PURECLOUD

together with CS Comms

Istra 9.1 SaasEnterprise REST API

Pure Cloud Solutions Ltd.
6 The Pavillions, Amber Close,
Tamworth, B77 4RP

www.purecloudsolutions.com

T: 0333 150 6780

I. Table of contents

II. Non-disclosure agreement

All information included in this document is the entire property of Pure Cloud Solutions, and as such, must stay confidential.

Access to this document is restricted to those companies or parties having signed a Non-Disclosure Agreement (NDA) with Pure Cloud Solutions.

Diffusing information to other parties without a signed Non-Disclosure Agreement between Pure Cloud Solutions and the other party is strictly forbidden.

III. Trademarks

V. About this document

The goal of this document is to present the SaasEnterprise REST API Centile offers, and is intended for an audience of developers.

The document considers you are already familiar with the REST API over HTTP principles[1], and the JSON format[2].

Also, because of the ubiquity of the HTTP protocol, the examples given in this document are not tied to a particular programming language: instead, the command line tool *curl*[3] is preferred, since it is available for the three major operating systems, in addition to offer a convenient abstraction for the sake of readability.

VI. Introduction

The document takes a didactic approach: it follows a natural progression from the very first connection/authentication to creating, updating and deleting the exposed REST resources, without forgetting error handling.

But before delving into the details of the REST API, a few words about its goals, its security model, and the REST representation used.

A. API goals

The `SaasEnterprise` REST API has been designed with 2 goals in mind:

- Exposing a public RESTful API over HTTP, that follows the REST principles
- Exposing a simple but powerful API, to enable simple provisioning from operator point of view

This API is ideal to provision enterprises from an operator billing point of view, without the need of low level details that require user data.

The API does not deal with first name, last name or title of subscribers. It deals more with how many extensions, which feature set, the PSTNS list assigned to it, the IP Phones auto declared, etc…

As a result, the natural complement of this API is the Istra web application myTelephony, which offers the end customer a modern and simplified interface to configure all this low level details given above, plus all the other aspects that do not impact the billing (groups, devices assignment, PSTN assignments, and other telephony settings), along with some summarized information about their account.

A typical scenario made possible with these both the `SaasEnterprise` API and myTelephony is:

---

1 https://en.wikipedia.org/wiki/Representational_state_transfer 2 https://fr.wikipedia.org/wiki/JavaScript_Object_Notation
3 http://curl.haxx.se/

- Operator sets up an information system to create such Enterprise (an ordering portal) , based on the main chargeable aspects:
    - it could be public (like a public SaasService to subscribe to)
    - or private if only, say, sales teams has access to it
- This portal uses the `SaasEnterprise` API to create/modify/delete the enterprise account
- A responsible in the customer enterprise is then granted the right to fine tune the telephony accounts je just got access to., in a typical self-care manner, relieving the operator.

Using myTelephony comes with some constraints: in particular, myTelephony assumes and expects a supported type of enterprise configuration. If one administrator modifies it through webAdmin, where all controls are possible, this could lead to assumptions being violated and unexpected results would appear in myTelephony.

Now, more specifically, the API is covers these basic needs:

1. Create / read / update / delete (referred to “CRUD” operation later on):
    a. an Enterprise on the platform
    b. its number of users, by service plans
    c. the IP Phones it will use
    d. the assigned PSTN / PLMN numbers (PLMN only for mobile network)
    e. the administrator(s) granted the access to myTelephony
    f. the length of the internal dial plan
2. Listing all `SaasEnterprise` defined (audit purpose)
3. Activating / deactivating an enterprise, so that account can be setup, before actually being chargeable
4. Some other optional operations (VoIP Sites CRUD, getting URI of the myTelephony webapp…)

B. Security model

1. Admin accounts

First, only Administrator accounts can use the API, thanks to their login and password.

Second, the multi-tenancy nature of Istra is strictly followed in the API: if you create a REST resource, it will be created in the administrative domain of the administrator requesting it. You cannot, for instance, sign in as a Top Level administrator and create an Enterprise in an Service Provider administrative domain. Only the later can.

For the remaining of this document, examples given will assume the login `myLogin` and the password `myPassword`: of course you will have to use yours.

2. HTTPS only

Istra is made from scratch to run only on HTTPS, and this includes of course this REST

API. This particularly means you must have a valid SSL certificate.

