AVAYA

OFFICELINX™

Developer's Guide for Administrative REST APIs

Version 10.7 (1) | Aug 2018.

AVAYA OFFICELINX DEVELOPER'S GUIDE FOR ADMINISTRATIVE REST

This document is a guide for using Avaya Officelinx's REST APIs. It will help developers gain a high level of understanding with the APIs and their use and deployment.

The functions of the Administrative REST API (UCBusinessRestService) interface are available with Avaya Officelinx 10.0 and higher.

Notice

While reasonable efforts have been made to ensure that the information in this document is complete and accurate at the time of printing, Avaya assumes no liability for any errors. Avaya reserves the right to make changes and corrections to the information in this document without the obligation to notify any person or organization of such changes.

Documentation disclaimer

"Documentation" means information published in varying mediums which may include product information, operating instructions and performance specifications that are generally made available to users of products. Documentation does not include marketing materials.

Avaya shall not be responsible for any modifications, additions, or deletions to the original published version of Documentation unless such modifications, additions, or deletions were performed by or on the express behalf of Avaya. End User agrees to indemnify and hold harmless Avaya, Avaya's agents, servants and employees against all claims, lawsuits, demands and judgments arising out of, or in connection with, subsequent modifications, additions or deletions to this documentation, to the extent made by End User.

Link disclaimer

Avaya is not responsible for the contents or reliability of any linked websites referenced within this site or Documentation provided by Avaya. Avaya is not responsible for the accuracy of any information, statement or content provided on these sites and does not necessarily endorse the products, services, or information described or offered within them. Avaya does not guarantee that these links will work all the time and has no control over the availability of the linked pages.

Warranty

Avaya provides a limited warranty on Avaya hardware and software. Refer to your sales agreement to establish the terms of the limited warranty. In addition, Avaya's standard warranty language, as well as information regarding support for this product while under warranty is available to Avaya customers and other parties through the Avaya Support website: https://support.avaya.com/helpcenter/getGenericDetails?detailId=C20091120112456651010 under the link "Warranty & Product Lifecycle" or such successor site as designated by Avaya. Please note that if You acquired the product(s) from an authorized Avaya Channel Partner outside of the United States and Canada, the warranty is provided to You by said Avaya Channel Partner and not by Avaya.

"Hosted Service" means an Avaya hosted service subscription that You acquire from either Avaya or an authorized Avaya Channel Partner (as applicable) and which is described further in Hosted SAS or other service description documentation regarding the applicable hosted service. If You purchase a Hosted Service subscription, the foregoing limited warranty may not apply but You may be entitled to support services in connection with the Hosted Service as described further in your service description documents for the applicable Hosted Service. Contact Avaya or Avaya Channel Partner (as applicable) for more information.

Hosted Service

THE FOLLOWING APPLIES ONLY IF YOU PURCHASE AN AVAYA HOSTED SERVICE SUBSCRIPTION FROM AVAYA OR AN AVAYA CHANNEL PARTNER (AS APPLICABLE), THE TERMS OF USE FOR HOSTED SERVICES ARE AVAILABLE ON THE AVAYA WEBSITE, HTTPS://SUPPORT.AVAYA.COM/LICENSEINFO UNDER THE LINK "Avaya Terms of Use for Hosted Services" OR SUCH SUCCESSOR SITE AS DESIGNATED BY AVAYA, AND ARE APPLICABLE TO ANYONE WHO ACCESSES OR USES THE HOSTED SERVICE. BY ACCESSING OR USING THE HOSTED SERVICE, OR AUTHORIZING OTHERS TO DO SO, YOU, ON BEHALF OF YOURSELF AND THE ENTITY FOR WHOM YOU ARE DOING SO (HEREINAFTER REFERRED TO INTERCHANGEABLY AS "YOU" AND "END USER"), AGREE TO THE TERMS OF USE. IF YOU ARE ACCEPTING THE TERMS OF USE ON BEHALF A COMPANY OR OTHER LEGAL ENTITY, YOU REPRESENT THAT YOU HAVE THE AUTHORITY TO BIND SUCH ENTITY TO THESE TERMS OF USE. IF YOU DO NOT HAVE SUCH AUTHORITY, OR IF YOU DO NOT WISH TO ACCEPT THESE TERMS OF USE, YOU MUST NOT ACCESS OR USE THE HOSTED SERVICE OR AUTHORIZE ANYONE TO ACCESS OR USE THE HOSTED SERVICE.

