



together with **CS Comms**

Istra 9.1

End User REST API

```
GET /restletrouter/v1/logs/Voicemail/ HTTP/1.1
Authorization: Basic V2hhdCBkaWQgeW91IGV4cGVjdD8gOy0p
Host: restletrouter.supertelephony.net
Accept: application/json
X-Application: myIstra;17e09411-d650-4801-98db-0200660ac681
Accept-encoding: gzip, deflate
```

Pure Cloud Solutions LTD.
www.purecloudsolutions.com

6 The Pavillions
Amber Close, Tamworth
B77 4RP

I. Table of contents

I.	Table of contents	2
II.	Non-disclosure agreement.....	6
III.	Trademarks	6
V.	About this document	7
VI.	Introduction	7
A.	API goals	7
B.	Security model.....	7
1.	End users accounts	7
2.	HTTPS only.....	8
3.	Istra Password Policy	8
C.	JSON representation	9
VII.	API general usage.....	9
A.	HTTP considerations	9
1.	Host	9
2.	Authentication.....	10
a)	Basic authentication	10
b)	Using cookie-based sessions	10
3.	Mandatory HTTP headers.....	11
a)	Content-Type: application/json.....	11
b)	X-Application: myIstra	11
c)	CSRF prevention token	11
4.	Recommended headers	12
a)	Accept-encoding: gzip,deflate	12
5.	HTTP methods	13
6.	Request body.....	13

7.	Summing it all.....	13
B.	Error handling.....	14
VIII.	REST resources.....	14
A.	v1/logs/Voicemail/bean.....	14
1.	GET.....	14
a)	Description.....	14
b)	Request.....	15
c)	Response.....	15
B.	v1/logs/Voicemail/bean/id.....	16
1.	GET.....	16
a)	Description.....	16
b)	Request.....	16
c)	Response.....	17
2.	PUT.....	17
a)	Description.....	17
b)	Request.....	17
c)	Response.....	17
3.	DELETE.....	17
a)	Description.....	17
b)	Request.....	18
C.	v1/logs/Voicemail/bean/id.audioFormat.....	18
1.	GET.....	18
a)	Description.....	18
b)	Request.....	18
c)	Response.....	18
D.	v1/logs/Phone/bean.....	18
1.	GET.....	19

a)	Description	19
b)	Request.....	19
c)	Response	19
E.	v1/logs/Phone/bean/id	20
1.	GET	20
a)	Description	20
b)	Request.....	20
c)	Response	20
2.	DELETE.....	20
a)	Description	20
b)	Request.....	21
F.	v1/user/Terminal/id/placeCall	21
1.	POST	21
a)	Description	21
b)	Request.....	21
c)	Response	22
G.	v1/user/Info	22
1.	GET	22
a)	Description	22
b)	Request.....	22
c)	Response	22
H.	v1/user/IUser/id?rt=customCallerIDChoice,customCallerID.....	23
1.	GET	23
a)	Description	23
b)	Request.....	23
c)	Response	23
2.	PUT	24

a)	Description	24
b)	Request.....	24
c)	Response	24
IX.	Websocket for real time monitoring of Voice mails and Call History	24
A.	Prerequisites.....	25
B.	Starting the web socket.....	25
C.	Keep alive	25
D.	v1/service/EventListener/bean: creating an events listener.....	26
1.	POST	26
a)	Request.....	26
b)	HTTP response.....	26
2.	DELETE.....	27
a)	Request.....	27
b)	HTTP response.....	27
E.	v1/logs/Notifications/bean: counter of unread voicemails and call logs.....	27
1.	GET	27
a)	Request.....	27
b)	HTTP response.....	27
c)	Web sockets real-time response	28

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

Centile™ and Istra™ are trademarks of Centile Telecom Applications SAS.

V. About this document

The goal of this document is to present the End User 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 principles¹, and the JSON format².

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 steps to subsequent available actions, 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 `End User` REST API has been designed with 2 goals in mind:

- Expose a public RESTful API over HTTP, that follows the REST principles
- Expose a simple but powerful API, to enable simple user related telephony actions as well as some self-care actions.

This API is ideal for any HTML5 applications developments, native app development, and also mobile back-end server development.