If you don't, you still have the option to ignore the server certificate during the SSL handshake: while this is convenient for development or test purpose, Centile does not recommend in any way to rely on such weak SSL environment in production.

That being clear, please note the `curl` tool can ignore the invalid server certificate thanks to its option `--insecure`. As a result, everywhere a `curl` example appears, you could optionally add this option to run the example in an invalid SSL environment. And most probably, your preferred programming language has a way to deal similarly (or, alternatively, you would find a way to force to trust your invalid certificate: details vary according to your programming environment).

3. Istra Password Policy

Also a word on administrator account usage: you might fear they could be compromised, especially if the REST API is exposed to Internet for instance.

Please note it is not necessary mandatory to expose the REST API over Internet : it all depend of your network architecture and needs: perhaps only a back-end access is enough for instance.

In case the API is exposed on Internet, please know that Istra supports a Password Policy that help you defeat malicious usage:

- It can define acceptable passwords (lower vs upper case, numbers, special chars, length, complexity, etc…)
- Also, accounts can be suspended in case of too many failures in a row, and you can control the parameters that detect this (against brute force attack).

More about security is described in another document[4].

C. JSON representation

Without surprise, the API deals with the JSON format. For instance, the main resource `SaasEnterprise` uses the following representation (more details later):

```
{
    "name": "myEnterprise",
    "users": {
        "Basic": 10,
        "Gold": 4,
        "Platinum": 2
    },
    "devices": {
        "csip-snom-870": 2,
        "csip-snom-821": 4,
        "csip-snom-760": 10
    },
    "adminEmail": "me@mydomain.com",
    "dialPlanLength": 3,
    "pstns": ["0497231260","0497231261"]
}
```

You might occasionally need to manipulate other JSON objects: they will be documented later.

[4] **Namely "Istra 9.0 Security Information".**

VII. API general usage

We cover here basic HTTP considerations (including authentication), as well as error handling and the common HTTP methods used.

A. HTTP considerations

1. Host

The REST API is exposed onto a host that could take the following appearance:

1. Either `https://restletrouter.mydomain.com/`
2. Or, alternatively, `https://webadmin.mydomain.com/restletrouter` (alternatively, the fragment `webAdmin` can be replaced with `mytelephony` as well, provided it is part of your deployment).

The important fragments are:

- `restletrouter`, which is actually the API end point itself
- `mydomain.com`, which is your host name (could be an IP as well).

Using the form 1 or 2 above just depends of your deployment architecture, and we can assist you determining the best option for your case.

The remaining document will assume the generic form `https://HOST/restletrouter`, and you will have to adapt the examples to your specific environment if needed.

2. Authentication

How to authenticate against the API?

a) Basic authentication

The simple basic authentication is used: it requires sending in clear text your administrator login and password. Since Istra runs over HTTPS, this is not an issue: this traffic will be encrypted through SSL. In `curl` (fragment only) :

```
curl -u mylogin:mypassword
```

b) Using cookie-based sessions

While the REST API is completely stateless itself, a login in the API initiates a session server side.

Sessions have a default expiration time after 30 minutes of inactivity (modifiable in the Istra Password Policy, mentioned in VI.B.3).

Using a session avoids the need to authenticate each time, which starts a new session. The way to do this is:

1. To identify the cookie named `SESSIONID` returned at initial login,
2. And then to resend it at next request without the need of the user and password.

For instance, such an example with curl for the point above would give, after an initial successful authenticated request:

```
curl -b "SESSIONID=86de9f94-cff6-492d-a6e4-67c9362a888c;Path=/restletrouter"
```

(The double quotes are used to escape the semicolon that would otherwise end the command line).

Note that providing an inexistent `SESSIONID` would result in a HTTP 401 Unauthorized response.

It is recommended to use cookie-based session to avoid creating a new session at each request. However, for the sake of readability, in order to ease the reading of this document, the next `curl` examples will not use cookie-based sessions.

3. Mandatory HTTP headers

When using the API, the following headers are required.

a) Content-Type: application/json

This header is required, and specifies the representation to use in requests/responses: so far, only JSON is supported. With `curl`, it translates to the following option (fragment only):

```
curl -H "Content-Type: application/json"
```

b) X-Application: SaasAPI

This second mandatory header is a proprietary one, and control whatever part of the API you can access. Indeed, several different applications or end points can use the Istra API, and according to the REST client that sets this header, the API will give access, or not, to some REST resources.