Licenses

THE SOFTWARE LICENSE TERMS AVAILABLE ON THE AVAYA WEBSITE, HTTPS://SUPPORT.AVAYA.COM/LICENSEINFO, UNDER THE LINK "AVAYA SOFTWARE LICENSE TERMS (Avaya Products)" OR SUCH SUCCESSOR SITE AS DESIGNATED BY AVAYA, ARE APPLICABLE TO ANYONE WHO DOWNLOADS, USES AND/OR INSTALLS AVAYA SOFTWARE, PURCHASED FROM AVAYA INC., ANY AVAYA AFFILIATE, OR AN AVAYA CHANNEL PARTNER (AS APPLICABLE) UNDER A COMMERCIAL AGREEMENT WITH AVAYA OR AN AVAYA CHANNEL PARTNER. UNLESS OTHERWISE AGREED TO BY AVAYA IN WRITING, AVAYA DOES NOT EXTEND THIS LICENSE IF THE SOFTWARE WAS OBTAINED FROM ANYONE OTHER THAN AVAYA, AN AVAYA AFFILIATE OR AN AVAYA CHANNEL PARTNER; AVAYA RESERVES THE RIGHT TO TAKE LEGAL ACTION AGAINST YOU AND ANYONE ELSE USING OR SELLING THE SOFTWARE WITHOUT A LICENSE. BY INSTALLING, DOWNLOADING OR USING THE SOFTWARE, OR AUTHORIZING OTHERS TO DO SO, YOU, ON BEHALF OF YOURSELF AND THE ENTITY FOR WHOM YOU ARE INSTALLING, DOWNLOADING OR USING THE SOFTWARE (HEREINAFTER REFERRED TO INTERCHANGEABLY AS "YOU" AND "END USER"), AGREE TO THESE TERMS AND CONDITIONS AND CREATE A BINDING CONTRACT BETWEEN YOU AND AVAYA INC. OR THE APPLICABLE AVAYA AFFILIATE ("AVAYA").

Avaya grants You a license within the scope of the license types described below, with the exception of Heritage Nortel Software, for which the scope of the license is detailed below. Where the order documentation does not expressly identify a license type, the applicable license will be a Designated System License as set forth below in the Designated System(s) License (DS) section as applicable. The applicable number of licenses and units of capacity for which the license is granted will be one (1), unless a dif-

ferent number of licenses or units of capacity is specified in the documentation or other materials available to You. “Software” means computer programs in object code, provided by Avaya or an Avaya Channel Partner, whether as stand-alone products, pre-installed on hardware products, and any upgrades, updates, patches, bug fixes, or modified versions thereto. “Designated Processor” means a single stand-alone computing device. “Server” means a set of Designated Processors that hosts (physically or virtually) a software application to be accessed by multiple users. “Instance” means a single copy of the Software executing at a particular time: (i) on one physical machine; or (ii) on one deployed software virtual machine (“VM”) or similar deployment.

License types

Designated System(s) License (DS). End User may install and use each copy or an Instance of the Software only: 1) on a number of Designated Processors up to the number indicated in the order; or 2) up to the number of Instances of the Software as indicated in the order, Documentation, or as authorized by Avaya in writing. Avaya may require the Designated Processor(s) to be identified in the order by type, serial number, feature key, Instance, location or other specific designation, or to be provided by End User to Avaya through electronic means established by Avaya specifically for this purpose.

Concurrent User License (CU). End User may install and use the Software on multiple Designated Processors or one or more Servers, so long as only the licensed number of Units are accessing and using the Software at any given time. A “Unit” means the unit on which Avaya, at its sole discretion, bases the pricing of its licenses and can be, without limitation, an agent, port or user, an e-mail or voice mail account in the name of a person or corporate function (e.g., webmaster or helpdesk), or a directory entry in the administrative database utilized by the Software that permits one user to interface with the Software. Units may be linked to a specific, identified Server or an Instance of the Software.

Named User License (NU). You may: (i) install and use each copy or Instance of the Software on a single Designated Processor or Server per authorized Named User (defined below); or (ii) install and use each copy or Instance of the Software on a Server so long as only authorized Named Users access and use the Software. “Named

User”, means a user or device that has been expressly authorized by Avaya to access and use the Software. At Avaya’s sole discretion, a “Named User” may be, without limitation, designated by name, corporate function (e.g., webmaster or helpdesk), an e-mail or voice mail account in the name of a person or corporate function, or a directory entry in the administrative database utilized by the Software that permits one user to interface with the Software.

Copyright

Except where expressly stated otherwise, no use should be made of materials on this site, the Documentation, Software, Hosted Service, or hardware provided by Avaya. All content on this site, the documentation, Hosted Service, and the product provided by Avaya including the selection, arrangement and design of the content is owned either by Avaya or its licensors and is protected by copyright and other intellectual property laws including the sui generis rights relating to the protection of databases. You may not modify, copy, reproduce, republish, upload, post, transmit or distribute in any way any content, in whole or in part, including any code and software unless expressly authorized by Avaya. Unauthorized reproduction, transmission, dissemination, storage, and or use without the express written consent of Avaya can be a criminal, as well as a civil offense under the applicable law.

Virtualization

The following applies if the product is deployed on a virtual machine. Each product has its own ordering code and license types. Note, unless otherwise stated, that each Instance of a product must be separately licensed and ordered. For example, if the end user customer or Avaya Channel Partner would like to install two Instances of the same type of products, then two products of that type must be ordered.