It exposes the following features, subject to future evolutions:

- Authenticate end user with login / password
- Manage voice mails
- Manage call history
- Configure caller line id

B. Security model

1. End users accounts

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

First, only end users accounts can use the API, thanks to their login and password:

- Logins:
 - Are created / chosen by the administrator of the platform, and cannot be changed by the end user.
 - They could be an arbitrary text string (e.g. "jsmith"), unique on the platform. Email are good candidates, since they are unique by nature (e.g.: jsmith@supertelephony.net)
 - Please note another, alternative, natural login identifier is a PSTN or PLMN of the user, but of course this applies only when the user has such a public number assigned
- Passwords:
 - For telephony reasons, the passwords are made of only digits (e.g.: 2345, 7825346...).
 - Indeed, when authenticating onto an IVR (VoiceMail for instance), one has to be able to enter his password using only the phone keypad.

Of course, when a user account signs in the API, it has only access to its personal settings and publicly available information, without access to other user's settings.

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

2. HTTPS only

Istra is made from scratch to run only on HTTPS (HTTP over TLS), and this applies to 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 end user 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 depends of your system & network architecture and/or 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 helps you defeat malicious usage:

- It can define acceptable passwords (length, complexity...)
- 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)
- Sessions are time limited, configurable on the platform

More about security is described in another document.

C. JSON representation

Without surprise, the API deals with the JSON format, which is expected to be already known by the reader.

You will need to manipulate some JSON objects, that will be documented later.

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` (where the fragment `webAdmin` can be replaced with `mytelephony` or `myIstra` as well, provided they are deployed on the platform).

The important fragments are:

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

Using one or the other form 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 end user login prefixed with `ENDUSER:` and password. Since Istra runs over HTTPS, this is not an issue: this traffic will be encrypted through SSL. In `curl`, this is done through option `-u` (fragment only shown below):

```
curl -u ENDUSER: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 and renew a session each time, which would consume resources on the server. The way to do this is:

1. To identify the cookie named `myIstra_SESSIONID` returned at initial login (in header `Set-Cookie`)
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 "myIstra_SESSIONID=8204685884374655192;Path=/restletrouter"
```

(NB: the double quote char `"` is used to escape the semicolon that would otherwise end the command line shell).

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.

Important notes:

1. Providing an inexistent `SESSIONID` would result in a `HTTP 401 Unauthorized response` as expected.

2. After first authentication, it is absolutely mandatory to provide another information, namely the CSRF⁵ prevention token, to be able to place subsequent calls onto the same session. See point § 3.c) “CSRF prevention token” below

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: myIstra

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.

Instead, this header serves another goal: it 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, self care, etc...)

In our case, setting this `myIstra` value let us access the REST resources exposed in this document.

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

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

c) CSRF prevention token

The way the API prevents CSRF attacks is, one a session has been establish, to expect from the client to give the CSRF prevention token in each subsequent request.

We explain below how to get this token, and how to use it.

(1) How to get the CSRF prevention token?

⁵ CSRF: Cross Site Request Forgery. See [https://www.owasp.org/index.php/Cross-Site_Request_Forgery_\(CSRF\)](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF))

To get it, one has to submit a login request first, which is performed through a `POST` onto the resource `v1/service/Login`. A curl example would be:

```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra" -H "Content-Type: application/json" -X POST https://HOST/restletrouter/v1/service/Login
```

In case of successful authentication (`200 OK`), the HTTP response contains the `X-Application` header, which includes both the sent `myIstra` value, and also the CSRF prevention token in return. Such an example could be:

```
X-Application:myIstra;f26d2c6e-cd90-4871-8bf9-f30fb9e63d85
```

Please note the CSRF prevention token is only returned for this REST resource, only in case of successful login.

However, for future compatibility, one should expect to detect and parse the token value in each response, if provided: maybe a future evolution in the API will change the CSRF token, based on some internal security policy.

(2) How to use the CSRF prevention token?

This is very simple: the token value must be used in all subsequent calls to the API, in the same `X-Application` header already seen above. The client must send the mandatory header with both the `myIstra` value and the token, separated by the semicolon character. A curl example would be (fragment only):

```
curl -H "X-Application: myIstra;f26d2c6e-cd90-4871-8bf9-f30fb9e63d85"
```

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 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`⁶ utility on the command line. For instance as follows (fragment only):

⁶ 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/cmdr` with the full package `"msysgit"`

```
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 production 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 methods: 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 (fragment only):

```
curl -X HTTP_METHOD
```

Where of course `HTTP_METHOD` is the method to use (e.g `GET` or `POST`, etc...)

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`, or the `@file` notation which is preferred⁷ in this document:

```
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:

⁷ Why using this notation? Because the same syntax of the `curl` examples can be used for the three operating systems. Otherwise, if the body request is part of the command line using `--data`, different escaping methods apply and as a result the examples must be rewritten for Windows vs Unix shells style (OSX and GNU/Linux).

