'match_incoming_sip': True,
        'match_incoming_webrtc': False,
        'match_string': '(\d{9,12}) (@example\.com)?',
        'replace_string': '\1',
        'called_device_type': 'teams_conference',
        'outgoing_protocol': 'teams',
    }
)
print("Created new Call Routing Rule:", response.headers['location'])
```

This example creates a Call Routing Rule that applies to outbound calls made from Pexip VMRs and routes them to registered devices only:

```
import json
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/gateway_routing_rule/",
    auth=(<user1>, <password1>),
    verify=True,
    json={
        'name': 'Outbound calls to registered devices',
        'priority': 60,
        'match_incoming_calls': False,
        'match_outgoing_calls': True,
        'match_string': '.*@mycompany\.com',
        'replace_string': '',
        'called_device_type': 'registration',
    }
)
print("Created new Call Routing Rule:", response.headers['location'])
```

Deleting a Call Routing Rule

The following example deletes the Call Routing Rule with ID 1:

```
import requests
response = requests.delete(
    "https://<manageraddress>/api/admin/configuration/v1/gateway_routing_rule/1/",
    auth=(<user1>, <password1>),
    verify=True
)
```

Downloading a Conferencing Node for deployment

By submitting a POST request specifying a `deployment_type` from the list below, you can download an OVA (or ZIP file for Hyper-V) which can be used to deploy a new Proxying Edge Node onto the appropriate host server:

- MANUAL-ESXI6_7
- MANUAL-ESXI6
- MANUAL-HYPERV2012
- MANUAL-KVM

```
import json
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/worker_vm/",
    auth=(<user1>, <password1>),
    verify=True,
    stream=True,
    json={
        'name': 'new_node',
        'hostname': 'newnode',
        'domain': 'example.test',
        'address': <newnode_ip_address>,
        'netmask': <newnode_ip_mask>,
        'gateway': <ip_gateway_address>,
        'password': <newnode_password>,
        'node_type': 'PROXYING',
        'system_location': '/api/admin/configuration/v1/system_location/1/',
        'deployment_type': 'MANUAL-ESXI6_7',
        'vm_cpu_count': '8',
        'vm_system_memory': '16384',
    }
)
with open('conferencing_node.ova', 'wb') as handle:
    for chunk in response.iter_content(10*1024):
        handle.write(chunk)
print("Downloaded Conferencing Node OVA: conferencing_node.ova")
```

Deploying a Conferencing Node using a VM template and configuration file

You can use a `deployment_type` of `MANUAL-PROVISION-ONLY` to create a VM template for deployment onto unsupported hypervisors or orchestration layers.

```
import json
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/worker_vm/",
    auth=(<user1>, <password1>),
    verify=True,
    stream=True,
    json={
        'name': 'new_node',
        'hostname': 'newnode',
        'domain': 'example.test',
        'address': <newnode_ip_address>,
        'netmask': <newnode_ip_mask>,
        'gateway': <ip_gateway_address>,
        'password': <newnode_password>,
        'node_type': 'CONFERENCING',
        'system_location': '/api/admin/configuration/v1/system_location/1/',
        'deployment_type': 'MANUAL-PROVISION-ONLY',
    }
)
with open('conferencing_node.xml', 'wb') as handle:
    for chunk in response.iter_content(10*1024):
        handle.write(chunk)
print("Downloaded Conferencing Node provisioning document: conferencing_node.xml")
```

After the provisioning document has been obtained from the management API as above, it may be injected into the Conferencing Node to be provisioned as follows:

```
import requests
with open('conferencing_node.xml', 'rb') as handle:
    document = handle.read()
    response = requests.post(
        "https://<conferencingnodeaddress>:8443/configuration/bootstrap",
        verify=True,
        headers={'Content-Type': 'text/xml'},
        data=document,
    )
    if response.status_code == requests.codes.ok:
        print("Successfully provisioned Conferencing Node")
```

Note that this API is available only on Conferencing Nodes created using the `MANUAL-PROVISION-ONLY` deployment type.

Uploading a service theme

The following example will create a new theme, upload the theme contents and then configure a VMR to use the theme:

```
import requests
import json
from urllib.parse import urlsplit

response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/ivr_theme/",
    auth=("<user1>", "<password1>"),
    json={"name": "New theme"},
    verify=True,
)
theme_uri = response.headers["location"]
theme_path = urlsplit(theme_uri).path
response = requests.patch(
    theme_uri,
    auth=("<user1>", "<password1>"),
    files={"package": open("test_theme.zip", "rb")},
    verify=True,
)
response = requests.patch(
    "https://<manageraddress>/api/admin/configuration/v1/conference/1/",
    auth=("<user1>", "<password1>"),
    json={"ivr_theme": theme_path},
    verify=True,
)
```

Returning a license

To return a license, send a DELETE request using the format:

```
import requests
response = requests.delete(
    "https://<manageraddress>/api/admin/configuration/v1/licence/<fulfillment_id>",
    auth=("<user1>", "<password1>"),
    verify=True
)
```

where:

- the fulfillment ID is a 10 digit number
- you can force offline mode by adding `?offline_mode=True` to the end of the URI, in which case it will return with a 202 response and a Location header telling you where to POST the response document.

Adding a license

To add a license entitlement key, send a POST request using the format:

```
import json
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/licence/",
    auth=('<user1>', '<password1>'),
    verify=True,
    json={
        'entitlement_id' : '<entitlement key>'
    }
)
print("Added license entitlement key:", response.headers['location'])
```

where:

- the entitlement key is in the form XXXX-XXXX-XXXX-XXXX (where X is a hex digit [0-9A-F]) or is in an encoded format such as Srf1FPcyRjitSHOhtU_XGQ==
- you can force offline mode by adding 'offline_mode' : True to the JSON data, in which case it will return with a 202 response and a Location header telling you where to GET the request file from, and also where to POST the response document.

To export an offline/stored license request file, you can use:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/configuration/v1/licence_request/<request_id>/export",
    auth=('<user1>', '<password1>'),
    verify=True
)
print("License request:", response.json()['actions'])
```

where the URL to GET the file from is the one returned in the Location header of the response to the initial activation request.

To upload a license response file for a stored license request:

```
import requests
with open('response.xml', 'r') as response_file:
    response = requests.post(
        "https://<manageraddress>/api/admin/configuration/v1/licence_request/<request_id>",
        auth=('<user1>', '<password1>'),
        verify=True,
        files={'response_xml': ('response.xml', response_file.read())})
    print(response.status_code)
```

where:

- 'response_xml' is the path to the file containing the response data
- in this example the response file is called response.xml and is in the current directory; depending on the type of activation key this could also be a .json file
- the URL to POST to is the one returned in the Location header of the response to the initial activation request.

Using the Google Meet Gateway Token Management API

All gateway token requests use the same endpoint: https://<manageraddress>/api/admin/configuration/v1/gms_gateway_token/1/

Generating a CSR

Send a GET request to the gateway token endpoint to receive a JSON response with a key `csr` that will contain the generated CSR.

If a private key has not already been uploaded, then this will generate a new one internally. Note that there is no way to extract this generated private key. If knowing the private key is important, then you will need to generate one locally, upload that, and then generate a new CSR.

Uploading a private key

Send a PATCH request to the gateway token endpoint with `private_key` set to the private key. The private key must be in PEM format with newlines between each line. A status code of 202 means that the change was successful.

