Pexip Infinity

Management API

Software Version 11

Document Version 11.a

December 2015

Contents

Introduction 5

About Pexip Infinity 5

About this guide 5

Definitions 5

Intended audience 5

Using the management API 6

Obtaining complete API information (schemas) 6

Authentication 6

Examples used in the guide 6

Configuration replication delay 6

API performance 7

API design guidelines 7

Security 7

Configuration API 8

Configuration resources 8

Resource details 9

Resource methods 9

Getting a single resource object 10

Getting multiple resource objects 10

Pagination 10

Filtering 11

Ordering 12

Creating a single resource object 12

Updating a single resource object 12

Updating multiple resource objects 12

Deleting all resource objects 12

Examples 13

Creating a Virtual Meeting Room, Virtual Auditorium or Virtual Reception 13

Getting a Virtual Meeting Room's configuration 13

Changing an existing Virtual Meeting Room 14

Adding a Virtual Meeting Room alias 14

Deleting a Virtual Meeting Room 14

Creating a Virtual Meeting Room with aliases 15

Getting all Virtual Meeting Rooms, Virtual Auditoriums and Virtual Receptions 15

Getting all Virtual Meeting Rooms only 15

Creating multiple Virtual Meeting Rooms ... 16
Creating a Virtual Auditorium ... 16
Creating a Virtual Reception ... 16
Creating Automatically Dialed Participants ... 17
Modifying an Automatically Dialed Participant ... 17
Removing an Automatically Dialed Participant from a Virtual Meeting Room ... 17
Deleting an Automatically Dialed Participant ... 18
Creating Gateway Routing Rules ... 18
Deleting a Gateway Routing Rule ... 18
Deploying a new Conferencing Node ... 19
Downloading a Conferencing Node for manual deployment ... 19
Deploying a Conferencing Node using a VM template and configuration file ... 20
Uploading a service theme ... 21

Status API **22**

Status resources ... 22
Resource methods ... 22
Pagination and filtering ... 23
Examples ... 23
Getting all active conference instances ... 23
Getting all active Virtual Meeting Room conferences ... 23
Getting all participants for a conference ... 24
Getting the media statistics for a participant ... 26
Getting the status of a Conferencing Node ... 27
Getting the load for a system location ... 27
Getting all registered aliases ... 27

History API **28**

History resources ... 28
Resource methods ... 28
Pagination and filtering ... 28
Examples ... 29
Getting all conference instances ... 29
Getting all participants for a conference instance ... 29
Getting all participants for a time period ... 31
Getting all participants with packet loss ... 31

Command API **32**

Command resources ... 32
Resource details ... 32
Resource methods ... 32

Response format 33
Examples 33
Dialing a participant into a conference 33
Disconnecting a participant 34
Muting a participant 34
Muting all guest participants 34
Unmuting a participant 34
Unmuting all guest participants 35
Locking a conference instance 35
Unlocking a conference instance 35
Unlocking a participant 35
Using the API with SNMP **36**
Examples 36
Retrieving the SNMP sysName 36
Retrieving CPU load average 36
Log output **37**
Administrator log system modules 37
administrator.system 37
administrator.configuration 38
administrator.conference 38
administrator.participantdialer 40
administrator.gatewayroutingrulematcher 41
administrator.registration 41
administrator.vmrsync 41
Support Log system modules 41
support.sip 41
support.dns 41
support.bfcp 41
support.h323.q931, support.h323.ras, support.h323.h245 42
support.rest 42
Content of signaling messages 42
support.participant 42
References **43**
HTTP(S) 43
JSON 43
REST 43
Python 43

Introduction

About Pexip Infinity

Pexip Infinity is a distributed conferencing platform. It provides any number of users with their own personal Virtual Meeting Room**s**, as well as larger Virtual Auditorium**s**, where they can hold conferences and share presentations. Participants can join over voice, video and mobile from virtually any type of endpoint. Virtual Meeting Rooms and Virtual Auditoriums exist simultaneously in multiple locations, allowing participants to share the same high-quality conferencing experience regardless of distance.

The Pexip Infinity platform includes a Virtual Reception IVR service, which allows all participants to dial a single number to access the Pexip Infinity, and then use the DTMF tones on their endpoint to enter the number of the specific Virtual Meeting Room or Virtual Auditorium they wish to join. The platform also includes the Pexip Distributed Gateway, which interworks calls between different protocols, therefore providing you with a truly interoperable conferencing service.

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. It also provides information on the content of the system's logs.