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

As already covered, 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 code⁸:

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

When a request succeeds, the `HTTP 200 OK` status code is used.

VIII. REST resources

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

In general, the expected HTTP response expected varies according to the request, but the HTTP status code `200` is always returned.

When an error occurs, the appropriate HTTP status code `4xx` or `5xx` is used, and the following JSON body is returned:

```
{
  "code": "Some message here."
}
```

The value of the `"code"` field will give some detailed information as both an API error code and human error message.

A. v1/logs/Voicemail/bean

This REST resource is the entry point to voice mails.

1. GET

a) Description

⁸ https://en.wikipedia.org/wiki/List_of_HTTP_status_codes#4xx_Client_Error

The `GET` method lists the existing voice mails for the current user.

b) Request

There is no body for this request, so client is left with the following:

```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -H "Content-Type: application/json" -X GET https://HOST/restletrouter/v1/logs/Voicemail/bean
```

Optional pagination parameters can be passed in the query string are, e.g.:
`v1/logs/Voicemail/bean?offset=0&length=20`

- `offset=0`: results are paginated. This parameter indicates from which index (zero-based) you want the list of voice mails (sorted by timestamp). Zero to start from the beginning.
- `length=20`: length of the requested page. 20 results will be returned.

c) Response

A response follows this syntax (given values just for illustration):

```
{
  "count":1,
  "items":[
    {
      "msgid":"20160219170845749-313-144",
      "filename":"20160219170845749-313-144_I_N-read.au",
      "id":"20160219170845749",
      "restUri":"v1/logs/Voicemail/bean/20160219170845749",
      "playNumber":"555;lra=144;cd_msgid=20160219170845749-313-144",
      "timestamp":"1455898125804",
      "durationSecond":5,
      "remoteNumber":"313",
      "restUriContact":"v1/directory/all/contacts/IUser-3646149346018693287",
      "remoteLabel":"John Doe",
      "type":"internal",
      "priority":"normal",
      "read":true,
      "callbackNumber":"313"
    }
  ]
}
```

In this JSON object, one sees these two main keys, count and items, described below:

Key	Type	Description
count	Integer	This is the number of results that are returned in the next field
Items	Array of voicemail JSON objects	The actual array of voice mails, each represented as a specific JSON voicemail object.

In turn, the JSON voicemail object is made of the following:

Key	Type	Description
msgid	String	An internal voicemail id, not useful for public usage (c.f. "id")

		instead).
filename	String ⁹	The actual file name containing the audio, as existing on the platform file system. Note the filename name can change according to its status read/unread.
id	String	This voice mail ID, as used in further REST calls
restUri	String	The URI used to refer to this voice mail for subsequent action
playNumber	String	This is a specific destination value to use when the client wants the Istra platform to place a call from a user phone terminal to the voicemail IVR. With this value, the later will play directly this specific voicemail, notably bypassing the main menu. Also note the way to place a call is described in section VIII.F “v1/user/Terminal/id/placeCall”
timestamp	String	The UNIX timestamp when the voicemail has been left.
durationSecond	Integer	The duration of the voicemail
remoteNumber	String	This is the actual number that left the voice mail. Could be an short number (internal call), or external number (PSTN, PLMN)
restUriContact	String	When the call is internal, this is an URI to the internal user that left the call.
remoteLabel	String	The actual name of the caller, as known by the platform (so, first name + last name if it is an internal known user, or caller line id as presented by the network otherwise)
type	String	A flag to indicate the call is internal or external.
priority	String	Sometimes, the priority can be set by the caller when he’s leaving the voice mail. This is offered by the voice mail IVR, and not often used since it is a legacy usage.
read	Boolean	Indicate whether this voicemail is read or unread.
callbackNumber	String	This is the number ready to dial (it includes dal prefix for eternal numbers if required) in order to reach the caller (internal or external number)

B. v1/logs/Voicemail/bean/id

This REST resource permits to read information (**GET**), update (**PUT**) or delete (**DELETE**) a specific voice mail.