Of course since this is sent by a potentially untrusted client, the header itself has little sense in terms of security and access control: indeed the REST API relies only on the login and password to ensure one has access.

This headers serves another goal: it just states which “part” of the API you plan to use as a client. This is a way to make sure you will not accidentally perform operations on others aspects of the API (e.g.: reporting, recording, user self care, etc…)

In our case, setting this `SaasAPI` value let us access the main REST resource `/v1/service/SaasEnterprise` which is our simplified access to provision enterprises ans users on Istra? (plus a few others as well, that will be detailed later).

This header translates to `curl` as follows (fragment only):

```
curl -H "X-Application: SaasAPI"
```

4. Recommended headers

The header below is recommended, as part of best practices.

a) Accept-encoding: gzip,deflate

One of best practices when it comes to HTTP traffic is to compress it, in order to save bandwidth. This comes as a little cost, since the compression must be done t server side, and decompression at client side, but it saves a lot since the JSON representation nature, being plain text, offers high compression ratio.

Speaking of `curl`, it can be used jointly with the `gunzip` [5]utility on the command line. For instance as follows (fragment only) :

```
curl -H "Accept-encoding:  gzip,deflate" https:// https://HOST/restletrouter/REST RESOURCE
| gunzip -c
```

For the sake of readability, examples will not include this header, but we recommend using this header in your environment.

5. HTTP methods

The API uses the following HTTP methods for the create, read, update, delete (CRUD) operations:

CRUD operation	HTTP method
Create	POST
Read	GET
Update	PUT
Delete	DELETE

Please note that not all REST resources support all method: some are read-only for instance. This will be indicated for each resource.

Again, in terms of `curl` usage, its option -X defines the method to issue:

```
curl -X HTTP_METHOD
```

Where of course `HTTP_METHOD` is the method to use.

[5] **Windows users: using the syntax given requires using a Unix type shell because of the pipe (|) usage. You could use for instance http://bliker.github.io/cmder with the full package "msysgit"**

6. Request body

Some HTTP methods require a body to be sent: the JSON payload.

When such a body is needed, one can use the `--data` option with curl, and the `@file` notation is preferred[6]:

```
curl --data @request.json
```

The `@` prefix indicates curl to use the content of the file whose name is given: here the file `request.json`, that is in the current working directory.

7. Summing it all

Cumulating all of the above gives a general `curl` command like this one:

```
curl -u mylogin:mypassword -H "X-Application: SaasAPI" -H "Content-Type: application/json"
-X HTTP_METHOD https://HOST/restletrouter/REST_RESOURCE --data @request.json
```

The argument `--data @request.json` is optional, and depends of the HTTP_METHOD used: some requests need a body, some not.

B. Error handling

Each time the API is invoked and an error occurs, you will get the following type of JSON answer response, plus an appropriate HTTP status code[7]:

```
{
"code": "ERR XYZ",
"message": "Some human readable error message detailing the XYZ code."
}
```

When a request succeeds,

Error codes and messages are given in § IX Error codes & messages.

VIII. REST resources and responses

This section lists the main REST resources offered through the `SaasEnterprise` API. You will note the API resources are prefixed with the `/v1` path fragment, which is the only version released so far.

In general, the expected HTTP response expected is a 200 OK, with an indicative JSON body being:

[6] Why using this notation? Because the same syntax of the curl examples are used for the three operating systems. Otherwise, if the body request is part of the command line, different escaping methods apply, and examples must be rewritten for Windows vs Unix shells style (OSX and GNU/Linux).

[7] https://en.wikipedia.org/wiki/List_of_HTTP_status_codes#4xx_Client_Error

```
{"code":"OK" }
```

This body, however, is not interesting to parse when in case of 200 OK and will not be repeated in the next examples.

Of course, it is much more interesting in case of an error, as detected when a 4xx or 5xx HTTP status code is returned.

A. /v1/service/SaasEnterprise

This REST resource is the main entry point to create enterprise (POST) and lists them (GET).

1. POST

a) Description

The POST creates an enterprise, based on the JSON input given.