This guide describes how to use the Pexip Infinity version 11 API. It is made up of the following sections:

Using the management API	Provides an overview of the RESTful web API used to communicate with a Pexip Infinity Management Node.
Configuration API	Describes how to use the API to configure the Pexip Infinity platform and services (Virtual Meeting Rooms, Virtual Auditoriums, Virtual Receptions and the Pexip Distributed Gateway).
Status API	Describes how to use the API to obtain status information on the Pexip Infinity platform and services.
History API	Describes how to use the API to obtain historical information on the Pexip Infinity platform and services.
Command API	Describes how to use the API to control aspects of Pexip Infinity conference instances.
Log output	Provides detailed information on the content of the Pexip Infinity support log and administrator log.
References	Provides links to further useful information.

Definitions

In the context of this API guide:

- A **service** is a Virtual Meeting Room, Virtual Auditorium, Virtual Reception or Pexip Distributed 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

It is assumed that readers are familiar with the concepts of HTTP(S), JSON and REST.

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 via **https://<manageraddress>/api/admin/schema/**. If you require the schemas but do not have access to a Management Node running v11, please contact your authorized Pexip 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 2.7 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 will be **admin**.

Configuration replication delay

It may take some time between configuration being provided by the REST API and this configuration taking effect across the Conferencing Nodes. This delay is not usually more than approximately 60s, depending on load and network conditions.

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.

When configuration is modified, the modified configuration is replicated to the various Conferencing Nodes. This means that configuration changes will 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

Some guidelines for designing your application:

- Avoid duplication. Avoid designs which will perform large amounts of polling for information that is not needed or which will result in repeatedly writing the same configuration values.
- Consider application designs which 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.
- 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.
- 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.

Security

We recommend that you upload a HTTPS certificate to your Management Node and each Conferencing Node and that client applications verify those certificates, in order to undertake server validation.

© 2015 Pexip AS

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

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/
Syslog server	/api/admin/configuration/v1/syslog_server/
SNMP NMS	/api/admin/configuration/v1/snmp_network_management_system
VM manager	/api/admin/configuration/v1/host_system/
Static route	/api/admin/configuration/v1/static_route/
Platform configuration	
System location	/api/admin/configuration/v1/system_location/
Management Node	/api/admin/configuration/v1/management_vm/
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/
Global settings	/api/admin/configuration/v1/global/
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/
Service configuration	
Virtual Meeting Room, Virtual Auditorium or Virtual Reception	/api/admin/configuration/v1/conference/
Alias for a Virtual Meeting Room, Virtual Auditorium or Virtual Reception	/api/admin/configuration/v1/conference_alias/
Automatically Dialed Participant	/api/admin/configuration/v1/automatic_participant/

Component	Path
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/
Conference LDAP sync source	/api/admin/configuration/v1/ldap_sync_source/
Conference LDAP sync field *	/api/admin/configuration/v1/ldap_sync_field/
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/
Utilities	
Upgrade	/api/admin/configuration/v1/upgrade/
System backup	/api/admin/configuration/v1/system_backup/

* This resource is new in Pexip Infinity v11.

Resource details

More information can be obtained for each resource by downloading the resource schema in a browser.

You may wish 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 type of data the field must contain
- help text with additional information on usage
- which criteria can be used for filtering searches.

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 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 will be a JSON object with attributes for each field of the resource.

Getting multiple resource objects

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

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

Pagination

By default, the response will be paginated and the response will contain 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`

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 following match specifiers are available:

Specifier	Description
exact	Matches objects whose field exactly matches the value.
iexact	Case-insensitive version of the **exact** match.
contains	Matches objects whose field contains the value.
icontains	Case-insensitive version of the **contains** match.
startswith	Matches objects whose field starts with the value.
istartswith	Case-insensitive version of the **startswith** match.
endswith	Matches objects whose field ends with the value.
iendswith	Case-insensitive version of the **endswith** match.
regex	Matches objects whose field matches the regular expression value.
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 Virtual Meeting Room, 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,
    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 will be 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.
```

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 no ordering section is present, then ordering is not available.

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 will contain a Location header which contains the REST URI of the newly created 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 or update 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 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 or Virtual Reception

To create a new Virtual Meeting Room, Virtual Auditorium or Virtual Reception, 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

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=False,
    data=json.dumps({'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 the new Virtual Meeting Room, you can get the Virtual Meeting Room configuration:

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

Example output

```
Virtual Meeting Room: {
    'aliases': [
        {
        'conference': '/api/admin/configuration/v1/conference/1/',
        'alias': 'meet@example.com',
        'description': '',
        'id': 1
        }
        ]
    'allow_guests': False,
    'automatic_participants': [],
    'description': '',
    'force_presenter_into_main': False,
    'guest_pin': '',
    'guest_view': None,
    'host_view': 'one_main_seven_pips',
    'id': 1,
    'ivr_theme': None,
    'max_callrate_in': None,
    'max_callrate_out': None,
    'name': 'VMR_1',
    'participant_limit': None,
    'pin': '',
    'resource_uri': '/api/admin/configuration/v1/conference/1/',
    'service_type': 'conference',
    'sync_tag': '',
    '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=False,
    data=json.dumps({'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=False,
    data=json.dumps({
        'alias': 'meet@example.com',
        'conference': '/api/admin/configuration/v1/conference/1/',
        })
    )
print "Created new alias:", response.headers['location']
```

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=False
    )
```