1. GET

a) Description

The **GET** method retrieves details of a specific voice mail.

b) Request

The request follows:

⁹ The API deals with long integer ID (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 Javascript Number implementation (based on 64-bit binary format IEEE 754).


```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -H "Content-Type: application/json" -X GET https://HOST/restletrouter/v1/logs/Voicemail/bean/id
```

c) Response

The response is a JSON object. It is exactly the same syntax than an element of the `items` array seen in REST resource `v1/logs/Voicemail/bean`

2. PUT

a) Description

The `PUT` modifies a voice mail, based on the JSON input given.

b) Request

Modifying a voice mail is done through:

```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -H "Content-Type: application/json" -X PUT https://HOST/restletrouter/v1/logs/Voicemail/bean/id --data @request.json
```

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

```
{
  "read": false
}
```

Keys behave as follows:

Key	Type	Mandatory?	Description
read	Boolean	Optional.	Changes the read status to the Boolean value given. Please note it changes the filename on the file system.

The read field is the only one that can be modified so far.

c) Response

As a response, the JSON voice mail object is returned, to reflect the changes. Please note toggling the read/unread status actually changes the filename on the file system.

3. DELETE

a) Description

This simply deletes the voice mail. The operation is not reversible; the corresponding audio file is permanently deleted.

b) Request

There is no body for this request, so client is left with the following:

```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -H "Content-Type: application/json" -X DELETE https://HOST/restletrouter/v1/logs/Voicemail/id
```

C. v1/logs/Voicemail/bean/id.audioFormat

This REST resource exposes the actual audio file itself.

1. GET**a) Description**

The `GET` method retrieves the audio file the given voice mail id: the answer is not a JSON response, but the actual binary content of that file. Multiple formats are accepted.

b) Request

The request follows:

```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -X GET https://HOST/restletrouter/v1/logs/Voicemail/bean/id.ogg
```

In the request above, the `.ogg` extension file is used in order to ask the audio file in Ogg Vorbis format. Other acceptable common format are `.ogg` (Ogg Vorbis), `.aac` or `.mp4` (Advanced Audio Coding), `.au` (AU μ -law).

Note that when used from a browser, the `GET` request accepts the query string `"dl=true"`: it will instruct the server to respond with the HTTP header `"Content-Disposition:attachment;"`, which triggers the download of the file rather than invoking the browser built-in player.

c) Response

The response is the audio file content itself, in the requested format.

Please note that the appropriate HTTP header `Content-Type` will be returned (`application/ogg`, `application/mp4`, `application/au`, etc...)

When the requests contained the query string `"dl=true"`, the HTTP header `"Content-Disposition:attachment;"` will be returned as well.

D. v1/logs/Phone/bean

This REST resource is the entry point to get information (`GET`) about the user's call history.

1. GET

a) Description

The `GET` method lists the existing calls history for the current user.

b) Request

The request is:

```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -H "Content-Type: application/json" -X GET https://HOST/restletrouter/v1/logs/Phone/bean
```

The pagination parameters introduced in § A.1.b)Request are also applicable here. The query string also support the following filters (e.g.: `v1/logs/Phone/bean?offset=0&length=20&isIncoming=true`)

- `isIncoming=true`: returns only incoming calls (excludes the ones I placed)
- `isMissed=true`: returns only missed calls (excludes the ones I answered)

c) Response

A response follows this syntax (given values just for illustration):

```
{
  "count":1,
  "items":[
    {
      "restUri":"v1/logs/Phone/bean/1456922299813-53816",
      "callHistoryId":"1456922299813-53816",
      "isIncoming":false,
      "isMissed":false,
      "isRejected":false,
      "isWritten":true,
      "remoteLabel":"00302286071130",
      "remoteNumber":"000302286071130",
      "remoteType":"EXTERNAL_NUMBER",
      "ringDurationSecond":38,
      "talkDurationSecond":4,
      "timestamp":"1456922299813",
      "userEntName":"centile",
      "userNumber":"144"
    }
  ]
}
```

This follows the same structure than the voice mails (`count` integer and `items` array).