Such an enterprise has the following default settings:

- User extension are in the 200 range
- There is no need to prefix a number to place an external (i.e: empty dial prefix on the unique PSTN gateway (unspecified behaviour if multiple gateway exist)
- IVR short numbers are in the 500 range:
  - a Conference Bridge IVR is created at extension 500, password-less.
  - a Voicemail IVR is created at extension 555
- The operator prefix is “0”
- Country-code is 33 (France)
- Language is FR (French)
- Activation status is: not activated (no external calls allowed)

Note that so far, that the assignment of the enterprise to an IPTC (the IP Telephony Container) is not managed in the API, but by configuration on webAdmin side.

b) Request

Creating an enterprise follows:

```
curl -u mylogin:mypassword -H "X-Application: SaasAPI" -H "Content-Type: application/json"
-X POST https://HOST/restletrouter/v1/service/SaasEnterprise --data @request.json
```

With the file `request.json` being the following valid JSON object:

```
{
    "name": "myEnterprise",
    "users": {
```

```
        "Basic": 2,
        "Gold": 1,
        "Platinum": 1
    },
    "devices": {
        "csip-snom-760": 2,
        "csip-snom-821": 1,
        "csip-snom-870": 1
    },
    "adminEmail": "customername@thecustomer.com",
    "dialPlanLength": 3,
    "pstns": ["0497231260","0497231261"]
}
```

NB: this example assumes the existence of the Service Plans named Basic, Gold and Platinum on the platform.

Let's breakdown the various keys of this JSON object

Key	Type	Mandatory?	Description
name	String	Mandatory	The unique name of the enterprise to create. Will be used as an ID in other request.
users	Object	Mandatory	An object requesting the creation of users : • Each key is an existing Service Plan[8] name defined in the platform • Each value is the number of user accounts to create for it The given example asks the creation of two user accounts in the `Basic` service plan, 1one in `Gold` and another one in `Premium`, for a total of four users. Each user is created with generic data, with a default extension number, and a random password. myTelephony allows the fine tuning of these defaults.
devices	Object	Mandatory	An object requesting the creation of telephony devices: • Each key is an existing device id[9] defined in the platform • Each value is how many devices to create The example asks to create two `snom 760` devices, one `snom 821` and one `snom 870`, for a total of four devices. Each device is created on the platform, unassigned. myTelephony allows the assignment to users, and input of the MAC address for auto provisioning
adminEmail	String	Mandatory	The enterprise administrator's email. Is also the login to myTelephony.

[8] A Service Plan is a set of feature accessible to an end user, already configured by the Operator in webAdmin. [9] Since new devices are integrated on a regular basis, this document does not give a definitive list that will become deprecated. Instead, and according to the devices you wish to deploy, we'll give you the ids to use separately.

			C.f. paragraph VIII.A.1.c) about passwords.
dialPlanLength	Number	Mandatory	This number indicates the length of the internal dial plan of the enterprise.
pstns	Array of Strings	Mandatory	These are all PSTN numbers to assign to the enterprise. They will be created and assigned to this enterprise. In this example, two PSTNs will be created and assigned to the enterprise (namely 0497231260 and 0497231261, typical of a French dial plan) myTelephony allows their assignment to users, groups or IVR services.
rcrs	Array of Strings	Optional Default: • [ ], or • [“DefaultRCR”]	The list of Restricted Call Rules (RCR) to assign to this enterprise. Each RCR is given by its name, and must exist on the platform. If nothing is given the API will try to apply the RCR called “DefaultRCR” if it is defined on the platform. Otherwise, no RCR are applied.

c) Administrator passwords

Passwords are randomly and automatically generated at creation time, and:

- They conform to the password policy defined in Istra (again, check our security document for more information on that).
- They are not recoverable since they are stored in a secured non reversible encrypted form.

So how do administrators and users can log in the first time?

They have to use the “Password issues?” link, present on the myTelephony login screen, in order to initiate a self-service password reset (SSPR) procedure. And of course, since this relies on email, the administrator has to provide a valid email address, hence the `adminEmail` key.

From a workflow perspective, once the enterprise is created by the ordering portal, it is expected it sends an email to the administrator, pointing him to myTelephony, where he can choose a password. (paragraph VIII.B “Users passwords

At this point, users are created, but have no email as long as the administrator does not provision them. Since the SSPR requires an email, it is mandatory the administrator sets them up first in myTelephony.