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, it is possible to 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=False,
    data=json.dumps({
        'name': 'VMR_1',
        'service_type': 'conference',
        'aliases': [{'alias' : 'meet'}, {'alias' : 'meet@example.com'}]
        })
    )
print "Created new Virtual Meeting Room:", response.headers['location']
```

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=False
    )
print "Virtual Meeting Rooms:", json.loads(response.text)['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=False
    )
print "Virtual Meeting Rooms:", json.loads(response.text)['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=False,
    data=json.dumps(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=False,
    data=json.dumps({
        '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=False,
    data=json.dumps({
        '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=False,
    data=json.dumps({
        'alias' : 'myendpoint@mydomain.com',
        '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=False,
    data=json.dumps({
        '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=False,
    data=json.dumps({
        '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=False
    )
```

Creating Gateway Routing Rules

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

The following example creates a Gateway Routing Rule with the name **Rule1**, which gateways from H.323, SIP and WebRTC to Microsoft Lync:

```
import json
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/gateway_routing_rule/",
    auth=('<user1>', '<password1>'),
    verify=False,
    data=json.dumps({
        'name': 'Rule1',
        'priority': 50,
        'match_incoming_h323': True,
        'match_incoming_mssip': False,
        'match_incoming_sip': True,
        'match_incoming_webrtc': True,
        'match_string' : '9(.*)',
        'replace_string' : '',
        'outgoing_protocol': 'mssip',
        })
    )
print "Created new Gateway Routing Rule:", response.headers['location']
```

Deleting a Gateway Routing Rule

The following example deletes the Gateway Routing Rule with ID **1**:

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

Deploying a new Conferencing Node

By submitting a POST request to the resource URI for Conferencing Nodes, you can create and deploy a new Conferencing Node onto a server running vSphere ESXi 4.1 or higher.

```
import json
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/worker_vm/",
    auth=('<user1>', '<password1>'),
    verify=False,
    data=json.dumps({
        'name': 'new_node',
        'hostname': 'newnode',
        'domain': 'example.test',
        'address': '<newnode_ip_address>',
        'netmask': '<newnode_ip_mask>',
        'gateway': '<ip_gateway_address>',
        'password': '<newnode_password>',
        'system_location': '/api/admin/configuration/v1/system_location/1/',
        'deployment_type': 'VMWARE',
        'host': 1,
        'host_username': '<vcenter_username>',
        'host_password': '<vcenter_password>',
        'host_network': 'VM Network',
        'host_resource_path': '/<vmware_datacenter>/host/<vmware_host>/Resources',
        'vm_cpu_count': '8',
        'vm_system_memory': '16384',
        })
    )
print "Created new Conferencing Node:", response.headers['location']
```

Downloading a Conferencing Node for manual deployment

By submitting a POST request with a `deployment_type` of `MANUAL`, you can download an OVA file which can be used to deploy a new Conferencing Node onto a host server running vSphere ESXi 5.0 or higher.

Deployment types of `MANUAL-ESXI4` and `MANUAL-HYPERV2012` can also be used.

```
import json
import requests
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/worker_vm/",
    auth=('<user1>', '<password1>'),
    verify=False,
    stream=True,
    data=json.dumps({
        'name': 'new_node',
        'hostname': 'newnode',
        'domain': 'example.test',
        'address': '<newnode_ip_address>',
        'netmask': '<newnode_ip_mask>',
        'gateway': '<ip_gateway_address>',
        'password': '<newnode_password>',
        'system_location': '/api/admin/configuration/v1/system_location/1/',
        'deployment_type': 'MANUAL',
        '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=False,
    stream=True,
    data=json.dumps({
        'name': 'new_node',
        'hostname': 'newnode',
        'domain': 'example.test',
        'address': '<newnode_ip_address>',
        'netmask': '<newnode_ip_mask>',
        'gateway': '<ip_gateway_address>',
        'password': '<newnode_password>',
        '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=False,
        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
import urlparse
response = requests.post(
    "https://<manageraddress>/api/admin/configuration/v1/ivr_theme/",
    auth=('<user1>', '<password1>'),
    data=json.dumps({
        'name': 'New theme'
    }),
    verify=False)
theme_uri = response.headers['location']
theme_path = urlparse.urlsplit(theme_uri).path
response = requests.patch(
    theme_uri,
    auth=('<user1>', '<password1>'),
    files={
        'package': open('test_theme.zip', 'rb')
    },
    verify=False)
response = requests.patch(
    "https://<manageraddress>/api/admin/configuration/v1/conference/1/",
    auth=('<user1>', '<password1>'),
    data=json.dumps({
        'ivr_theme': theme_path
    }),
    verify=False)
```

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 time format.