In turn, the JSON call log object is made of the following:

Key	Type	Description
<code>restUri</code>	String	The URI used to refer to this call log mail for subsequent action
<code>callHistoryId</code>	String	This call log ID, as used in further REST calls
<code>isIncoming</code>	Boolean	Was it an incoming call? (otherwise: outgoing)
<code>isMissed</code>	Boolean	Was it missed? (otherwise: answered)
<code>isRejected</code>	Boolean	<i>Not implemented</i>

isWritten	Boolean	<i>Not implemented</i>
remoteLabel	String	The actual caller id of this call, of the caller, as known by the platform (so, first name + last name if it is an internal known user, or caller line id as presented by the network otherwise)
remoteType	String	The actual type of the caller. Most of the time will be "EXTERNAL_NUMBER" (for an external PSTN or PLMN cller) or "STATION" (for an internal extension caller). For the sake of exhaustiveness, value can be one of the following string: <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 5px auto;"> STATION, TRUNK, FAX, VOICEMAIL, MEDIASERVER, REMOTE_STATION, GROUP, SPEEDDIAL, DEFAULT_ADDRESS, ACD_GROUP, PAGING_GROUP, UNKNOWN, EXTERNAL_NUMBER, MOBILE_STATION, TRUNK_MNO, TRUNK_VASPLATFORM </div>
ringDurationSecond	Integer	How long did this call ring, in seconds?
talkDurationSecond	Boolean	How long did this call last, in terms of talking time.
timestamp	String	This is the number to dial in order to reach the caller (internal or external number)
userEntName	String	Name of the enterprise to which the suser belongs
userNumber	String	Extension of the user.

E. `v1/logs/Phone/bean/id`

This REST resource is the entry point to manage one specific call log of the user's call history.

1. GET

a) Description

The `GET` method retrieves details of a specific all history entry.

b) Request

The request follows:

```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -H "Content-Type: application/json" -X GET https://HOST/restletrouter/v1/logs/Phone/id
```

c) Response

The response is a JSON object. It is exactly the same syntax than an element of the `items` array seen in REST resource `v1/logs/Phone/bean`

2. DELETE

a) Description

This simply deletes a call history entry. The operation is not reversible.

b) Request

There is no body for this request, so client is left with the following:

```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -H "Content-Type: application/json" -X DELETE https://HOST/restletrouter/v1/logs/Phone/id
```

F. v1/user/Terminal/id/placeCall

This paragraph explains how to place a call to a given destination.

A use case in the context of voicemail management is to be able to call the Voicemail IVR, and make it plays a specific voicemail instead of going though all the main menu and choices.

1. POST

a) Description

This REST resource actually makes Istra place a call from the given phone terminal to a given destination.

The client must give an existing phone terminal id: this one is retrieved through another REST resource, "v1/user/Info", exposed in next paragraph § VIII.G

b) Request

Request follows the following form:

```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -H "Content-Type: application/json" -X POST https://HOST/restletrouter/v1/user/Terminal/id/placeCall -data @request.json
```

The payload in `request.json` is actually a JSON object which has the following simple structure:

```
{
  "destination": "targetNumber"
}
```

The value `"targetNumber"` is a string containing the value of the number to dial.

In the case of the voicemail management, it is the value `"playNumber"` of a given voicemail JSON object seen in § VIII.A.1.c), so something like `"555;lra=144;cd_msgid=20160219170845749-313-144"`.

It could also be another number like an extension (e.g.: `"200"`), a PSTN prefixed by its external dial prefix (e.g. `"00497231260"` in the French national dial plan with 0 as the external dial prefix) or a PLMN

prefixed as well (e.g.: "00632654230" with same conditions) or any other number dialable from the telephony point of view (" +632654230" ...)

c) Response

Unless an error occurs (HTTP status code will tell), the expected response is a `HTTP 200 OK`, which is supposed to be returned simultaneously with the phone terminal placing the call.

G. v1/user/Info

This REST resource exposes some informations about the user: only a small subset is described here, the part actually useful to retrieve information about the user phone terminals.