2. GET

a) Description

The GET method lists a summary of all enterprise already defined in Istra.

b) Request

There is no body for this request (will be ignored) , so client is left with the following:

```
curl -u mylogin:mypassword -H "X-Application: SaasAPI" -H "Content-Type: application/json"
-X GET https://HOST/restletrouter/v1/service/SaasEnterprise
```

c) Response

A response is formed this way:

```
{
	"enterprises": [
		{
			"activated": false,
			"entID": "456647282719151299",
			"name": "testDevices",
			"ownerAdmtiveDomainID": "AdmtiveDomain-0.1."
		},
		{
			"activated": false,
			"entID": "738142820089298879",
			"name": "myOtherEnterprise",
			"ownerAdmtiveDomainID": "AdmtiveDomain-0.2."
		}
	]
}
```

In this reponse, one can see the enterprise “myEnterprise”, resulting from the POST seen earlier, and another enterprise “myOtherEnterprise” resulting from another imaginary POST, for the sake of illustration.

Let’s see what makes this answer:

Key	Type	Description
enterprises	Array of object	This is the list of all enterprises already defined. Each element is an enterprise summary object.

In turn, the enterprise summary object is defined as follows:

Key	Type	Description
activated	Boolean	Indicates whether the enterprise is activated or not.
entID	String[10]	An internal unique ID referring to this enterprise. Given as a reference, usage reserved.
name	String	The name of this enterprise. Used to designate thie enterprise in a REST URI.
ownerAdmtiveDomainID	String	An internal unique ID referring to this enterprise administrative domain ID.

10 The API deals with long integer (64 bits) as String, in order to prevents potential Javascript client to fail when it tries to parse a “too large” long that would not fit the underlying Number implementation (based on 64-bit binary format IEEE 754).

Given as a reference, usage reserved.

B. /v1/service/SaasEnterprise/anEnterpriseName

This REST resource permits to read (GET), update (PUT) or delete (DELETE) the enterprise named `anEnterpiseName` found in the last part of this REST resource. Only this enterprise is targeted.

1. GET

a) Description

The GET method returns a detailed view of the SaasEnterprise.

b) Request

The request follows:

```
curl -u mylogin:mypassword -H "X-Application: SaasAPI" -H "Content-Type: application/json"
-X GET https://HOST/restletrouter/v1/service/SaasEnterprise/myEnterprise
```

c) Response

The response is a JSON object structured as follows:

```
{
	"activated": false,
	"adminEmail": ["customername@thecustomer.com"],
	"devices": {
		"csip-snom-760": 2,
		"csip-snom-821": 1,
		"csip-snom-870": 1
	},
	"dialPlanLength": "3",
	"entID": "88582133641153353 18",
	"name": "myEnterprise",
	"ownerAdmtiveDomainID": "0.29.",
	"pstns": [
		"1497231260",
		"1497231261"
	],
	"sites": [
		{
			"externalCallLimit": -1,
			"isDefaultSite": true,
			"name": "default",
			"siteID": "5156931585460865683"
		}
	],
	"users": {
		"Basic": 1,
		"Gold": 2,
		"Platinum": 1
	}
}
```

Description of this object:

Key	Type	Description
activated	Boolean	Indicates whether the enterprise is activated or not.
adminEmail	Array of Strings	The list of administrator emails defined for this enterprise. Normally, only one entry is expected after the original POST. However, enterprise administrator can be added through other menas (webAdmin most notably), hence the array.
devices	Object	C.f § VIII.A.1 POST
dialPlanLength	Number	C.f § VIII.A.1 POST
entID	String	C.f. § VIII.A.2 GET
name	String	C.f § VIII.A.1 POST
ownerAdmtiveDomainID	String	C.f. § VIII.A.2 GET
pstns	Array of Strings	C.f § VIII.A.1 POST
sites	Array of Objects	These are actually the list of VoIP Sites defined in the enterprise. Each object int eh array is a Site, represented by these keys: • externalCallLimit: a Number: the call limit. Is >= 0 when the limite exists, otherwise -1 for unlimited. • isDefaultSite: this Boolean tells whether this is the default site • name: name , as a String • siteID: a long integer represented as a String, which is an internal unique ID referring to this Site. Given as a reference, usage reserved.
users	Object	C.f § VIII.A.1 POST

2. PUT

a) Description

The PUT modifies the enterprise, based on the JSON input given.

b) Request

Creating an enterprise follows:

```
curl -u mylogin:mypassword -H "X-Application: SaasAPI" -H "Content-Type: application/json"
-X PUT https://HOST/restletrouter/v1/service/SaasEnterprise/myEnterprise --data
@request.json
```