Status resources

The following status resources are available via the REST API:

Component	Path
Conference instance	/api/admin/status/v1/conference/
Conference instance per node	/api/admin/status/v1/conference_shard/
Participant	/api/admin/status/v1/participant/
Registered aliases *	/api/admin/status/v1/registration_alias/
Conferencing Node	/api/admin/status/v1/worker_vm/
Participant media statistics	/api/admin/status/v1/participant/<participant_id>/media_stream/
Conferencing Node load statistics	/api/admin/status/v1/worker_vm/<worker_vm_id>/statistics/
System location	/api/admin/status/v1/system_location/
System location load statistics	/api/admin/status/v1/system_location/<system_location_id>/statistics/
Backplane	/api/admin/status/v1/backplane/
Backplane media statistics	/api/admin/status/v1/backplane/<id>/media_stream/
Alarms	/api/admin/status/v1/alarm/
Licensing	/api/admin/status/v1/licensing/
Conference synchronization	/api/admin/status/v1/conference_sync/

* This resource is new in Pexip Infinity v11.

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 parametrized with pagination and filter fields. For more information, see Pagination and Filtering in the Configuration API section.

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=False
    )
print "Active conferences:", json.loads(response.text)['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,
    '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=False
    )
print "Active conferences:", json.loads(response.text)['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=False
    )
print "Active participants for VMR_1:", json.loads(response.text)['objects']
```

Example output

```
Active participants for VMR_1: [
    {
    'bandwidth': 576,
    'call_direction': 'in',
    'call_uuid': '1c26be9c-6511-4e5c-9588-8351f8c3decd',
    'conference': 'VMR_1',
    'connect_time': '2015-04-02T09:46:11.116767',
    'destination_alias': 'meet@example.com',
    'display_name': 'Alice',
    'encryption': 'On',
    'has_media': False,
    'id': '00000000-0000-0000-0000-000000000002',
    'is_muted': False,
    'is_on_hold': False,
    'is_presentation_supported': True,
    'is_presenting': False,
    'is_streaming': False,
    'license_count': 0,
    'media_node': '10.0.0.1',
    'parent_id': '',
    'participant_alias': 'Infinity_Connect_10.0.0.3',
    'protocol': 'WebRTC',
    '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_uuid': 'b0a5b554-d1de-11e3-a321-000c29e37602',
    'conference': 'VMR_1',
    'connect_time': '2015-04-02T09:46:53.712941',
    'destination_alias': 'meet@example.com',
    'display_name': 'Bob',
    'encryption': 'On',
    'has_media': False,
    'id': '00000000-0000-0000-0000-000000000003',
    'is_muted': False,
    'is_on_hold': False,
    'is_presentation_supported': False,
    'is_presenting': False,
    'is_streaming': False,
    'license_count': 1,
    'media_node': '10.0.0.1',
```

```
'parent_id': '',
'participant_alias': 'bob@example.com',
'protocol': 'H323',
'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)'
}
]
```

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=False
    )
print "Media streams for participant:", json.loads(response.text)['objects']
```

Example output

```
Media streams for participant: [
    {
    'end_time': '',
    'id': '1',
    'node': '10.0.0.1'
    'rx_bitrate': 513,
    'rx_codec': 'VP8',
    'rx_jitter': 0.29,
    'rx_packet_loss': 0,
    'rx_packets_lost': 0,
    'rx_packets_received': 28761,
    'rx_resolution': '640x480',
    'start_time': '2015-07-22T13:13:52.921269',
    'tx_bitrate': 503,
    'tx_codec': 'VP8',
    'tx_jitter': 7.68,
    'tx_packet_loss': 0,
    'tx_packets_lost': 0,
    'tx_packets_sent': 26041,
    'tx_resolution': '768x448',
    'type': 'video'
    },
    {
    'end_time': '',
    'id': '0',
    'node': '10.0.0.1'
    'rx_bitrate': 12,
    'rx_codec': 'opus',
    'rx_jitter': 0.56,
    'rx_packet_loss': 0,
    'rx_packets_lost': 0,
    'rx_packets_received': 21148,
    'rx_resolution': '',
    'start_time': '2015-07-22T13:13:52.873617',
    'tx_bitrate': 2,
    'tx_codec': 'opus',
    'tx_jitter': 0.21,
    'tx_packet_loss': 0,
    'tx_packets_lost': 0,
    'tx_packets_sent': 42386,
    'tx_resolution': '',
    '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 how an automatic deployment is progressing and 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=False
    )
print "Conferencing node status:", json.loads(response.text)
```

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=False
    )
print "System Location statistics:", json.loads(response.text)
```

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=False
    )
print "Registered aliases:", json.loads(response.text)['objects']
```

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 time format.