Uploading the certificate

Send a PATCH request to the gateway token endpoint with `certificate` set to the certificate. A status code of 202 means that the change was successful.

If you have both the private key and certificate, then they can be uploaded at the same time by setting both keys in the request data.

Status API

The current status of the Pexip Infinity platform and any conference instances currently in progress can be viewed using the API.

Note that all date and time fields are in UTC format.

 If you are using the status API heavily, consider using [Event sinks](#) instead, to reduce the load on the Management Node.

Status resources

A summary of the schema for status resources is available from the:

- Management Node web interface via <https://<manageraddress>/admin/platform/schema/>
- REST API via <https://<manageraddress>/api/admin/status/v1/>

The following status resources are available via the REST API:

Component	Path
Conference instances	/api/admin/status/v1/conference/
Conference instances per node	/api/admin/status/v1/conference_shard/
Participants	/api/admin/status/v1/participant/
Participant media statistics	<a href="/api/admin/status/v1/participant/<participant_id>/media_stream/">/api/admin/status/v1/participant/<participant_id>/media_stream/
Registered aliases	/api/admin/status/v1/registration_alias/
Conferencing Nodes	/api/admin/status/v1/worker_vm/
Conferencing Node load statistics	<a href="/api/admin/status/v1/worker_vm/<worker_vm_id>/statistics/">/api/admin/status/v1/worker_vm/<worker_vm_id>/statistics/
System locations	/api/admin/status/v1/system_location/
System location load statistics	<a href="/api/admin/status/v1/system_location/<system_location_id>/statistics/">/api/admin/status/v1/system_location/<system_location_id>/statistics/
Backplanes	/api/admin/status/v1/backplane/
Backplane media statistics	<a href="/api/admin/status/v1/backplane/<id>/media_stream/">/api/admin/status/v1/backplane/<id>/media_stream/
Management Node	/api/admin/status/v1/management_vm/
Alarms	/api/admin/status/v1/alarm/
Licenses	/api/admin/status/v1/licensing/
Conference synchronization	/api/admin/status/v1/conference_sync/
List all cloud overflow Conferencing Nodes	/api/admin/status/v1/cloud_node/
List all locations monitored for dynamic bursting	/api/admin/status/v1/cloud_monitored_location/
List all locations that contain Conferencing Nodes that may be used for dynamic bursting	/api/admin/status/v1/cloud_overflow_location/
Exchange scheduler	/api/admin/status/v1/exchange_scheduler/
One-Touch Join endpoints	/api/admin/status/v1/mjx_endpoint/
One-Touch Join meetings	/api/admin/status/v1/mjx_meeting/

Component	Path
Teams Connector node status	/api/admin/status/v1/teamsnode/
Teams Connector node call status	/api/admin/status/v1/teamsnode_call/

Specifying the object ID in the path

To retrieve the status of a specific resource, append the object ID of the resource to the path.

For example, a path of `api/admin/status/v1/conference/68aef1a9-7b1b-4442-848d-cbad4b48b320/` retrieves the status of the conference with a conference ID of 68aef1a9-7b1b-4442-848d-cbad4b48b320.

You must also use the relevant object ID in the path of component requests when retrieving resources such as participant media statistics, Conferencing Node load statistics or backplane media statistics.

Resource details

More information can be obtained for each resource by downloading the resource schema in a browser. You may want to install a JSON viewer extension to your browser in order to view the JSON strings in a readable format.

For example, to view the schema for **conference instances** the URI would be:

```
https://<manageraddress>/api/admin/status/v1/conference/schema/?format=json
```

Each schema contains information on the available fields including:

- whether the field is optional (`nullable: true`) or required (`nullable: false`)
- the default value
- the type of data the field must contain
- the choices for the field, e.g. `valid_choices: ["audio", "video", "video-only"]`
- which criteria can be used for [filtering](#) and [ordering](#) searches
- help text with additional information on usage.

Resource methods

Each status resource supports the following HTTP methods:

Method	Action
GET	Retrieves the current status for a resource.

Pagination and filtering

Status requests can be parameterized with pagination and filter fields. For more information, see [Retrieving, paginating, filtering and ordering resource details](#).

Examples

Getting all active conference instances

Retrieving all the conference instances is achieved by submitting a GET request to the resource URI for conference status:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/status/v1/conference/",
    auth=('<user1>', '<password1>'),
    verify=True
)
print("Active conferences:", response.json()['objects'])
```

Example output

```
Active conferences: [
  {
    'start_time': '2015-04-02T09:46:06.106482',
    'resource_uri': '/api/admin/status/v1/conference/00000000-0000-0000-0000-000000000001/',
    'id': '00000000-0000-0000-0000-000000000001',
    'name': 'VMR_1',
    'service_type': 'conference',
    'is_locked': False,
    'is_started': True,
    'guests_muted': False,
    'tag': ''
  }
]
```

Getting all active Virtual Meeting Room conferences

Retrieving only those conference instances that are being held in a Virtual Meeting Room is achieved by submitting a GET request to the resource URI for conference status as above, but filtering it by `service_type`:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/status/v1/conference/?service_type=conference",
    auth=('<user1>', '<password1>'),
    verify=True
)
print("Active conferences:", response.json()['objects'])
```

Getting all participants for a conference

Retrieving all the active participants for a conference instance is achieved by submitting a GET request to the resource URI for participant status and supplying a query parameter to specify the VMR.

The following example finds all participants for the Virtual Meeting Room **VMR_1**:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/status/v1/participant/?conference=VMR_1",
    auth=('<user1>', '<password1>'),
    verify=True
)
print("Active participants for VMR_1:", response.json()['objects'])
```

Example output

```
Active participants for VMR_1: [
  {
    'bandwidth': 576,
    'call_direction': 'in',
    'call_quality': '1_good',
    'call_tag': 'def456',
    'call_uuid': '1c26be9c-6511-4e5c-9588-8351f8c3decd',
    'conference': 'VMR_1',
    'connect_time': '2015-04-02T09:46:11.116767',
    'conversation_id': '1c26be9c-6511-4e5c-9588-8351f8c3decd',
    'destination_alias': 'meet@example.com',
    'display_name': 'Alice',
    'encryption': 'On',
    'has_media': False,
    'id': '00000000-0000-0000-0000-000000000002',
    'idp_uuid': '',
    'is_disconnect_supported': true,
    'is_mute_supported': true,
    'is_idp_authenticated': false,
    'is_muted': False,
    'is_on_hold': False,
    'is_presentation_supported': True,
    'is_presenting': False,
    'is_streaming': False,
    'is_transfer_supported': true,
    'license_count': 0,
    'license_type': 'nolicense',
    'media_node': '10.0.0.1',
    'parent_id': '',
    'participant_alias': 'Infinity_Connect_10.0.0.3',
    'protocol': 'WebRTC',
    'proxy_node': '10.10.0.46',
    'remote_address': '10.0.0.3',
    'remote_port': 54686,
    'resource_uri': '/api/admin/status/v1/participant/00000000-0000-0000-0000-000000000002/',
    'role': 'chair',
    'service_tag': '',
    'service_type': 'conference',
    'signalling_node': '10.0.0.1',
    'source_alias': 'Infinity_Connect_10.0.0.3',
    'system_location': 'London',
    'vendor': 'Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.101 Safari/537.36'
  },
  {
    'bandwidth': 768,
    'call_direction': 'in',
    'call_quality': '1_good',
    'call_tag': 'wxyz789',
    'call_uuid': 'b0a5b554-d1de-11e3-a321-000c29e37602',
    'conference': 'VMR_1',
    'connect_time': '2015-04-02T09:46:53.712941',
    'conversation_id': 'b0a5b554-d1de-11e3-a321-000c29e37602',
    'destination_alias': 'meet@example.com',
```