Third Party Components

“Third Party Components” mean certain software programs or portions thereof included in the Software or Hosted Service may contain software (including open source software) distributed under third party agreements (“Third Party Components”), which contain terms regarding the rights to use certain portions of the Software (“Third Party Terms”). As required, information regarding distributed Linux OS source code (for those products that have distributed Linux OS source code) and identifying the copyright holders of the Third Party Components and the Third Party Terms that apply is available in the products, Documentation or on Avaya’s website at: https:// support.avaya.com/Copyright or such successor site as designated by Avaya. The open source software license terms provided as Third Party Terms are consistent with the license rights granted in these Software License Terms, and may contain additional rights benefiting You, such as modification and distribution of the open source software. The Third Party Terms shall take precedence over these Software License Terms, solely with respect to the applicable Third Party Components to the extent that these Software License Terms impose greater restrictions on You than the applicable Third Party Terms.

The following applies only if the H.264 (AVC) codec is distributed with the product. THIS PRODUCT IS LICENSED UNDER THE AVC PATENT PORTFOLIO LICENSE FOR THE PERSONAL USE OF A CONSUMER OR OTHER USES IN WHICH IT DOES NOT RECEIVE REMUNERATION TO (i) ENCODE VIDEO IN COMPLIANCE WITH THE AVC STANDARD (“AVC VIDEO”) AND/OR (ii) DECODE AVC VIDEO THAT WAS ENCODED BY A CONSUMER ENGAGED IN A PERSONAL ACTIVITY AND/OR WAS OBTAINED FROM A VIDEO PROVIDER LICENSED TO PROVIDE AVC VIDEO. NO LICENSE IS GRANTED OR SHALL BE IMPLIED FOR ANY OTHER USE.

ADDITIONAL INFORMATION MAY BE OBTAINED FROM MPEG LA, L.L.C. SEE HTTP://WWW.MPEGLA.COM.

Service Provider

THE FOLLOWING APPLIES TO AVAYA CHANNEL PARTNER'S HOSTING OF AVAYA PRODUCTS OR SERVICES. THE PRODUCT OR HOSTED SERVICE MAY USE THIRD PARTY COMPONENTS SUBJECT TO THIRD PARTY TERMS AND REQUIRE A SERVICE PROVIDER TO BE INDEPENDENTLY LICENSED DIRECTLY FROM THE THIRD PARTY SUPPLIER. AN AVAYA CHANNEL PARTNER'S HOSTING OF AVAYA PRODUCTS MUST BE AUTHORIZED IN WRITING BY AVAYA AND IF THOSE HOSTED PRODUCTS USE OR EMBED CERTAIN THIRD PARTY SOFTWARE, INCLUDING BUT NOT LIMITED TO MICROSOFT SOFTWARE OR CODECS, THE AVAYA CHANNEL PARTNER IS REQUIRED TO INDEPENDENTLY OBTAIN ANY APPLICABLE LICENSE AGREEMENTS, AT THE AVAYA CHANNEL PARTNER'S EXPENSE, DIRECTLY FROM THE APPLICABLE THIRD PARTY SUPPLIER.

WITH RESPECT TO CODECS, IF THE AVAYA CHANNEL PARTNER IS HOSTING ANY PRODUCTS THAT USE OR EMBED THE G.729 CODEC, H.264 CODEC, OR H.265 CODEC, THE AVAYA CHANNEL PARTNER ACKNOWLEDGES AND AGREES THE AVAYA CHANNEL PARTNER IS RESPONSIBLE FOR ANY AND ALL RELATED FEES AND/OR ROYALTIES. THE G.729 CODEC IS LICENSED BY SIPRO LAB TELECOM INC. SEE WWW.SIPRO.COM/CONTACT.HTML. THE H.264 (AVC) CODEC IS LICENSED UNDER THE AVC PATENT PORTFOLIO LICENSE FOR THE PERSONAL USE OF A CONSUMER OR OTHER USES IN WHICH IT DOES NOT RECEIVE REMUNERATION TO: (I) ENCODE VIDEO IN COMPLIANCE WITH THE AVC STANDARD ("AVC VIDEO") AND/OR (II) DECODE AVC VIDEO THAT WAS ENCODED BY A CONSUMER ENGAGED IN A PERSONAL ACTIVITY AND/OR WAS OBTAINED FROM A VIDEO PROVIDER LICENSED TO PROVIDE AVC VIDEO. NO LICENSE IS GRANTED OR SHALL BE IMPLIED FOR ANY OTHER USE. ADDITIONAL INFORMATION FOR H.264 (AVC) AND H.265 (HEVC) CODECS MAY BE OBTAINED FROM MPEG LA, L.L.C. SEE HTTP:// WWW.MPEGLA.COM.

Compliance with Laws

You acknowledge and agree that it is Your responsibility for complying with any applicable laws and regulations, including, but not limited to laws and regulations related to call recording, data privacy, intellectual property, trade secret, fraud, and music performance rights, in the country or territory where the Avaya product is used.