Up to 5000 entries each are retained for participants and conference instances. Above this, the oldest entries are deleted each time a new entry is made.

History resources

The following history resources are available via the REST API:

Component	Path
Conference instance	/api/admin/history/v1/conference/
Participant	/api/admin/history/v1/participant/
Participant media statistics	/api/admin/history/v1/participant/<participant_id>/media_stream/

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 parametrized with pagination and filter fields. For more information, see Pagination and Filtering in the Configuration API section.

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=False
    )
print "Conference history:", json.loads(response.text)['objects']
```

Note that up to 5000 conference instances will be stored in the history and by default the response will be 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=False
    )
print "Participants for conference:", json.loads(response.text)['objects']
```

Example output

```
Participants for conference: [
    {
    'bandwidth': 768,
    'call_direction': 'in',
    'call_uuid': 'b0a5b554-d1de-11e3-a321-000c29e37602',
    'conference': '/api/admin/history/v1/conference/00000000-0000-0000-0000-000000000001/',
    'conference_name': 'VMR_1',
    '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',
    'license_count': 1,
    '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',
    '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_uuid': '1c26be9c-6511-4e5c-9588-8351f8c3decd',
    'conference': '/api/admin/history/v1/conference/00000000-0000-0000-0000-000000000001/',
    'conference_name': 'VMR_1',
    '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',
    'license_count': 0,
    'local_alias': 'meet@example.com',
    'media_node': '10.0.0.1',
    'media_streams': [],
    'parent_id': '',
    'protocol': 'WebRTC',
    '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'
       }
    ]
```

Getting all participants for a time period

The following example uses filters to find all the participants whose call ended on 8 April 2015:

```
import json
import requests
response = requests.get(
    "https://<manageraddress>/api/admin/history/v1/participant/?end_time__gte=2015-04-08T00:00:00&end_
time__lt=2015-04-09T00:00:00",
    auth=('<user1>', '<password1>'),
    verify=False
    )
print "Participants for 8th April :", json.loads(response.text)['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=False
    )
print "Participants with high transmit packet loss :", json.loads(response.text)['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=False
    )
print "Participants with high receive packet loss :", json.loads(response.text)['objects']
```

Command API

The command API allows conference control operations to be invoked.

Command resources

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/
Create backup	/api/admin/command/v1/platform/backup_create/
Restore backup	/api/admin/command/v1/platform/backup_restore/
Template sync	/api/admin/command/v1/conference/sync/

Resource details

More information can be obtained for each resource by downloading the resource schema in a browser.

You may wish 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 type of data the field must contain
- 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	`success` or `error` depending on whether or not the command succeeded.
message	An informational message string if the result was `error`.
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.
- One or both of the `node` and `system_location` must be specified.
- The `node` will override the `system_location` if both are given.
- If only the `system_location` is given, a random node in that location will be 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 new call from the Conferencing Node with IP address **1.1.1.1** to the **H.323** endpoint with alias **alice**. 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**), 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',
        'protocol': 'h323',
        'system_location': 'London',
        'role': 'chair',
        },
    verify=False)
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=False)
```

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=False)
```

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 can 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=False)
```

Unmuting a participant

By submitting a POST request to the `unmute` resource URI, a previously muted conference participant can 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=False)
```

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 can 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=False)
```

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=False)
```

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=False)
```

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=False)
```

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 '?') )
```

Log output

The Pexip Infinity Management Node collates the logs from itself and all Conferencing Nodes and compiles these into the support log and the administrator log (which is a subset of the events in the support log). To view these logs from the Pexip Infinity Administrator interface, go to **Status > Support log** or **Status > Administrator log**.

You can also use a syslog server to collate logs remotely.

Log timestamps always use UTC.

The log appears in the format:

```
syslog_time system originating_time level name details
```

where:

Field	Description
`syslog_time`	In the format: `year-month-dayThour:minute:second.millisecondUTC_offset` The time that the event was logged by syslog on the originating system.
`system`	The IP address or host name of the system that sent the log message.
`originating_time`	In the format: `year-month-day hour:minute:second,millisecond` The time at which the event occurred on the originating system.
`level`	The severity of the event. The levels, in order of increasing severity, are: • DEBUG • INFO • WARNING • ERROR
`name`	The system module producing the log output.
`details`	Information about the event, as a series of name=value pairs.

Administrator log system modules

The administrator log contains information about events occurring during the normal use of the system which may be of interest to a system administrator.

The following list shows typical events that are shown in the administrator log. In addition to these, logs may also contain one-off events, such as bulk configuration import and incident reports.

For the sake of brevity, the examples shown below feature only the `name` and `details` sections of the log message.

administrator.system

Logs when the Pexip Infinity application has started on this Conferencing Node. Also logs web-based activities such as:

- import of service configuration
- deployment of a Conferencing Node
- configuration synchronization towards Conferencing Nodes
- liveness status of Conferencing Nodes

Examples