```

'display_name': 'Bob',
'encryption': 'On',
'has_media': False,
'id': '00000000-0000-0000-0000-000000000003',
'is_disconnect_supported': true,
'is_mute_supported': true,
'is_muted': False,
'is_on_hold': False,
'is_presentation_supported': False,
'is_presenting': False,
'is_streaming': False,
'is_transfer_supported': true,
'license_count': 1,
'license_type': 'port',
'media_node': '10.0.0.1',
'parent_id': '',
'participant_alias': 'bob@example.com',
'protocol': 'H323',
'proxy_node': '',
'remote_address': '10.0.0.2',
'remote_port': 11007,
'resource_uri': '/api/admin/status/v1/participant/00000000-0000-0000-0000-000000000003/',
'role': 'chair',
'service_tag': '',
'service_type': 'conference',
'signalling_node': '10.0.0.1',
'source_alias': 'bob@example.com',
'system_location': 'London',
'vendor': 'TANDBERG (Tandberg 257)'
}
]

```

Note that:

- Perceived call quality:
 - `call_quality` is the most frequently occurring call quality calculation in the last 3 windows, and is determined as described below.
 - `historic_call_quality` shows the sequence of call quality calculations over time. For completed calls, the system looks at packet loss in both directions (Tx and Rx) over multiple 20 second time windows throughout the call and calculates the call quality per window. Packet loss is used to calculate call quality for a time window: < 1% packet loss is perceived as Good quality; < 3% is OK; < 10% is Bad; otherwise it is Terrible. These readings are reported as 0 = Unknown, 1 = Good, 2 = OK, 3 = Bad, 4 = Terrible.
 - `bucketed_call_quality` is a summary of the call quality calculations. For example, reading `[0, 7, 3, 1, 2]` from left to right means there were 0 x Unknown, 7 x Good, 3 x OK, 1 x Bad and 2 x Terrible quality calculations.

Note that `historic_call_quality` and `bucketed_call_quality` are not reported in the response to management API requests for all participants for a conference instance (current and historic), but this data is reported when requesting a specific participant.
- `conversation_id` is the same as `call_uuid` except for Skype for Business / Lync calls.
- `parent_id` is always "".

Getting the media statistics for a participant

Retrieving all the media stream statistics for a participant is achieved by submitting a GET request to the resource URI for the participant status.

The following example finds all media streams for the participant with ID 00000000-0000-0000-0000-000000000002.

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/status/v1/participant/00000000-0000-0000-0000-000000000002/media_stream/",
    auth=('<user1>', '<password1>'),
    verify=True
)
print("Media streams for participant:", response.json()['objects'])
```

Example output

```
Media streams for participant: [
  {
    'end_time': '',
    'id': '1',
    'node': '10.0.0.1'
    'rx_bitrate': 513,
    'rx_codec': 'VP8',
    'rx_fps': 31.0,
    'rx_jitter': 0.29,
    'rx_packet_loss': 0,
    'rx_packets_lost': 0,
    'rx_packets_received': 28761,
    'rx_resolution': '640x480',
    'rx_windowed_packet_loss': 0.0,
    'start_time': '2015-07-22T13:13:52.921269',
    'tx_bitrate': 503,
    'tx_codec': 'VP8',
    'tx_fps': 30.0,
    'tx_jitter': 7.68,
    'tx_packet_loss': 0,
    'tx_packets_lost': 0,
    'tx_packets_sent': 26041,
    'tx_resolution': '768x448',
    'tx_windowed_packet_loss': 0.0,
    'type': 'video'
  },
  {
    'end_time': '',
    'id': '0',
    'node': '10.0.0.1'
    'rx_bitrate': 12,
    'rx_codec': 'opus',
    'rx_fps': 0.0,
    'rx_jitter': 0.56,
    'rx_packet_loss': 0,
    'rx_packets_lost': 0,
    'rx_packets_received': 21148,
    'rx_resolution': '',
    'rx_windowed_packet_loss': 0.0,
    'start_time': '2015-07-22T13:13:52.873617',
    'tx_bitrate': 2,
    'tx_codec': 'opus',
    'tx_fps': 0.0,
    'tx_jitter': 0.21,
    'tx_packet_loss': 0,
    'tx_packets_lost': 0,
    'tx_packets_sent': 42386,
    'tx_resolution': '',
    'tx_windowed_packet_loss': 0.0,
    'type': 'audio'
  }
]
```

Getting the status of a Conferencing Node

By submitting a GET request to the status resource URI of a Conferencing Node you can get its current status. This status will show the last time the Conferencing Node was configured and contacted.

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/status/v1/worker_vm/1/",
    auth=('<user1>', '<password1>'),
    verify=True
)
print("Conferencing node status:", response.json())
```

Getting the load for a system location

By submitting a GET request to the statistics resource URI of the system location status you can get an estimate of the current media load.

The following example gets the media load for the system location with ID 1:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/status/v1/system_location/1/statistics/",
    auth=('<user1>', '<password1>'),
    verify=True
)
print("System Location statistics:", response.json())
```

Getting all registered aliases

Retrieving all the currently registered aliases is achieved by submitting a GET request to the resource URI for registration alias status:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/status/v1/registration_alias/",
    auth=('<user1>', '<password1>'),
    verify=True
)
print("Registered aliases:", response.json()['objects'])
```

Listing all cloud overflow Conferencing Nodes

To retrieve a list of all cloud overflow Conferencing Nodes, submit a GET request to the resource URI for cloud node status:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/status/v1/cloud_node/",
    auth=('<user1>', '<password1>'),
    verify=True
)
print("Cloud nodes:", response.json()['objects'])
```

Example output

```
Cloud nodes: {
  "meta": {
    "limit": 20,
    "next": null,
    "offset": 0,
    "previous": null,
    "total_count": 2
  },
  "objects": [
    {
      "aws_instance_id": "i-edblae65",
      "aws_instance_ip": "172.30.3.6",
      "aws_instance_launch_time": "2016-06-27T23:44:41",
```



```

    "aws_instance_name": "aws_overflow1",
    "aws_instance_state": "stopped",
    "cloud_instance_id": "i-edblae65",
    "cloud_instance_ip": "172.30.3.6",
    "cloud_instance_launch_time": "2016-06-27T23:44:41",
    "cloud_instance_name": "aws_overflow1",
    "cloud_instance_state": "stopped",
    "max_hd_calls": 0,
    "media_load": 100,
    "resource_uri": "/api/admin/status/v1/cloud_node/i-edblae65/",
    "workerm_configuration_id": 6,
    "workerm_configuration_location_name": "AWS",
    "workerm_configuration_name": "aws_overflow1"
  },
  {
    "aws_instance_id": "i-cfb0af47",
    "aws_instance_ip": "172.30.11.83",
    "aws_instance_launch_time": "2016-06-27T23:46:43",
    "aws_instance_name": "aws_overflow2",
    "aws_instance_state": "stopped",
    "cloud_instance_id": "i-cfb0af47",
    "cloud_instance_ip": "172.30.11.83",
    "cloud_instance_launch_time": "2016-06-27T23:46:43",
    "cloud_instance_name": "aws_overflow2",
    "cloud_instance_state": "stopped",
    "max_hd_calls": 0,
    "media_load": 100,
    "resource_uri": "/api/admin/status/v1/cloud_node/i-cfb0af47/",
    "workerm_configuration_id": 7,
    "workerm_configuration_location_name": "AWS",
    "workerm_configuration_name": "aws_overflow2"
  }
]
}

```

Note that:

- When a Conferencing Node is not available for use (in this example the instances are "stopped"), Pexip Infinity reports a media load of 100% to indicate that there is no current capacity available.
- The `aws_instance` prefixed fields are deprecated. Please use the `cloud_instance` prefixed fields instead.