Preventing Toll Fraud

"Toll Fraud" is the unauthorized use of your telecommunications system by an unauthorized party (for example, a person who is not a corporate employee, agent, subcontractor, or is not working on your company's behalf). Be aware that there can be a risk of Toll Fraud associated with your system and that, if Toll Fraud occurs, it can result in substantial additional charges for your telecommunications services.

Avaya Toll Fraud intervention

If You suspect that You are being victimized by Toll Fraud and You need technical assistance or support, call Technical Service Center Toll Fraud Intervention Hotline at +1-800-643-2353 for the United States and Canada. For additional support telephone numbers, see the Avaya Support website: https://support.avaya.com or such successor site as designated by Avaya.

Security Vulnerabilities

Information about Avaya's security support policies can be found in the Security Policies and Support section of https:// support.avaya.com/security.

Suspected Avaya product security vulnerabilities are handled per the Avaya Product Security Support Flow (https://support.avaya.com/css/P8/documents/100161515).

Downloading Documentation

For the most current versions of Documentation, see the Avaya Support website: https://support.avaya.com, or such successor site as designated by Avaya.

Contact Avaya Support

See the Avaya Support website: https://support.avaya.com for product or Hosted Service notices and articles, or to report a problem with your Avaya product or Hosted Service. For a list of support telephone numbers and contact addresses, go to the Avaya Support website: https://support.avaya.com (or such successor site as designated by Avaya), scroll to the bottom of the page, and select Contact Avaya Support.

Trademarks

The trademarks, logos and service marks ("Marks") displayed in this site, the Documentation, Hosted Service(s), and product(s) provided by Avaya are the registered or unregistered Marks of Avaya, its affiliates, its licensors, its suppliers, or other third parties. Users are not permitted to use such Marks without prior written consent from

Avaya or such third party which may own the Mark. Nothing contained in this site, the Documentation, Hosted Service(s) and product(s) should be construed as granting, by implication, estoppel, or otherwise, any license or right in and to the Marks without the express written permission of Avaya or the applicable third party.

Avaya is a registered trademark of Avaya Inc.

All non-Avaya trademarks are the property of their respective owners. Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

DEVELOPER'S GUIDE FOR ADMINISTRATIVE REST APIS

Table of Contents

9 CONFIGURING OAUTH2 FOR REST APIS

9 INTRODUCTION

9 PREREQUISITES

10 CONFIGURING OAUTH2

13 AVAYA OFFICELINX REST APIS

13 INTRODUCTION

13 HTTP COMMANDS

14 FEATURE GROUP FUNCTIONS

16 MAILBOX COMMANDS

18 ADDRESS SECTION

20 COMPANY SECTION

22 DEPARTMENT SECTION

25 REST APIS SAMPLE CODE

25 INTRODUCTION

25 SAMPLES

1 CONFIGURING OAUTH2 FOR REST APIS

Introduction

This document is a guide for using Avaya Officelinx's REST APIs. It will help developers gain a high level of understanding with the APIs and their use and deployment.

The functions of the Administrative REST API (UCBusinessRestService) interface are available with Avaya Officelinx 10.0 and higher.

Prerequisites

Avaya Officelinx version 10.0 or higher is required to use the REST APIs.

It is recommended that SSL be enabled to ensure a secure connection. Refer to the chapter Client Preparations on page 733 in Avaya's Server Configuration guide for additional details on this setup.

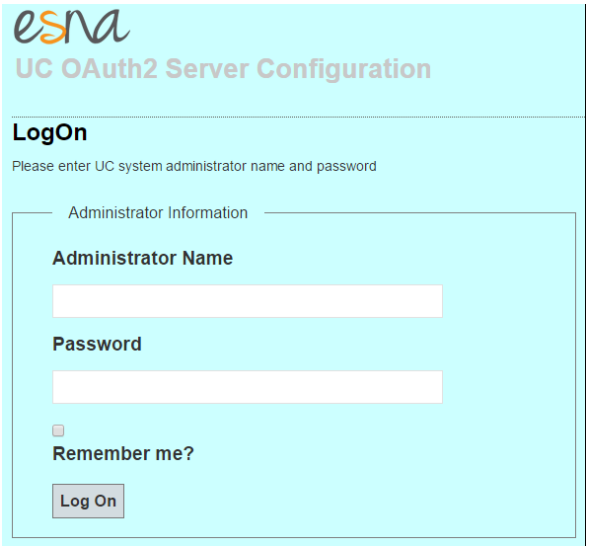
A high level of understanding of **OAuth2** authorization and deployment is expected before proceeding further into this document.

(OAuth2 is based on **IETF RFC 6749:** http://tools.ietf.org/html/rfc6749.)

Configuring OAuth2

The end point on the server needs to be configured before the REST APIs can be called.

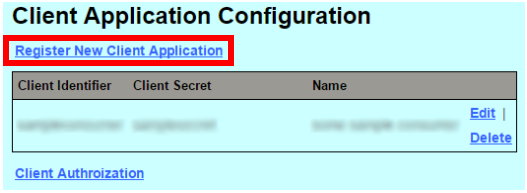
1. Log into the OAuth 2 configuration page (https://yourservername/UCOAuth2) using the Officelinx Administrative credentials.