```
Name="administrator.system" Message="Starting Infinity application." Pid="1650"
Name="administrator.system" Message="Deploying worker virtual machine." Address="192.168.0.2"
Name="administrator.system" Message="Successfully deployed worker virtual machine."
Address="192.168.0.2"
Name="administrator.system.configuration" Message="Conferencing node configuration updated."
Node="conferencingnode1"
Name="administrator.system.configuration" Message="Conferencing node appears unavailable."
Node="conferencingnode1" Last-Reported="2013-06-20 14:21:35"
Name="administrator.system.configuration" Message="Conferencing node successfully updated after being
marked unavailable." Node="conferencingnode1"
Name="administrator.system.connectivity" Message="Connectivity lost to node." Node="192.168.0.2" Last-
Reported="Mon Dec  9 15:28:12 2013"
Name="administrator.system.connectivity" Message="Connectivity re-established." Node="192.168.0.2" Last-
Reported="Mon Dec  9 15:28:12 2013"
```

administrator.configuration

This module logs when:

- configuration is added or changed
- a user logs in or logs out of the system, or fails to log in
- a participant is manually dialed into a conference
- license requests are processed

Messages

```
Configuration added
Configuration changed
Configuration deleted
User logged in
User logged out
User login failed
Initiated dial command
License activation request stored for offline processing
Processed license request
```

Parameters

Parameter	Definition
User	The user who made the change.
Type	The field that was changed/added.
ID	The value that was assigned.
Remote-Address	The IP address of the system from which the request was received.
Remote-Port	The port on the system from which the request was received.

Examples

```
Name="administrator.configuration" Message="Configuration added" User="admin" Type="Virtual Meeting
Room" ID="meet.alice"
Name="administrator.configuration" Message="User login failed" User="admins" Remote-Address="192.0.2.0"
Remote-Port="53991"
```

administrator.conference

This module logs conference activity.

Parameters

Each log entry includes information in the form of a series of name=value pairs. These are defined as follows:

Parameter	Definition
Conference	The name of the service being connected to.
ConferenceAlias	The alias that was dialed in order to connect to the service.
Service-tag	The unique identifier for the service, if configured.
Service-type	This will be one of the following: • `conference` for a Virtual Meeting Room • `lecture` for a Virtual Auditorium • `two_stage_dialing` for a Virtual Reception
Location	The system location of the Conferencing Node that received the call.
Protocol	The protocol of the incoming call.
Licenses	The number of licenses allocated for the call.
Vendor	The vendor string of the endpoint that placed the call.
Signaling-node	The IP address of the Conferencing Node handling the signaling for the call.
Media-node	The IP address of the Conferencing Node handling the media processing for the call.
Duration	(for disconnect events) The duration of the call.
Detail	(for disconnect events) The reason this call was disconnected.

Examples

```
Message="Conference has been created." Conference="meet" Service-tag="" Service-type="conference"
```

Logged when a conference instance is created on a particular Conferencing Node. This occurs when the first participant attempts to join this conference on a particular Conferencing Node:

Note that the name given in the **Conference=** parameter is the name of the service, not the alias being dialed.

```
Message="Failed creating conference." Conference="meet" Service-tag=""
```

Logged if the conference creation fails.

```
Message="Conference has been stopped." Conference="meet" Service-tag="" Duration="62.234"
```

Logged when a conference has stopped on a particular conferencing node. The `Duration` of this conference is given in seconds.

```
Message="Participant attempting to join conference." ConferenceAlias="sip:meet@example.com"
Participant="sip:bob@example.com" Protocol="SIP"
```

When a participant tries to connect to a conference, this is the first thing logged (note that at this stage it is the conference alias that is logged).

```
Message="Participant failed to join conference." ConferenceAlias="sip:meet.invalid@example.com"
Participant="sip:bob@example.com" Reason="Conference not found"
```

Logged if the participant cannot join the conference. The `Reason` may vary as appropriate; the example above shows one possibility.

```
Message="Participant has joined." Conference="meet" Service-tag="" Participant="sip:alice@example.com"
DisplayName="Alice Smith" Protocol="SIP" Vendor="TANDBERG/257 (TE4.1.1.273710)" Call-
id="c15b773a81af1740@192.168.0.1" Location="London" Licenses="1" Signaling-node="192.168.0.1" Media-
node="192.168.0.2"
```

Logged when the participant joins the conference. Note at this stage it is now using the Virtual Meeting Room name. The `Call-ID` allows correlation of messages from the same participant URIs but different calls. `Location` reports the configured location of the Conferencing Node.