Listing all locations monitored for dynamic bursting

To list all of the system locations that are being monitored for dynamic bursting, submit a GET request to the resource URI for cloud monitored location status:

```

import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/status/v1/cloud_monitored_location/",
    auth=(<user1>, <password1>),
    verify=True
)
print("Monitored principal locations:", response.json()['objects'])

```

Example output

```

Monitored principal locations: {
  "meta": {
    "limit": 20,
    "next": null,
    "offset": 0,
    "previous": null,
    "total_count": 2
  },
  "objects": [
    {
      "free_hd_calls": 31,
      "id": 4,
      "max_hd_calls": 33,
      "media_load": 5,

```

```

    "name": "London",
    "resource_uri": "/api/admin/status/v1/cloud_monitored_location/4/"
  },
  {
    "free_hd_calls": 0,
    "id": 2,
    "max_hd_calls": 42,
    "media_load": 100,
    "name": "Oslo",
    "resource_uri": "/api/admin/status/v1/cloud_monitored_location/2/"
  }
]
}

```

Listing all locations containing dynamic bursting Conferencing Nodes

To list all of the system locations that contain Conferencing Nodes that may be used for dynamic bursting, submit a GET request to the resource URI for cloud monitored location status:

```

import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/status/v1/cloud_overflow_location/",
    auth=('<user1>', '<password1>'),
    verify=True
)
print("Overflow locations:", response.json()['objects'])

```

Example output

```

Overflow locations: {
  "meta": {
    "limit": 20,
    "next": null,
    "offset": 0,
    "previous": null,
    "total_count": 2
  },
  "objects": [
    {
      "free_hd_calls": 31,
      "id": 4,
      "max_hd_calls": 33,
      "media_load": 5,
      "name": "London",
      "resource_uri": "/api/admin/status/v1/cloud_overflow_location/4/"
      "systemlocation_id": 3"
    },
    {
      "free_hd_calls": 0,
      "id": 2,
      "max_hd_calls": 42,
      "media_load": 100,
      "name": "Oslo",
      "resource_uri": "/api/admin/status/v1/cloud_overflow_location/2/"
      "systemlocation_id": 1"
    }
  ]
}

```

History API

The API can be used to view historical information about Pexip Infinity conference instances that are no longer in progress. For example, this can be used to obtain Call Detail Records (CDRs) of the Pexip Infinity platform.

Note that all date and time fields are in UTC format.

Up to 10,000 conference instances are retained, along with all the participant instances associated with each of those conferences. Above 10,000 conference instances, each time a new entry is made the oldest entry is deleted (along with all the participant instances associated with it).

History resources

A summary of the schema for history resources is available from the:

- Management Node web interface via <https://<manageraddress>/admin/platform/schema/>
- REST API via <https://<manageraddress>/api/admin/history/v1/>

The following history resources are available via the REST API:

Component	Path
Alarm history	/api/admin/history/v1/alarm/
Backplane	/api/admin/history/v1/backplane/
Backplane media statistics	/api/admin/history/v1/backplane/<backplane_id>/media_stream/
Conference instance	/api/admin/history/v1/conference/
Participant media statistics	/api/admin/history/v1/participant/<participant_id>/media_stream/
Participant	/api/admin/history/v1/participant/
Conferencing Node events	/api/admin/history/v1/workermv_status_event/

Resource details

More information can be obtained for each resource by downloading the resource schema in a browser. You may want to install a JSON viewer extension to your browser in order to view the JSON strings in a readable format.

For example, to view the schema for **backplane instances** the URI would be:

```
https://<manageraddress>/api/admin/history/v1/backplane/schema/?format=json
```

Each schema contains information on the available fields including:

- whether the field is optional (`nullable: true`) or required (`nullable: false`)
- the default value
- the type of data the field must contain
- the choices for the field, e.g. `valid_choices: ["audio", "video", "video-only"]`
- which criteria can be used for [filtering](#) and [ordering](#) searches
- help text with additional information on usage.

Resource methods

Each history resource supports the following HTTP methods:

Method	Action
GET	Retrieves the history for a resource.

Pagination and filtering

History requests can be parameterized with pagination and filter fields. For more information, see [Retrieving, paginating, filtering and ordering resource details](#).

Examples

Getting all conference instances

Retrieving all the conference history is achieved by submitting a GET request to the resource URI for conference history:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/history/v1/conference/",
    auth=(<user1>, <password1>),
    verify=True
)
print("Conference history:", response.json()['objects'])
```

Note that up to 10,000 conference instances may be stored in the history and by default the response is paginated.

Example output

```
Conference history: [
{
  'duration': 579,
  'end_time': '2015-04-10T09:49:26.556929',
  'id': '8583f400-7886-48c9-874b-5fefc2ac097e',
  'instant_message_count': 0,
  'name': 'meet.alice',
  'participant_count': 3,
  'participants': [
    '/api/admin/history/v1/participant/e9883f1d-88ca-495d-8366-b6eb772dfe57/',
    '/api/admin/history/v1/participant/5881adda-00ef-4315-8886-5d873d2ef269/',
    '/api/admin/history/v1/participant/29744376-0436-4fe1-ab80-06d93c71eb1c/'
  ],
  'resource_uri': '/api/admin/history/v1/conference/8583f400-7886-48c9-874b-5fefc2ac097e/',
  'service_type': 'conference',
  'start_time': '2015-04-10T09:39:47',
  'tag': ''
},
]
```

Getting all participants for a conference instance

Retrieving all the participants for a historical conference instance is achieved by submitting a GET request to the resource URI for participant history and supplying a query parameter to specify the conference.

The following example finds all participants for the conference with the ID 00000000-0000-0000-0000-000000000001:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/history/v1/participant/?conference=00000000-0000-0000-0000-000000000001",
    auth=(<user1>, <password1>),
    verify=True
)
print("Participants for conference:", response.json()['objects'])
```