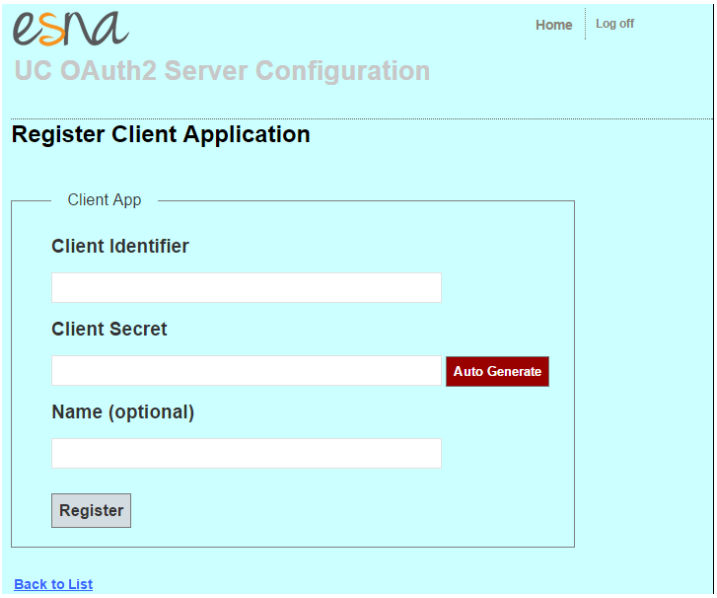
   Refer to the Officelinx Server Configuration Guide for details on setting up users.

2. The Client Application Configuration Page appears. **Client Application Registration** is completed here.

   Click Register New Client Application.



3. For a client application to communicate with the UC server through OAuth2, the client application must be registered on the UC system. Enter the required information.

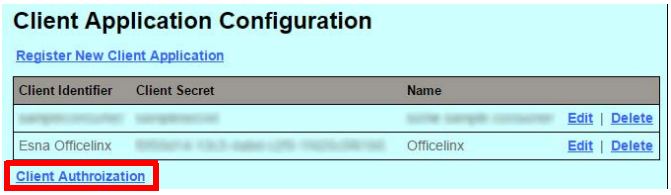


Hint: Use **Auto Generate** to randomly create a strong **Client Secret**.

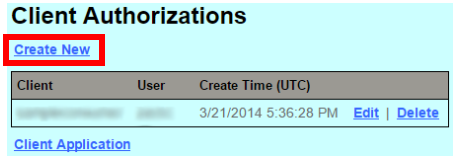
Click **Register** when finished.

4. The application has been registered and the Client Application Configuration page returns.

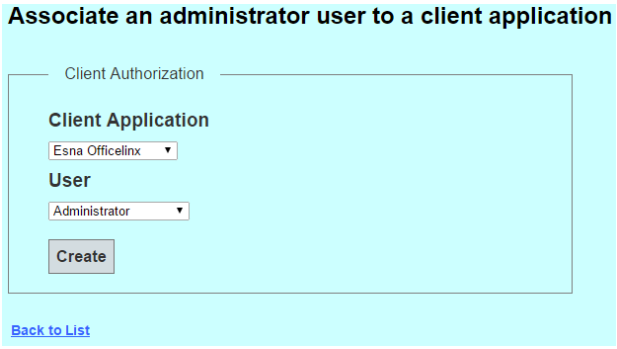
Click Client Authorization.



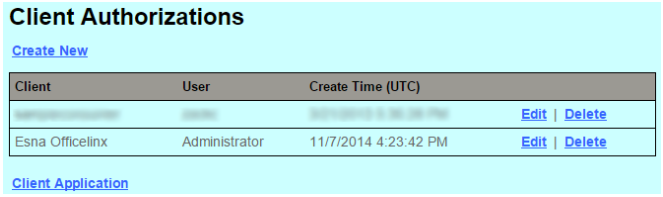
5. From the Client Authorizations page, click Create New.



6. Select an authorized user and an associated client application. This is part of the OAuth2 authorization mechanism that is required for an administrator to be registered under the specific client application.
Choose the Client Application and assign the user to it. Click **Create**.



7. The user has been created for the client application.



2 AVAYA OFFICELINX REST APIS

Introduction

REST, an acronym for REpresentational State Transfer, is becoming a powerful Web service design model. **RESTful** services are built around the HTTP protocol, and use basic HTTP commands (i.e. GET, POST, PUT, DELETE) in order to define the action. A REST Web Service uses HTTP methods to access and change resources.

> **Note**: All of the functions listed in this document assume that the OAuth2 token has already been accepted by the server. These functions ONLY work once a valid token is presented to the UC server.

Tests are performed throughout this manual using a debugging tool called **Advanced Rest Client**, which displays its output in the Chrome browser.