```
Message="Participant has disconnected." Conference="meet" Service-tag=""
Participant="sip:alice@example.com" Call-id="c15b773a81af1740@192.168.0.1" Location="London"
Licenses="1" Signaling-node="192.168.0.1" Media-node="192.168.0.1" Duration="63.123" Detail="Remote
disconnect"
```

Logged when a participant leaves the conference. Parameters are the same as for `Participant has joined`, apart from the inclusion of a `Duration` (in seconds) that they were present for. This message provides the definitive record of the length of time a call has been connected to a conference.

```
Message="Participant is presenting." Conference="meet" Service-tag=""
Participant="sip:chuck@example.com" Call-id="c15b773a81af1740@192.168.0.1"
```

Logged when participants start presenting.

```
Message="Participant has stopped presenting." Conference="meet" Service-tag=""
Participant="sip:chuck@example.com" Call-id="c15b773a81af1740@192.168.0.1"
```

Logged when participants stop presenting.

```
Message="Participant hold requested by API." Conference="meet" Service-tag=""
Participant="sip:alice@pexip.com" Call-id="a364a6e8d9012fb9@192.168.0.1"
Message="Participant resume requested by API." Conference="meet" Service-tag=""
Participant="sip:alice@pexip.com" Call-id="a364a6e8d9012fb9@192.168.0.1"
Message="Participant disconnect requested by API." Conference="meet" Service-tag=""
Participant="sip:alice@pexip.com" Call-id="a364a6e8d9012fb9@192.168.0.1"
Message="Participant mute/unmute requested by API." Conference="meet" Service-tag=""
Participant="sip:alice@pexip.com" Call-id="a364a6e8d9012fb9@192.168.0.1"
Message="Participant presentation receipt enabled by API." Conference="meet" Service-tag=""
Participant="sip:alice@pexip.com" Call-id="a364a6e8d9012fb9@192.168.0.1"
Message="Participant presentation receipt disabled by API." Conference="meet" Service-tag=""
Participant="sip:alice@pexip.com" Call-id="a364a6e8d9012fb9@192.168.0.1"
Message="Outgoing call requested" ConferenceAlias="sip:meet" Participant="sip:alice@pexip.com"
Protocol="SIP"
```

The above messages are logged when the Infinity Connect Mobile client or Infinity Connect client (or any third party using the Pexip Client API) controls participants in a conference.

```
Message="PIN entry correct." Participant="sip:bob@example.com" ConferenceAlias="sip:meet@example.com"
Call-id="c15b773a81af1740@192.168.0.1" Detail="Participant entered correct conference PIN and will be
placed into the conference."
Message="Participant entered incorrect conference PIN" RetriesRemaining="2"
Participant="sip:bob@example.com" ConferenceAlias="sip:meet@example.com" Call-
id="c15b773a81af1740@192.168.0.1"
Message="Participant exceeded PIN entry retries. Call will be disconnected."
Participant="sip:bob@example.com" ConferenceAlias="sip:meet@example.com" Call-
id="c15b773a81af1740@192.168.0.1"
Message="PIN entry timed out." Participant="sip:bob@example.com" ConferenceAlias="sip:meet@example.com"
Call-id="c15b773a81af1740@192.168.0.1" Detail="The correct PIN has not been received. Call will be
disconnected."
```

The above messages are logged when participants attempt to join a PIN-protected conference.

```
Message="Conference host timeout." Participant="sip:bob@example.com"
ConferenceAlias="sip:meet@example.com" Call-id="c15b773a81af1740@192.168.0.1" Detail="The conference
host has not joined. Call will be disconnected."
Message="Guest participant joining." Participant="sip:bob@example.com"
ConferenceAlias="sip:meet@example.com" Call-id="c15b773a81af1740@192.168.0.1" Role="guest"
Detail="Participant will be automatically placed into the conference because a conference host is
present."'
Message="First host seen in the conference; guest participant will be transferred in."
Participant="sip:bob@example.com" ConferenceAlias="sip:meet@example.com"
```

The above messages occur when guest participants are involved in a PIN-protected conference.

administrator.participantdialer

This module logs outbound calls placed from a conference or via the Pexip Distributed Gateway.

Examples

```
Message="Placing outbound call" ConferenceAlias="meet.alice@example.com" DisplayName="meet.alice"
Participant-alias="sip:bob@example.com" Protocol="SIP"
```

Logged when dialing out from a conference.