Example output

```
Participants for conference: [
{
  'bandwidth': 768,
  'call_direction': 'in',
  'call_quality': 'l_good',
  'call_tag': 'wxyz789',
  'call_uuid': 'b0a5b554-d1de-11e3-a321-000c29e37602',
}
```

```
'conference': '/api/admin/history/v1/conference/00000000-0000-0000-0000-000000000001/',
'conference_name': 'VMR_1',
'conversation_id': 'b0a5b554-d1de-11e3-a321-000c29e37602',
'disconnect_reason': 'Call disconnected',
'display_name': 'Bob',
'duration': 519,
'encryption': 'On',
'end_time': '2015-04-02T09:55:33.141261',
'has_media': True,
'id': '00000000-0000-0000-0000-000000000003',
'idp_uuid': '',
'is_idp_authenticated': false,
'is_streaming': false,
'license_count': 1,
'license_type': 'port',
'local_alias': 'meet@example.com',
'media_node': '10.0.0.1',
'media_streams': [
  {
    'end_time': '2015-07-22T12:43:33.645043',
    'id': 36,
    'node': '10.0.0.1',
    'participant': '/api/admin/history/v1/participant/5881adda-00ef-4315-8886-5d873d2ef269/',
    'rx_bitrate': 29,
    'rx_codec': 'opus',
    'rx_packet_loss': 0.0,
    'rx_packets_lost': 0,
    'rx_packets_received': 28091,
    'rx_resolution': '',
    'start_time': '2015-07-22T12:33:31.909536',
    'stream_id': '0',
    'stream_type': 'audio',
    'tx_bitrate': 2,
    'tx_codec': 'opus',
    'tx_packet_loss': 0.0,
    'tx_packets_lost': 0,
    'tx_packets_sent': 56347,
    'tx_resolution': ''
  },
  {
    'end_time': '2015-07-22T12:43:33.683385',
    'id': 37,
    'node': '10.0.0.1',
    'participant': '/api/admin/history/v1/participant/5881adda-00ef-4315-8886-5d873d2ef269/',
    'rx_bitrate': 511,
    'rx_codec': 'VP8',
    'rx_packet_loss': 0.01,
    'rx_packets_lost': 2,
    'rx_packets_received': 37027,
    'rx_resolution': '1280x720',
    'start_time': '2015-07-22T12:33:32.151438',
    'stream_id': '1',
    'stream_type': 'video',
    'tx_bitrate': 511,
    'tx_codec': 'VP8',
    'tx_packet_loss': 0.0,
    'tx_packets_lost': 0,
    'tx_packets_sent': 37335,
    'tx_resolution': '768x448'
  }
],
'parent_id': '29744376-0436-4fe1-ab80-06d93c71eb1c',
'protocol': 'WebRTC',
'proxy_node': 'None',
'remote_address': '10.0.0.2',
'remote_alias': 'Infinity_Connect_Media_10.0.0.2',
'remote_port': 11007,
'resource_uri': '/api/admin/history/v1/participant/00000000-0000-0000-0000-000000000003/',
'role': 'chair',
'service_tag': '',
'service_type': 'conference',
'signalling_node': '10.0.0.1',
'start_time': '2015-04-02T09:46:53.712941',
```

```

    'system_location': 'London',
    'vendor': 'Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.101
Safari/537.36'
  },
  {
    'bandwidth': 0,
    'call_direction': 'in',
    'call_quality': '0_unknown',
    'call_tag': 'def456',
    'call_uuid': '1c26be9c-6511-4e5c-9588-8351f8c3decd',
    'conference': '/api/admin/history/v1/conference/00000000-0000-0000-0000-000000000001/',
    'conference_name': 'VMR_1',
    'conversation_id': '1c26be9c-6511-4e5c-9588-8351f8c3decd',
    'disconnect_reason': 'Call disconnected',
    'display_name': 'Alice',
    'duration': 578,
    'encryption': 'On',
    'end_time': '2015-04-02T09:55:49.348317',
    'has_media': False,
    'id': '00000000-0000-0000-0000-000000000002',
    'is_streaming': false,
    'license_count': 0,
    'license_type': 'nolicense',
    'local_alias': 'meet@example.com',
    'media_node': '10.0.0.1',
    'media_streams': [],
    'parent_id': '',
    'protocol': 'WebRTC',
    'proxy_node': 'None',
    'remote_address': '10.0.0.2',
    'remote_alias': 'Infinity_Connect_10.0.0.2',
    'remote_port': 54686,
    'resource_uri': '/api/admin/history/v1/participant/00000000-0000-0000-0000-000000000002/',
    'role': 'chair',
    'service_tag': '',
    'service_type': 'conference',
    'signalling_node': '10.0.0.1',
    'start_time': '2015-04-02T09:46:11.116767',
    'system_location': 'London',
    'vendor': 'Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.101
Safari/537.36'
  }
]

```

Note that:

- Perceived call quality:
 - `call_quality` is the most frequently occurring call quality calculation, and is determined as described below.
 - `historic_call_quality` shows the sequence of call quality calculations over time. For completed calls, the system looks at packet loss in both directions (Tx and Rx) over multiple 20 second time windows throughout the call and calculates the call quality per window. Packet loss is used to calculate call quality for a time window: < 1% packet loss is perceived as Good quality; < 3% is OK; < 10% is Bad; otherwise it is Terrible. These readings are reported as 0 = Unknown, 1 = Good, 2 = OK, 3 = Bad, 4 = Terrible.
 - `bucketed_call_quality` is a summary of the call quality calculations. For example, reading `[0, 7, 3, 1, 2]` from left to right means there were 0 x Unknown, 7 x Good, 3 x OK, 1 x Bad and 2 x Terrible quality calculations.

Note that `historic_call_quality` and `bucketed_call_quality` are not reported in the response to management API requests for all participants for a conference instance (current and historic), but this data is reported when requesting a specific participant.

- `conversation_id` is the same as `call_uuid` except for Skype for Business / Lync calls.
- `parent_id` is always "".

Getting all participants for a time period

The following example uses filters to find all the participants whose call ended on 8 April 2019:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/history/v1/participant/?end_time__gte=2019-04-08T00:00:00&end_time__lt=2019-04-09T00:00:00",
    auth=('<user1>', '<password1>'),
    verify=True
)
print("Participants for 8th April :", response.json()['objects'])
```

Getting all participants with packet loss

The following example uses a filter to find all the participants with a media stream that had transmit packet loss greater than 0.5%:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/history/v1/participant/?media_streams__tx_packet_loss__gte=0.5",
    auth=('<user1>', '<password1>'),
    verify=True
)
print("Participants with high transmit packet loss :", response.json()['objects'])
```