HTTP Commands

GET

Retrieves a resource

POST

- **Updates a Resource** when requesting the server to update one or more subordinates of the specified resource.
- **Creates Resource** when sending a command to the server to create a subordinate of the specified resource using a server-side algorithm.

PUT

- **Creates a Resource** if sending the full content of the specified resource (URL).
- **Update a Resource** when updating the full content of the specified resource.

DELETE

Deletes a resource.

Feature Group Functions

GetFeatureGroup

Parameters:

FGroupID

Logic:

The **Get()** method returns a single FeatureGroup when FGID is passed to this function.

For example in our case:

https://localhost/UCBusinessRest/api/EEAMFGroups/1

This function retrieves FeatureGroup information from the provided FGroupID. The caller should ensure the correct value for FGroup.ID prior to calling this function.

Return values:

<Response [200]> shows that the request has been accepted and will be returned with a string including the feature group name, ID, etc.

GetListOfGroupsInCompany

Parameters:

CompanyID

Logic:

The **Get()** method returns a list of FeatureGroups in a company, filtered by **CompanyID**.

For getting the ListOfFGroupsInCompany, use the browser. Navigate to the page

http://localhost:34133/api/EEAMFGroups?companyid=1

This will return the list of FGroups.

Return values:

<Response [200]> shows that the request has been accepted and a string with the feature group names and information will be attached. If the CompanyID does not exist, a **404** error will be returned.

DeleteFeatureGroup

Parameters:

FGroupID

Logic:

To delete an already existing feature group, pass only the actual ID as defined below:

https://localhost/UCBusinessRest/api/EEAMFGroups/3

Select **DELETE** in the Advanced Rest Client tool.

Return values:

When trying to delete a non-existent resource, status **404 Not Found** is returned.

If there is an exception, **Bad Request** is returned.

FeatureGroupAdd

Parameters:

INPUT (for all elements except FGroupID)

Logic:

To create new content, use **POST** to send the data to the server. This function adds a Feature Group to the system database.

Creating a new resource requires the following:

1. The client should send a **POST** request to the API to create the resource.
2. The API call must contain the necessary fields for creating a feature group.

The function automatically generates a unique FGroupID, which is the last in the database, and adds it to the FGroup Structure for **OUTPUT**.

FGroupID is a system generated unique identifier (primary key) for a mailbox. A caller can use this unique FGroupID to access a feature group.

Return values:

REST Service should return a **201 Created HTTP** response, with the URI of the newly created resource in the Location header. **Bad Request** will be returned to the client if there was an error.

FeatureGroupUpdate

Parameters:

FGroupID, Object(tFGroup)

Logic:

To update an existing Feature Group, use the **PUT** method. This function updates the elements of the specified FGroupID with the objects defined and passed through the API call. Update the feature group, and then change the element and update the required entities.

Return values:

1. Normal execution returns status **200 OK**.
2. If updating a non-existent object, the HTTP code **Not Found** will be returned to the client.
3. If there is an error, a **Bad Request** message will be returned.
4. With the return parameter of **HttpResponseMessage**, the status only is returned, not the content.
5. The Location and Feature Group are not included since they are already known (see above).

Mailbox Commands

MailboxAdd

Parameters:

tMailbox (INPUT for all elements except MailboxIDID)

(OUTPUT for MailboxID)

Logic:

To create new content, use **POST** to send the data to the server. This function adds a Mailbox to the system database.

Creating a new resource requires the following:

1. The client should send a **POST** request to the API / product to create the resource.
2. The API call must contain the necessary fields for creating a mailbox
3. When successful, the REST Service will return a **201 Created** HTTP response, with the URI of the newly created resource in the Location header.
4. If there are errors, a **Bad Request** message will be returned to the client.

The function automatically generates a unique MailboxID, which is the highest in the database, and adds it to the Mailbox Structure for **OUTPUT**.

MailboxID is a system generated unique identifier (primary key) for a mailbox. The caller can use it to access a mailbox, or use CompanyID + MailboxNumber to access a mailbox.

Note that **UserName**, **MailboxNo**, **CompanyIF**, **FirstName**, **LastName**, **FGroupID** are the necessary entities for creating a mailbox.

The function also creates following default system folders for the new Mailbox: TOP, DRAFT, INBOX, OUTBOX, SENT, TRASH.

Return values:

<Response [201]> will be returned if the mailbox is created. The content of the response shows all of the entities created with the mailbox.

MailboxUpdate

Parameters:

There are two parameters: the ID of the Mailbox to be updated, and the object(tMailbox) itself.

Logic:

To update an existing Mailbox, use the **PUT** method. This function updates the elements that are different between the two structures **prevMailbox** and **newMailbox**.

One approach is to make **newMailbox** equal to **prevMailbox**, and then modify the elements in newMailbox.