```
Message="Placing outbound call" Conference="Route to Lync:423a18c5-b06c-4654-9d34-e6b82f308b64"
ConferenceAlias="alice@example.com" DisplayName="Alice" Participant-alias="sip:carol@example.com"
Protocol="MSSIP"
```

Logged when making a call via the Pexip Distributed Gateway.

administrator.gatewayroutingrulematcher

This module logs the aliases that are matched against Gateway Routing Rules. The parsed or unparsed alias match type depends upon the rule's **Match against full alias URI** setting.

Example

```
Message="Alias matched gateway rule" Name="Route calls from Lync" Description="" Service-tag=""
Unparsed-alias="alice@example.com" Parsed-alias="alice@example.com" Matched-alias-type="Parsed-alias"
```

administrator.registration

This module logs endpoint registration activities.

Examples

```
Message="Registration added" Alias="sip:alice@example.com" Protocol="SIP" Registration-id="8695ea23-
66fb-41d2-9266-d16598f5d133" Natted="False"
```

Logged when a registration is added.

```
Message="Registration deleted" Alias="bob@example.com" Protocol="WebRTC" Registration-id="6936f540-6dbc-
40ac-9d55-5469bbf4e561" Reason="Registration expired"
```

Logged when a registration is removed.

administrator.vmrsync

This module logs VMR synchronization template activities.

Examples

```
Message="Beginning VMR Sync" template="All employees" creating_vmrs="219" deleting_vmrs="0" modifying_
vmrs="0" unchanged_vmrs="0"
```

Logged when a VMR template synchronization is initiated.

```
Message="Completed VMR Sync" template="Europe" status="sync_succeeded" vmrs_created="18" vmrs_
deleted="10" vmrs_updated="5" vmrs_unchanged="34" warnings="3" last_warning="Sync template clash:
attempt to create same conference multiple times"
```

Logged when a VMR template synchronization finishes.

Support Log system modules

Support logs provide a more detailed record of activity on the MCU.

support.sip:

Shows SIP signaling.

support.dns

Shows DNS lookups associated with SIP signaling.

support.bfcp

Shows BFCP signaling.

support.h323.q931, support.h323.ras, support.h323.h245

Shows H.323 signaling.

support.rest

Shows messages to and from the Pexip Client API (used by the Infinity Connect Mobile client, Infinity Connect client, and other third parties).

Content of signaling messages

All log lines for signaling-related messages contain source and destination IP addresses and ports.

Multi-line messages are encoded on a single line with the character sequence `^M` representing a linebreak.

In addition to observing remote IP addresses, for H323 logging, a `Uuid` is present for H.225 (RAS) and H.245 signaling messages. These come from the same IP address but different ports, and the Uuid allows you to identify that they relate to the same call.

For BFCP signaling, all log lines are tied to a SIP `Call-id`.

Most of the content of the support log is the signaling messages themselves; however it also includes logging such as TCP and TLS connection attempts, successes, and failures.

support.participant

This module logs details about conference participants, including presentation stream activation, loss of incoming video, and call media statistics when a call is completed.

Example

An example of end-of-call media statistics is as follows:

```
Name="support.participant" Message="Media Stream destroyed" Participant="sip:alice@example.com" Call-
id="890b38520c27737c@192.168.0.1" Detail="Stream 0 (audio)
    RX: rate 54kbps loss 0.77% jitter 6ms
    TX: rate 61kbps loss 0.00% jitter 2ms
        codec MP4A-LATM rate 64kbps"
Name="support.participant" Message="Media Stream destroyed" Participant="sip:alice@example.com" Call-
id="890b38520c27737c@192.168.0.1" Detail="Stream 1 (video)
    RX: rate 1085kbps loss 0.00% jitter 3ms
    TX: rate 699kbps loss 0.00% jitter 5ms
        codec H264 rate 700kbps resolution 768x448 fps 30"
Name="support.participant" Message="Current participant is the presenter"
Participant="sip:bob@example.com" Call-id="c15b773a81af1740@192.168.0.1"
Name="support.participant" Message="Presentation started" Presenter="sip:bob@example.com"
Participant="sip:alice@example.com" Call-id="890b38520c27737c@192.168.0.1"
Name="support.participant" Message="Stopped sending presentation to participant" Last-
presenter="sip:bob@example.com" Participant="sip:alice@example.com" Call-
id=890b38520c27737c@192.168.0.1"
Name="support.participant" Message="Call is on hold" Participant="sip:alice@example.com" Call-
id="890b38520c27737c@192.168.0.1"
Name="support.participant" Message="Call has resumed" Participant="sip:alice@example.com" Call-
id="890b38520c27737c@192.168.0.1"
Name="support.participant" Message="Lost incoming video" Participant="sip:alice@example.com" Call-
id="890b38520c27737c@192.168.0.1"
```

References

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
- http://tools.ietf.org/html/rfc2616
- http://tools.ietf.org/html/rfc2818

JSON

- http://en.wikipedia.org/wiki/JSON
- http://www.json.org/
- http://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://www.python.org/about/gettingstarted/
- http://docs.python-requests.org/en/latest/
- http://pysnmp.sourceforge.net