1. GET

a) Description

The `GET` method retrieves details about the user.

b) Request

Request is simply:

```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -H "Content-Type: application/json" -X GET https://HOST/restletrouter/v1/user/Info
```

c) Response

The response is a JSON object. Only a small subset is described below:

```
{
  "userName": "Emmanuel Roubion",
  "istraVersion": "9.1.93",
  "istraBuildID": "647",
  "ent": {
    "fullName": "Centile",
    ...
  },
  "terminals": [
    {
      "abstractTerminalID": "8925157588004488415",
      "label": "my Desk Phone",
      ...
    }
  ],
  "user": {
    "customCallerIDChoice": "ManualSetting",
    "customCallerID": "+33489879144",
    "restUri": "v1/user/IUser/8967300633232582576"
    ...
  },
  ...
}
```

In this JSON object, the three dots character "..." indicates more keys exists but are not documented. Documented keys are:

Key	Type	Description
userName	String	The first and last name of this user
istraVersion	String	Server version
istraBuildID	String	Build number for this version
ent	String	Another JSON object, with main sub key being "fullName" and is the full name of the enterprise in which this user is defined.
terminals	Array of terminal JSON objects	The actual array of terminals that this user owns: only one most of the time, but several in case of multi-terminals deployment. Its main key are: <ul style="list-style-type: none"> "abstractTerminalID": it is a String containing the ID of this terminal. This ID must be used in order to place call as explained in § VIII.F "v1/user/Terminal/id/placeCall" "label": an other String, containing the label of this terminal. The user can change this value in his myIstra.
user	JSON object	A JSON object containing various user setting, and most notably: <ul style="list-style-type: none"> "customCallerIDChoice": values "FromPresenceState" and "manualCustomCallerID" are accepted. The later must be in use, in order to specify arbitrarily the caller line ID "customCallerID": the actual caller ID in E.164 format "restUri": this is the URI of the REST resources that exposes the user object, used in next section "v1/user/IUser/8967300633232582576"

H. v1/user/IUser/id?rt=customCallerIDChoice,customCallerID

This REST resource, with the given query string, exposes the caller id of the user.

1. GET

a) Description

The `GET` method retrieves details about the user's caller line ID

b) Request

Request is using the `rt` query string in order to reduce the scope of the request to the useful sub set:

```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -H "Content-Type: application/json" -X GET https://HOST/restletrouter/v1/user/IUser/id?rt=customCallerIDChoice,customCallerID
```

c) Response

The response is a JSON object:

```

"restUri":"v1/user/IUser/8967300633232582576",
"customCallerIDChoice":"ManualSetting",
"customCallerID":"+33489879144"
}

```

(note the `restUri` is always returned in case of the `rt` query string).

This JSON object reuses the information already seen in § G "v1/user/Info"

2. PUT

a) Description

The `PUT` can manages the caller line ID management.

b) Request

Modifying the CLI is done as follows:

```

curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -H "Content-Type: application/json" -X PUT
https://HOST/restletrouter/v1/user/IUser/id?rt=customCallerIDChoice,customCallerID --data
@request.json

```

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

```

{
  "customCallerIDChoice":"ManualSetting",
  "manualCustomCallerID":"+33489879144"
}

```

The client has to make sure:

- The field `"customCallerIDChoice"` is set to `"ManualSetting"`
- The field `"manualCustomCallerID"` is set to an acceptable caller ID in E.164 format. It must be acceptable in terms of what the Istra platform will accept according to its `custom caller id` policy.

c) Response

The response is a JSON object identical to the one submitted in request, with the additional `restUri` field.

IX. Websocket for real time monitoring of Voice mails and Call History

In this section, we describe the usage of the web socket¹⁰ scheme protocol that the API offers. Web sockets have the protocol scheme `ws://`, or `wss://` for its TLS-based equivalent: only the later is supported.

In this mode, the client subscribes to some particular events through a web socket, and is notified by a server push of the events it subscribed to. This eliminates the need for a polling mechanism.

A. Prerequisites

In order to establish a web socket connection, the client has to be already authenticated and opened a HTTP session onto the API. Then its session has the right to upgrade to a `wss://` session.

Unfortunately, the way to deal with we sockets varies according to programming langues, and the reader is left with his skills to implement web sockets in his application or back-end server.

Additionally, because the `curl` tool used so far does not support neither `ws://` nor `wss://` protocols, and because there is no popular open source tool available for the command line interface, no examples on command line are given.