Note the following:

1. There are two parameters: the mailbox to be updated, and the object itself.
2. Normal execution of the command returns the code **200 OK**.
3. When updating a non existing object, the HTTP code **Not Found** is returned to the client.
4. A **Bad Request** message is returned if there is an error.
5. The returned parameter is of the type **HttpResponseMessage**. Only the status is returned, not the content.
6. The Location and Feature Group are not included since they are already known (see above).
7. The entities that are being updated must match the mailbox entities. Therefore, the mailbox information should be acquired before attempting any updates.

Return values:

1. Normal execution returns the status message **200 OK**.
2. When updating a non existing object, the HTTP code **Not Found** is returned to the client.
3. A **Bad Request** message is returned if there is an error.
4. The returned parameter is of the type **HttpResponseMessage**. Only the status is returned, not the content.

MailboxDelete

Parameters:

MailboxID

Logic:

To delete an existing mailbox, pass the ID as defined in the database and shown in code below:

http://localhost:34133/UCBusinessRest/api/EEAMMailboxes/1

Select DELETE in the Advanced Rest Client tool.

Return values:

<Response [200]> shows that the mailbox has been successfully deleted.

When trying to delete a non-existent resource, the status message **404 Not Found** is returned.

GetMailbox

Parameters:

MailboxID

Logic:

The **Get()** method returns a single Mailbox when Mailbox ID is passed to this function.

as shown here:

https://localhost:34133/api/EEAMMailboxes/1)

This function retrieves Mailbox information from the provided MailboxID. Ensure that the correct MailboxID is included before calling this function.

Return values:

<Response [200]> means that the request has been processed and a string with the mailbox information will be returned as well.

If the MAILBOXID is not found, an error of type **HttpResponseException** is returned. The HTTP response in this case will be a **Not Found** message.

GetListOfMailboxesByPage

Parameters:

CompanyID, PageNumber

Logic:

The **Get()** method returns a list of Mailboxes filtered by page number. To return the ListOfMailboxesFilteredByPage, call the following:

https://localhost/UCBusinessRest/api/EEAMMailboxes?companyid=1,page=1)

Return Values:

If the CompanyID is not found, an **HttpResponseException** error is returned. The HTTP response in this case will be a **Not Found** message. If PageNumber is included, the function will return the first page of mailboxes for the company.

Address Section

GetMailboxAddresses

Parameters:

MailboxID

Logic:

The **Get()** Method returns all eligible extensions for a mailbox based upon the **MailboxID**. The following URL returns the addresses associated with mailboxid = 2:

http://localhost:34133/UCBusinessRest/api/EEAMMailboxAddresses?mailboxid=2&addresstype=0

Return Values:

If the MailboxID is not found, an **HttpResponseException** error is returned. The HTTP response will be **Not Found**.

<Response [200]> will be returned along with the addresses associated with the specified mailbox when the request has been accepted and processed.

MailboxAddrAdd

Parameters:

tAddress (INPUT for all elements except AddressID)

Logic:

This function adds an address for a mailbox. In order to create new content, we are going to use the **POST** mechanism to send the data to the server. This function adds an Address into the system database associated with the specified mailbox.

Creating a new resource requires the following:

1. The client should send a **POST** request to API / product to create the resource.
2. The entities need to match the entities defined in the database
3. The function automatically generates unique AddressID, which is the maximum in the database, and adds it to the Address Structure for **OUTPUT**.

Return Values:

REST Service should return a **201 Created HTTP** response, with the URI of the newly created resource in the Location header. If there are errors, a **Bad Request** will be returned.

MailboxAddrUpdate

Parameters:

AddressID, New address(data)

Logic:

To update an existing Address, use the **PUT** method. This function updates the elements that are different between the structures **prevAddress** and **newAddress**.

First make **newAddress** equal to **prevAddress**, then modify the appropriate elements in **newAddress**.

Return Values:

Normal execution returns the status message **200 OK**.

If the object does not exist, the HTTP code **Not Found** will be returned.

If there are any errors, a **Bad Request** message is returned.

The returned parameter is of the type **HttpResponseMessage**. Only the status is returned, not the content.

The Location is not included since it is already known (see above).

MailboxAddrDelete

Parameters:

AddressID

Logic:

To delete an already existing Address we need to pass only the actual ID, as defined in the code below:

http://localhost/UCBusinessRest/api/EEAMMailboxAddresses/1

Select the **DELETE** option in **Advanced Rest Client** tool.

Return Values:

If the **ADDRESSID** is not found, an error of type **HttResponseException** is returned. The HTTP response in this case will be a **Not Found** message

Company Section

GetCompany

Parameters:

CompanyID

Logic:

The **Get()** method returns company based on provided **CompanyID**. The following URL sends back the addresses associated with companyid= 1:

http://localhost:34133/UCBusinessRest/api/EEAMCompanies/1

Return Values:

If the COMPANYID is not found, an error of type **HttpResponseException** is returned. The HTTP response in this case will be the **Not Found** message.

GetListOfCompanies

Parameters:

PBXID

Logic:

The **Get()** method returns a list of Companies. The following URL returns the companies associated with PBXID= 1.

http://localhost/UCBusinessRest/api/EEAMCompanies?pbxid=1

Return Values:

In the products do not exist, the **404 Not found** status message is returned with no content.