The following example uses a filter to find all the participants with a media stream that had receive packet loss greater than 0.5%:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/history/v1/participant/?media_streams__rx_packet_loss__gte=0.5",
    auth=('<user1>', '<password1>'),
    verify=True
)
print("Participants with high receive packet loss :", response.json()['objects'])
```

Command API

The command API allows conference control and platform-level operations to be invoked.

Command resources

A summary of the schema for command resources is available from the:

- Management Node web interface via <https://<manageraddress>/admin/platform/schema/>
- REST API via:
 - <https://<manageraddress>/api/admin/command/v1/participant/>
 - <https://<manageraddress>/api/admin/command/v1/conference/>
 - <https://<manageraddress>/api/admin/command/v1/platform/>

The following command resources are available via the REST API:

Component	Path
Dial	/api/admin/command/v1/participant/dial/
Disconnect participant	/api/admin/command/v1/participant/disconnect/
Disconnect conference	/api/admin/command/v1/conference/disconnect/
Mute participant	/api/admin/command/v1/participant/mute/
Mute all Guests	/api/admin/command/v1/conference/mute_guests/
Unmute participant	/api/admin/command/v1/participant/unmute/
Unmute all Guests	/api/admin/command/v1/conference/unmute_guests/
Lock conference	/api/admin/command/v1/conference/lock/
Unlock conference	/api/admin/command/v1/conference/unlock/
Unlock participant	/api/admin/command/v1/participant/unlock/
Transfer participant	/api/admin/command/v1/participant/transfer/
Change participant role	/api/admin/command/v1/participant/role/
Transform layout	/api/admin/command/v1/conference/transform_layout/
Create backup	/api/admin/command/v1/platform/backup_create/
Restore backup	/api/admin/command/v1/platform/backup_restore/
Sync LDAP template	/api/admin/command/v1/conference/sync/
Send provisioning email to VMR owner	/api/admin/command/v1/conference/send_conference_email/
Send provisioning email to device owner	/api/admin/command/v1/conference/send_device_email/
Certificate upload	/api/admin/command/v1/platform/certificates_import/
Start an overflow Conferencing Node	/api/admin/command/v1/platform/start_cloudnode/
Take system snapshot	/api/admin/command/v1/platform/snapshot/
Platform upgrade	/api/admin/command/v1/platform/upgrade/
Upload software bundle	/api/admin/command/v1/platform/software_bundle/

Resource details

More information can be obtained for each resource by downloading the resource schema in a browser. You may want to install a JSON viewer extension to your browser in order to view the JSON strings in a readable format.

For example, to view the schema for the **dial** command the URI would be:

`https://<manageraddress>/api/admin/command/v1/participant/dial/schema/?format=json`

Each schema contains information on the available fields including:

- whether the field is optional (`nullable: true`) or required (`nullable: false`)
- the default value
- the type of data the field must contain
- the choices for the field, e.g. `valid_choices: ["audio", "video", "video-only"]`
- help text with additional information on usage.

Resource methods

Each command resource supports the following HTTP methods:

Method	Action
POST	Invokes a new command for the resource.

Response format

The response for a command will be a JSON object with the following attributes:

Attribute	Value
status	<code>success</code> or <code>error</code> depending on whether or not the command succeeded.
message	An informational message string if the result was <code>error</code> .
data	Command-specific response data.

Individual commands may have response attributes specific to the command. See the command examples for more information.

Examples

Dialing a participant into a conference

By submitting a POST request to the `dial` resource URI, a new participant can be dialed into a conference.

When using this command, note that:

- The `conference_alias` is used for two purposes:
 - The participant being dialed will see the incoming call as coming from this alias.
 - On answer, the participant will join the conference instance associated with this alias.
- If `routing` is set to `routing_rule` then:
 - The `destination` alias must match an outgoing Call Routing Rule for the call to be placed (and it then uses the protocol, outgoing location and call control systems etc. as configured for that rule).
 - You must specify `system_location`. In this case, this acts as the notional source location used when considering if a Call Routing Rule applies or not. A random node in that location is used, which may just take the signaling for the call if the node is loaded.
 - You do not need to specify `node` or `protocol`.
- If `routing` is set to `manual` then:
 - One or both of the `node` and `system_location` must be specified.
 - The `node` overrides the `system_location` if both are given.
 - If only the `system_location` is given, a random node in that location is used, which may just take the signaling for the call if the node is loaded.
 - The `node` must be an IP address and not an FQDN.

The following example places a call to the alias **alice** and uses Call Routing Rules. The node/location used to place the call is determined by whichever Call Routing Rule matches this dial request. The rule can optionally use the **Calls being handled in location** condition to test against the nominated source location of **London**. Alice will see the call as coming from alias **meet@example.com**, and on answer will join the Virtual Meeting Room associated with that alias (in this case **VMR_1**, based on our configuration API examples), as a Host.

```
import requests
import json
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/participant/dial/",
    auth=(<user1>, <password1>),
    data={
        'conference_alias': 'meet@example.com',
        'destination': 'alice',
        'routing': 'routing_rule',
        'remote_display_name': 'Alice Parkes',
        'role': 'chair',
        'system_location': 'London',
    },
    verify=True)
print("New participant created:", json.loads(response.content)['data']['participant_id'])
```

Note that if the dial command is successful the response will contain a `participant_id` attribute which can be used in queries for status and other conference control commands such as [Disconnecting a participant](#) and [Muting a participant](#).

Disconnecting a participant

By submitting a POST request to the `disconnect` resource URI, an existing participant can be disconnected from a conference instance.

```
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/participant/disconnect/",
    auth=(<user1>, <password1>),
    data={
        'participant_id': '00000000-0000-0000-0000-000000000001',
    },
    verify=True)
```

Muting a participant

By submitting a POST request to the `mute` resource URI, the audio being sent from an existing conference participant can be muted, meaning all other participants will not hear them.

```
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/participant/mute/",
    auth=(<user1>, <password1>),
    data={
        'participant_id': '00000000-0000-0000-0000-000000000001',
    },
    verify=True)
```

Muting all Guest participants

By submitting a POST request to the `mute_guests` resource URI, the audio being received from all Guest participants within an existing conference will be muted.

```
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/conference/mute_guests/",
    auth=(<user1>, <password1>),
    data={
        'conference_id': '00000000-0000-0000-0000-000000000001',
    },
    verify=True)
```

Unmuting a participant

By submitting a POST request to the `unmute` resource URI, a previously muted conference participant will have their audio restored, meaning other participants will again be able to hear them.

```
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/participant/unmute/",
    auth=(<user1>, <password1>),
    data={
        'participant_id': '00000000-0000-0000-0000-000000000001',
    },
    verify=True)
```

Unmuting all Guest participants

By submitting a POST request to the `unmute_guests` resource URI, the audio being received from all Guest participants within an existing conference will be unmuted.

```
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/conference/unmute_guests/",
    auth=(<user1>, <password1>),
    data={
        'conference_id': '00000000-0000-0000-0000-000000000001',
    },
    verify=True)
```

Locking a conference instance

By submitting a POST request to the conference `lock` resource URI, the conference instance will be locked, preventing new participants from joining. Instead new participants will be held at the **Waiting for conference host** screen. Note that if a service has a Host PIN, participants who enter the PIN will still be able to access the conference even when it is locked.

```
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/conference/lock/",
    auth=('<user1>', '<password1>'),
    data={
        'conference_id': '00000000-0000-0000-0000-000000000001',
    },
    verify=True)
```

Unlocking a conference instance

By submitting a POST request to the conference `unlock` resource URI, a previously locked conference instance can be unlocked, meaning new participants will be allowed to join. Any participants held at the **Waiting for conference host** screen will also automatically join the conference.

```
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/conference/unlock/",
    auth=('<user1>', '<password1>'),
    data={
        'conference_id': '00000000-0000-0000-0000-000000000001',
    },
    verify=True)
```

Unlocking a participant

By submitting a POST request to the participant `unlock` resource URI, a participant held at the **Waiting for conference host** screen because the conference has been locked will be allowed to join the conference.

```
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/participant/unlock/",
    auth=('<user1>', '<password1>'),
    data={
        'participant_id': '00000000-0000-0000-0000-000000000001',
    },
    verify=True)
```

Transferring a participant

By submitting a POST request to the participant `transfer` resource URI, a participant can be moved from one conference to another. The target conference is identified by an alias in the `conference_alias` field, and they will have the specified `role`.

If the target conference is PIN-protected, the participant will bypass the PIN entry.

```
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/participant/transfer/",
    auth=('<user1>', '<password1>'),
    data={
        'participant_id': '00000000-0000-0000-0000-000000000001',
        'conference_alias': 'meet@example.com',
        'role': 'guest'
    },
    verify=True)
```

Changing a participant's role

By submitting a POST request to the participant `role` resource URI, a participant can have its role changed. The target participant is identified by the `participant_id` field, and it will be given the specified `role` ("guest" or "chair").