B. Starting the web socket

A client has to upgrade the current HTTP connection into a web socket one. This is usually done by an HTTP `GET` request onto a specific web sockets URI, asking for upgrade in headers. For instance:

- a `GET` method
- onto URI `wss://HOST/restletrouter/ws-service/myIstra`
- with mandatory headers already seen in previous section (please note the CSRF prevention token is not required here, by design)
- with additional header `Connection: Upgrade`

Success is indicated by

- an HTTP status code `101 Switching Protocols`
- the HTTP response header `Connection: Upgrade`
- the HTTP response header `Upgrade: WebSocket`
- and of course the web socket connection opened

C. Keep alive

In order to keep alive the connection, client has to send a keep alive periodically during to prevent inactivity to expire the session.

¹⁰ **WebSocket is a protocol providing full-duplex communication channels over a single TCP connection, transported over HTTP & HTTPS. C.f. <https://en.wikipedia.org/wiki/WebSocket>**

The recommended period is 30 seconds, and the recommended JSON message is:

```
{
  "keepalive" : "true"
}
```

D. v1/service/EventListener/bean: creating an events listener

Once the web socket is opened, the client has to ask the API to create an event listener, through a HTTP request.

1. POST

Creation of a listener.

a) Request

The request is:

```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -H "Content-Type: application/json" -X POST https://HOST/restletrouter/v1/service/EventListener/bean -@request.json
```

The actual JSON payload is:

```
{
  "name": "MixedLogsL"
}
```

The JSON payload is made of:

Key	Type	Description
Name	String	A local identifier, whose value is arbitrarily decided by the client.

This actually creates a listener: there is still the step to make it subscribe to some specified events.

b) HTTP response

The HTTP response is:

```
{
  "name": "MixedLogsL",
  "restUri": "v1/service/EventListener/bean/MixedLogsL"
}
```

Which is made of:

Key	Type	Description
name	String	The given local identifier
restUri	String	The URI to refer to for this listener.

2. DELETE

A graceful termination of the subscription can be requested.

a) Request

The request is simply.

```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -H "Content-Type: application/json" -X DELETE https://HOST/restletrouter/v1/service/EventListener/bean/MixedLogsL
```

b) HTTP response

No particular response is expected.

E. v1/logs/Notifications/bean: counter of unread voicemails and call logs

Once a listener exists, the client can point to this REST resource to `GET` a snapshot of actual values, and/or specifies the listener he wants to use in order to subscribe to it.

1. GET

a) Request

For instance, the client will `GET` the actual snapshot of the unread voice mails and unread call logs, and ask the resource to be observer by a previously reated listener:

```
curl -u ENDUSER:mylogin:mypassword -H "X-Application: myIstra;CSRF_TOKEN" -H "Content-Type: application/json" -X GET https://HOST/restletrouter/v1/logs/Notifications/bean?listenerName=MixedLogsL
```

b) HTTP response

The HTTP response is the actual snapshot:

```
{
  "model": "v1/data/RestletApiSessionCounters",
  "unreadMissedCalls": 2,
  "unreadTextMessages": 0,
  "unreadVoiceMails": 1
}
```

Description of fields:

Key	Type	Description
model	String	The actual model used in this reponse. Only "v1/data/RestletApiSessionCounters" exists so far.
unreadMissedCalls	Integer	Actual number of missed calls
unreadTextMessages	Integer	Actual number of unread text messages (applies only to the instant messaging feature of the platform)
unreadVoiceMails	Integer	Actual number of unread voice mails

When the web socket is also opened, the server will push a new frame to the client to indicate the new snapshot in real time, as shown next.


c) Web sockets real-time response

When the web socket connection is established, and a listener has been created, and a new event arises in the system, it is pushed to the client as a new frame onto the web socket, as a JSON object described below:

```
{
  "listenerName": "MixedLogsL",
  "item": {
    "counters": {
      "model": "v1/data/RestletApiSessionCounters",
      "unreadMissedCalls": 0,
      "unreadTextMessages": 0,
      "unreadVoiceMails": 1
    }
  }
}
```

Description of fields:

Key	Type	Description
listenerName	String	The name of the previously created listener (useful if multiple listeners exists on different resources)
item	JSON objet	The object actually embeds a <code>counters</code> JSON object, which is the new snapshot of the events being listened to. It is exactly the same structure than the one obtained through a <code>GET</code> .



Pure Cloud Solutions Ltd
www.purecloudsolutions.com

6 The Pavillions, Amber Close
Tamworth, B77 4RP
0333 150 6780