With the file `request.json` being the following valid JSON object:

```
{
	"activated": true,
	"adminEmail": [
		"customername@thecustomer.com": "new.customername@thecustomer.com"
	],
	"devices": {
```

```
                "csip-snom-760": 2,
                "csip-snom-821": 0,
                "csip-snom-870": 2
        },
        "pstns": [
                "0497231270",
                "0497231271"
        ],
        "users": {
                "Basic": 2,
                "Gold": 0,
                "Platinum": 2
        }
}
```

All of these keys are optional, and:

- Any key present in the body will trigger an update in the enterprise
- Any key not present will be ignored, and do not trigger an update (nor a deletion/reset of the field).
- Any update is performed only if it is not destructive (for instance, deletion of assigned devices or PSTNS will be refused : the administrator has to free them first).

Keys behave as follows:

Key	Type	Mandatory?	Description
activated	Boolean	Optional. Default: no activation change	Changes the activation status.
adminEmail	Object	Optional. Default: {} (no adminEmail change)	This object list the administrator emails (logins) to change. A key is a currently existing email/login, and its value the new one to use. In this example,
devices	Object	Optional. Default: no devices change	C.f § VIII.A.1 POST for the syntax. The semantic of this update is to define the new set of devices present in the enterprise. For each device model given, the given counter is checked against the current number of existing devices, and: • If increased, the difference is the number of new device to create • If decreased, the difference is the number of existing devices to delete and **only** the **unassigned devices** will be deleted. Otherwise the update will be rejected. • If a device counter is left unchanged, or if a device model is not given, no update is made. In the request example, if we consider it comes just after the example used in the POST of § VIII.A.1: • The two `snom 760` are not modified • The previously existing `snom 821` is deleted **only** if it is not currently assigned (otherwise request is rejected).

			• A new `snom 870` is created, ready to be assigned in myTelephony.
pstns	Array of Strings	Optional. Default: no PSTN change	C.f. § VIII.A.1 for the syntax. The array given is actually the new list of PSTN available in the enterprise: • Any new PSTN is added (if not only existing in another enterprise, otherwise request is rejected). • Any already PSTN already existing in enterprise is left untouched. • Any PSTN already defined in the enterprise, and **not** present in this list, will be removed **only** if it is unassigned. Otherwise, request is rejected. As a result, the list must be exhaustive each time it is passed.
users	Object	Optional. Default: no users change	C.f § VIII.A.1 POST for the syntax. The semantic of this update is to define the new set of user present in the enterprise. For each service plan given, the given counter is checked against the current number of users for this service plan, and: • If increased, the difference is the number of new user to create • If decreased, the difference is the number of existing devices to delete and **only** the user previously marked as **candidate for deletion** in myTelephony will be deleted. Otherwise the update will be rejected. • If a device counter is left unchanged, or if a device model is not given, no update is made. In the request example, if we consider it comes just after the example used in the POST of § VIII.A.1: • The two `Basic` users are not modified • The previously existing `Gold` user is deleted **only** if it has been marked candidate for deletion (otherwise request is rejected). Upon deletion the associated PSTN(s) and device(s) will be released. • A new `Platinum` user is created, ready to be assigned in myTelephony. It is not possible to change a user from a service plan to another through the API (a manual update in webAdmin can be done though; myTelephony cannot).

3. DELETE

a) Description

This simply deletes the whole enterprise with all its data: users, groups, devices, PSTNs are deleted. The operation is not reversible; all data are lost forever.

b) Request

There is no body for this request (will be ignored), s o client is left with the following:

```
curl -u mylogin:mypassword -H "X-Application: SaasAPI" -H "Content-Type: application/json"
-X DELETE https://HOST/restletrouter/v1/service/SaasEnterprise/myEnterprise
```

C. /v1/service/WebappUri

This REST resource is the entry point to get information (GET) about web applications URI.

1. GET

a) Description

As mentioned earlier, the administrator in charge of the enterprise must get an email, sent by the ordering portal. How does the portal can know the URI of the web application?

It performs this GET request.

b) Request

There is no body for this request (will be ignored), h owever, the query string of the URI is used here:

```
curl -u mylogin:mypassword -H "X-Application: SaasAPI" -H "Content-Type: application/json"
-X GET https://HOST/restletrouter/v1/service/WebappUri?appName=mytelephony
```

The query string includes the key `appName`, set to `mytelephony` (value is case insensitive). It actualle requests the URI for the web application myTelephony.