```
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/participant/role/",
    auth=('<user1>', '<password1>'),
    data={
        'participant_id': '00000000-0000-0000-0000-000000000001',
        'role': 'chair',
    },
    verify=True)
```

Changing a conference's layout

By submitting a POST request to the conference `transform_layout` resource URI, you can control the layout, overlay text and other in-conference indicators.

```
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/conference/transform_layout/",
    auth=('<user1>', '<password1>'),
    data={
        'conference_id': '00000000-0000-0000-0000-000000000001',
        'enable_overlay_text': True,
        'layout': '4:0'
    },
    verify=True)
```

Creating a system backup

You can create a system backup by submitting a POST request to the platform `backup_create` resource URI.

You must specify a passphrase (replacing `<backup_password>` in the example below). The passphrase is used to encrypt the backup file. You must remember the passphrase as it will be required if you need to subsequently restore the data.

```
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/platform/backup_create/",
    auth=('<user1>', '<password1>'),
    data={
        'passphrase': '<backup_password>',
    },
    verify=True)
```

The backup file is created on the Management Node under the `https://<manageraddress>/api/admin/configuration/v1/system_backup/` location.

Restoring a system backup

To restore a system backup (from a file stored on the Management Node) you need to:

1. List the available backups on the Management Node to identify the filename of the backup you want to restore.
2. Download the backup file to a temporary location.
3. Restore the contents of the backup file to your Pexip Infinity system.

Listing the available backups

You can perform a GET on `/api/admin/configuration/v1/system_backup/` to list the available backup files on the Management Node that can be restored (however, restoration must occur on exactly the same software version of Pexip Infinity that the backup was taken from).

```
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/configuration/v1/system_backup/",
    auth=(<user1>, <password1>),
    verify=True)
print(response.content)
```

The response will include data similar to this for each backup file:

```
{
  "build": "59437.0.0",
  "date": "2021-01-08T15:36:16",
  "filename": "pexip_backup_10-44-7-0-mgr_25_59437.0.0_21_01_08_09_05_43.tar.pexbak",
  "resource_uri": "/api/admin/configuration/v1/system_backup/pexip_backup_10-44-7-0-mgr_25_59437.0.0_21_01_08_09_05_43.tar.pexbak/",
  "size": 214702132,
  "version": "25"
}
```

The "filename" value is what you will use in place of <filename> in the following GET command to download the file.

Note that if a backup file is no longer needed, you can perform a DELETE on the

`https://<manageraddress>/api/admin/configuration/v1/system_backup/<filename>/` URL to delete individual files.

Downloading the backup file

Next you can perform a GET on `/api/admin/configuration/v1/system_backup/<filename>/` to download the backup file to a temporary location.

```
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/configuration/v1/system_backup/<filename>/",
    auth=(<user1>, <password1>),
    verify=True)
with open("/tmp/temp.bak", "w") as file_handle:
    file_handle.write(response.content)
```

This downloads the backup file to `/tmp/temp.bak` — you can change this to your preferred location and filename if required.

Using the example response from the list of available backups, the GET would be for:

`https://10.44.7.0/api/admin/configuration/v1/system_backup/pexip_backup_10-44-7-0-mgr_25_59437.0.0_21_01_08_09_05_43.tar.pexbak/`

Restoring the backup

Finally, you can restore the contents of the backup file by submitting a POST request to the platform `backup_restore` resource URI, telling it to use the temporary file created in the previous step.

You must also specify the passphrase that was used to encrypt the backup file when it was generated.

```
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/platform/backup_restore/",
    auth=(<user1>, <password1>),
    data={
        'passphrase': <backup_password>,
    },
    files={
        'package': open('/tmp/temp.bak'),
    },
    verify=True)
```

Starting an overflow Conferencing Node

If you are using dynamic bursting, the `start_cloudnode` resource can be used to manually start up a specific overflow node.

```
import requests
import json
response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/platform/start_cloudnode/",
    json={'instance_id': '<node instance id>'},
    auth=('<user1>', '<password1>'),
    verify=True)
print(response.content)
```

You can use the [cloud_node status resource](#) to obtain the instance IDs for your overflow nodes.

If the command is successful the response content takes the format: `'{"status": "success"}'`

If the Conferencing Node is already running, the response is: `'{"status": "failed", "error": "Unable to start a cloud node which isn't in 'STOPPED' state"}'`

If the instance_id does not match a cloud overflow node, the response is a 400 Bad Request with the following content: `'{"start_cloudnode": {"instance_id": ["Failed to find the cloud node."}]}'`

Taking a system snapshot

To take a 12 hour system snapshot:

```
import requests

save_path = '/tmp/snapshots/'

response = requests.post(
    "https://<manageraddress>/api/admin/command/v1/platform/snapshot/",
    auth=('<user1>', '<password1>'),
    data={
        'limit': 12 # hours
    },
    verify=True)

get_filename = response.headers['Content-Disposition'].split("=") # snapshot filename is returned in the header
set_filename = get_filename[1]
save_path = save_path + set_filename

if response.status_code == 200:
    snapshot = response.content
    with open(save_path, 'wb') as f:
        f.write(snapshot)
```

Retrieving, paginating, filtering and ordering resource details

This section describes how the management API resources are organized, and how to retrieve the current configuration for a resource via the management API. It covers [Dealing with resources](#), [Pagination](#), [Filtering](#) and [Ordering](#).

Dealing with resources

Every element on the API is described as a resource.

Using VMRs as an example, VMR configuration is represented by the `conference` resource. As you typically want to configure more than one VMR, the API lets you deal with multiple resources via a **list**.

If you do a GET query for `api/admin/configuration/v1/conference/` you get all the currently configured VMRs as a list. If you send a DELETE request to the same URL then you will delete **all** VMR configuration.

However, often you're only interested in dealing with one VMR, in which case you want the **detail** of a specific VMR. You can send a GET request with a URL that targets a specific VMR resource, for example a GET of `api/admin/configuration/v1/conference/1/` will return only the configuration for the first VMR. Similarly if you want to change just that VMR then you would send a PATCH request to the same URL with just the fields you wanted to change.

Getting a single resource object

By concatenating an object ID with the resource URI, details can be obtained for a single resource object.

For example, to GET the **alias** with ID 1 the URI would be:

```
https://<manageraddress>/api/admin/configuration/v1/conference_alias/1/
```

The response to a GET for a single resource is a JSON object with attributes for each field of the resource.

The ID of existing objects can be identified from the URI shown in the Administrator interface when viewing the item, for example when viewing a Virtual Meeting Room the URI is in the format `https://<manageraddress>/admin/conferencing/conference/117/change/` where in this case the ID of this VMR is 117.

When using the management API to retrieve information about existing resources, the `id` field identifies the object ID. In this example of a VMR resource, the ID is 3:

```
objects: [
  - {
    - aliases: [
      - {
        alias: "test_call",
        conference: "/api/admin/configuration/v1/conference/3/",
        creation_time: "2020-03-25T20:17:41.971755",
        description: "Pexam alias test_call",
        id: 3
      }
    ],
    allow_guests: false,
    automatic_participants: [ ],
    call_type: "video",
    creation_time: "2020-03-25T20:17:41.962332",
    crypto_mode: "",
    description: "Pexam generated conference 'Test Call Service'",
    enable_chat: "default",
    enable_overlay_text: false,
    force_presenter_into_main: false,
    gms_access_token: null,
    guest_pin: "",
    guest_view: "one_main_seven_pips",
    guests_can_present: true,
    host_view: "one_main_seven_pips",
    id: 3,
    ivr_theme: null,
    match_string: "",
    max_callrate_in: null,
    max_callrate_out: null,
    max_pixels_per_second: null,
    mssip_proxy: null,
    mute_all_guests: false,
    name: "Test Call Service",
    participant_limit: null,
    pin: "",
    post_match_string: "",
    post_replace_string: "",
    primary_owner_email_address: "",
    replace_string: "",
    resource_uri: "/api/admin/configuration/v1/conference/3/",
    scheduled_conferences: [ ],
    scheduled_conferences_count: 0,
    service_type: "test_call",
    sync_tag: "",
    system_location: null,
    tag: "",
    teams_proxy: null,
    two_stage_dial_type: "regular"
  }
]
```

Getting multiple resource objects

If a GET operation is invoked on the root resource URI then details about multiple objects are returned. The response to a GET for multiple resources is a JSON object with two attributes:

Attribute	Value
meta	This is an object giving meta information about the response.
objects	This is a list of resource objects.

For example, to GET all conference aliases the URI would be:

```
https://<manageraddress>/api/admin/configuration/v1/conference_alias/
```

Global settings

The **global settings** (`/api/admin/configuration/v1/global/`) are a special case.

To make the API as consistent as possible, the global settings are treated in the same way as other resources i.e. there is a list of global settings. However, you are only allowed one global settings resource. Thus, if you get the list of all global settings you will only ever get a list with one entry in it. Also, you are not allowed to delete all global settings.

Therefore you should almost always deal directly with the detail of the existing resource (via `api/admin/configuration/v1/global/1/`) rather than using it a list.

Pagination

By default, the response is paginated and it contains the first page of 20 results. To retrieve subsequent pages of results the **offset** parameter must be specified in the URI.

To carry on from our previous example, to retrieve the second page of results the URI would be:

```
https://<manageraddress>/api/admin/configuration/v1/conference_alias/?offset=20
```

The **limit** parameter can be used to change the number of results in the response.

For example, to return the first 100 objects the URI would be:

```
https://<manageraddress>/api/admin/configuration/v1/conference_alias/?limit=100
```

The **limit** parameter has a maximum value of **10000**.

Note that if there are a large number of aliases configured this may put a significant load on both the Management Node and the client making the request.

Filtering

A request for multiple objects can be filtered so that only those objects that match specific criteria are returned. The table below lists the available match specifiers.

Note that:

- Filtering is case insensitive for the **contains**, **startswith** and **endswith** keywords (but not **exact** and **regex**).
- The **lt**, **lte**, **gt** and **gte** filters use ordinal comparison for strings — thus Ark < Rock < ark < rock.

Specifier	Description
exact	Matches objects whose field exactly matches the value (case sensitive).
iexact	Case-insensitive version of the exact match.
contains	Matches objects whose field contains the value.
startswith	Matches objects whose field starts with the value.
endswith	Matches objects whose field ends with the value.
regex	Matches objects whose field matches the regular expression value (case sensitive).
iregex	Case-insensitive version of the regex match.
lt	Matches objects whose field is less than the value.
lte	Matches objects whose field is less than or equal to the value.
gt	Matches objects whose field is greater than the value.
gte	Matches objects whose field is greater than or equal to the value.

The criteria that can be used for filtering are included at the end of the schema for each resource. If no `filtering` section is present, then filtering is not available.

For example, to see which criteria you can use to filter a search for service aliases, look at the associated schema:

```
https://<manageraddress>/api/admin/configuration/v1/conference_alias/schema/?format=json
```

You will see at the end:

```
filtering: {
  alias: 1,
  conference: 2,
  creation_time": 1,
  description: 1
}
```

This indicates that you can filter by any of the criteria listed above. The `1` and `2` following each criteria indicate that all of the match specifiers listed in the table above are valid when filtering using that criteria. Alternatively, if not all specifiers are valid, those that are valid are listed instead.

For example, to search for information about the alias **meet.alice** the URI would be:

```
https://<manageraddress>/api/admin/configuration/v1/conference_alias/?alias=meet.alice
```

For example, to search for all aliases that start with **meet**, the URI would be:

```
https://<manageraddress>/api/admin/configuration/v1/conference_alias/?alias__startswith=meet.
```

To query in an array, for example `aliases`, you must query on items inside the array i.e. `aliases__alias`. For example, to display all VMRs containing an alias in a URI format:

```
https://<manageraddress>/api/admin/configuration/v1/conference/?aliases__alias__contains=@
```

Ordering

A request for multiple objects can be ordered by field values. For example, to get the conference history ordered by start time the URI would be:

```
https://<manageraddress>/api/admin/history/v1/conference/?order_by=start_time
```

The order can be reversed by adding a hyphen character (-) before the field name. For example, to order by descending start time so that the most recent conferences are listed first, the URI would be:

```
https://<manageraddress>/api/admin/history/v1/conference/?order_by=-start_time
```

The fields that can be used for ordering are included at the end of the schema for each resource. If an ordering section is not present, then ordering is not available.

Using the API with SNMP

If SNMP is enabled on each Conferencing Node and the Management Node, you can use the Management API to obtain information using SNMP.

Note that SNMP is disabled by default, and is enabled and disabled on each node individually.

Examples

The examples below assume a community name of **public** (this is the default value and so is insecure).

Retrieving the SNMP sysName

```
# Retrieve the SNMPv2-MIB 'sysName' using an SNMP GET operation using SNMPV2c
from pysnmp.entity.rfc3413.oneliner import cmdgen
pexip_node_ip_address = '<pexipipaddress>'
cmdGen = cmdgen.CommandGenerator()
error_indication, error_status, error_index, var_binds = cmdGen.getCmd(
    cmdgen.CommunityData('public'),
    cmdgen.UdpTransportTarget((pexip_node_ip_address, 161)),
    cmdgen.MibVariable('SNMPv2-MIB', 'sysName', 0)
)
# Check for errors and print out results
if not error_indication and not error_status:
    for name, val in var_binds:
        print('%s = %s' % (name.prettyPrint(), val.prettyPrint()))
else:
    if error_indication:
        print(error_indication)
    if error_status:
        print('%s at %s' % (error_status.prettyPrint(), error_index and var_binds[int(error_index)-1] or '?'))
```

Retrieving CPU load average

The value returned is on a scale where 1.00 equals one CPU core at full load. So a 4 CPU system would return values between 0.00 (completely idle) and 4.00 (completely maxed out).

```
# Retrieve the 1 minute CPU load average (OID: '.1.3.6.1.4.1.2021.10.1.3.1')
# using an SNMP GET operation using SNMPV2c
from pysnmp.entity.rfc3413.oneliner import cmdgen
pexip_node_ip_address = '<pexipipaddress>'
cmdGen = cmdgen.CommandGenerator()
error_indication, error_status, error_index, var_binds = cmdGen.getCmd(
    cmdgen.CommunityData('public'),
    cmdgen.UdpTransportTarget((pexip_node_ip_address, 161)),
    '.1.3.6.1.4.1.2021.10.1.3.1'
)
# Check for errors and print out results
if not error_indication and not error_status:
    for name, val in var_binds:
        print('%s = %s' % (name.prettyPrint(), val.prettyPrint()))
else:
    if error_indication:
        print(error_indication)
    if error_status:
        print('%s at %s' % (error_status.prettyPrint(), error_index and var_binds[int(error_index)-1] or '?'))
```