Note that the query string refers only to the default name of the application, as branded by Centile, and this name must be used even if you chose to rebrand it under another name.

For instance, you could have rebranded myTelephony into SuperTelephony, so that it would be reachable at a URI like https://supertelephony.mydomain.com). Even in this case, you have to query with this default Centile name.

Also, the query string supports multiple webapp names as in this example that shows a query for both myTelephony and myIstra web application:

```
curl -u mylogin:mypassword -H "X-Application: SaasAPI" -H "Content-Type: application/json"
-X GET https://HOST/restletrouter/v1/service/WebappUri?appName=mytelephony,myistra
```

For the sake of reference, the usable web applications name follow (you may not need them all in your offering):

- acdstats
- cdr (CDR browser)
- mycallcenter
- mycompany
- myistra
- mytelephony
- webacdconsole
- webadmin
- webswitchboard (deprecated application)
- webxpad (deprecated application)
- welcomeattendantivreditor

c) Response

A HTTP status code 200 OK returns a JSON object structured this way:

```
{
	"webapp.uri.mytelephony": {
		"url": "https://mytelephony.MYDOMAIN.COM"
	},
	"webapp.uri.myistra": {
		"url": "https://myistra.aeon.centile.net"
		"urlLoginOnBehalf": "https://myistra.MYDOMAIN.COM
/?login=admin:%ADMIN%||%USER%&referer=webadmin&useragent=mobile",
	}
}
```

This object contains a key for each of the requested web application, under the form `webapp.uri.webappName`.

The value for this key is another JSON object with the mandatory key `url`, a string that is the URI of the web application as known by the platform.

The other key `urlLoginOnBehalf` is optional and is provided only when the feature login on behalf[11] is supported. This is the URL to use to perform a login on behalf of an end user, and of course the fragments %ADMIN% and %USER% must be replaced by the administrator login, and end user login.

This does not apply to myTelephony.

IX. Error codes & messages

[11] **The feature login on behalf lets an administrator log into a end user application on behalf of a end user login, using his own administrator password (similarly to the Unix su command)**

The list of error codes is listed [12]below.

```
1000 Internal server error code.
1001 Istra LightApplicationLayer exception
1002 RCR {1} is not assignable
1003 RCR {1} is not un-assignable
2000 Incorrect inputs.
2001 Service plan '{1}' does not exist.
2002 Device model '{1}' does not exist.
2003 The given administrator email '{1}' already exists, must be unique (logins are based
on email, but you know that, don't you ?).
2004 Enterprise name '{1}' does not exist.
2005 Cannot delete {1} '{2}' devices: only {3} of them are unassigned, cannot guess the
others.
2007 Cannot delete {1} user(s) in service plan '{2}': only {3} of them are marked for
deletion, cannot guess the others.
2008 Enterprise name '{1}' already exists, must be unique.
2009 Pstn(s) '{1}' already exist, must be unique.
2010  Cannot delete Ptsn(s) '{1}': they are currently assigned.
2011 The given administrator email '{1}' does not exist.
2012 You asked to create {1} extension(s), but only {2} are available on plateform, try
modify internalDialPlan or add a new one to enterprise.
2013 RCR named '{1}' does not exist
3000 Unexpected plateform pre-configuration.
3001 Cannot retrieve allowed extensions list for enterprise '{1}': check the dialplan is
correctly configured, or that licence capacity is not reached
3002 User '{1}' has unexpectedly no assigned extension, please delete this user.
3003 Cannot create a Remote Terminal: no gateway supporting them exists.
3004 Cannot create conference bridge no extension 5** is available.
3005 Unable to set SkipPassword Property on enterpriseVM
3006 Cannot Find CallQueingService
4000 Malformed Json payload.
4001 Expected type '{1}' for field '{2}'.
4001 Mandatory field '{1}' is missing.
4002 Incorrect inputs.  Unexpected key(s) found in JSON entry :  '{1}' .
```

[12] Note from the author: direct extraction from source file easy with the command :
gawk 'match($0, /new Error\(([0-9]+), \"(.+)\"\)/, a) { print a[1] " " a[2] }' ProvisioningException.java

Pure Cloud Solutions Ltd.
www.purecloudsolutions.co.uk

6 The Pavillions, Amber Close,
Tamworth, B77 4RP

T: 0333 150 6780