CompanyAdd

Parameters:

tCompany (INPUT for all elements except CompanyID)

Logic:

To create new content, use the **POST** mechanism to send data to the server. This function adds a Company to the system database.

Creating a new resource requires the following:

1. The client should send a **POST** request to API / product to create the resource.
2. The entities need to match the entities defined in the database.
3. The function automatically generates unique **CompanyID**, which is the maximum in the database, and adds it to the Company Structure for **OUTPUT**.

Return Values:

The REST Service should return a **201 Created HTTP** response, with the URI of the newly created resource in the Location header.

If there are errors, a **Bad Request** message will be returned.

If there is already a record with the same CompanyID, the function does not add the item to the database and returns **RET_RECEXISTS**.

CompanyUpdate

Parameters:

CompanyID, Data (new object)

Logic:

To update an existing Company, use the **PUT** method. This function updates the elements that are different between the structures **prevComapny** and **newCompany**.

Make **newCompany** equal to **prevCompany**, and then modify the elements in **newCompany** as required.

Return Values:

Normal execution returns the HTTP status message **200 OK**.

If the object does not exist, the HTTP code **Not Found** will be returned.

If there are any errors, a **Bad Request** message is returned.

The returned parameter is of the type **HttpResponseMessage**. Only the status is returned, not the content.

CompanyDelete

Parameters:

CompanyID

Logic:

To delete an already existing Address, pass only the actual ID as shown in the code below.

http://localhost/UCBusinessRest/api/EEAMCompanies/3

Select **DELETE** in Advanced Rest Client tool.

Note: If user passes ComapnyID=1, the company will not be deleted.

Return Values:

If the **COMPANYID** is not found, an error of type **HttpResponseException** is returned. The HTTP response in this case will be the **Not Found** message.

Department Section

GetDepartment

Parameters:

DepartmentID

Logic:

The **Get()** method returns a single department when **departmentID** is passed to this function as shown below.

https://localhost/UCBusinessRest/api/EEAMDepartments/1

Return Values:

If the ID does exist, the department name and infomation will be returned (**Response 200**).

If the **DepartmentID** is not found, an error of type **HttpResponseException** is returned. The HTTP response in this case will be the **Not Found** message.

GetListOfDepartmentsInCompany

Parameters:

CompanyID

Logic:

The **Get()** method returns a list of Departments. The following example will return all of the departments under companyid=1.

http://localhost/UCBusinessRest/api/EEAMDepartments?companyid=1

Return Values:

Normal execution returns the status message **Response200** with the list of departments in the specified company.

If the products do not exist, a **404 Not found** status message is returned with no content.

DeleteDepartment

Parameters:

DepartmentID

Logic:

To delete an existing Department, pass only the ID as shown in the code below.

http://localhost:34133/api/EEAMDepartments/1

Select **DELETE** in the Advanced Rest Client tool.

DepartmentAdd

Parameters:

tDepartment (INPUT for all elements except DepartmentID)

Logic:

To create new content, use the **POST** mechanism to send the data to the server. This function adds a Department to the system database.

Creating a new resource requires the following:

1. The client should send a POST request to API / product to create the resource.
2. The entities need to match the entities defined in the database.
3. The function automatically generates a unique **DepartmentID**, which is the last in the database, and adds it to the Department Structure for **OUTPUT**.
4. **DepartmentID** is a system generated unique identifier (primary key) for a Department. The caller can use this unique DepartmentID to access a Department.

Return Values:

The REST Service returns a **201 Created HTTP** response, with the URI of the newly created resource in the Location header. If there are any errors, a **Bad Request** message is returned.

DepartmentUpdate

Parameters:

DepartmentID, new data(Object)

Logic:

To update an existing Department, use the **PUT** method. This function updates the elements that are different between the structures **prevDepartment** and **newDepartment**.

Make **newDepartment** equal to **prevDepartment**, and then modify the elements in **newDepartment** where required.

Return Values:

Normal execution returns the status message **200 OK**.

If the object does not exist, the HTTP code **Not Found** will be returned.

If there are any errors, a **Bad Request** message is returned.

The returned parameter is of the type **HttpResponseMessage**. Only the status is returned, not the content.

3 REST APIS SAMPLE CODE

Introduction

The following sample code shows the implementation of all the functions related to feature groups and the mailbox. These examples were created using Python, but developers may use any language.

Samples

The IP addresses shown in these examples is generic (**111.111.111.111**). Replace the sample code address with the IP address of your Officelinx voice server. In a High Availability installation, this will be the IP address of the Consolidated server.

```
######## Feature Group Functions ########

# GetFeatureGroup
    r = s.get('https://111.111.111.111:443/UCBusinessRest/api/EEAMFGroups/1')

# GetListOfFGroupsInCompany
    r = s.get('https://111.111.111.111:443/UCBusinessRest/api/EEAMFGroups?companyid=1'

# DeleteFeatureGroup
    r = s.delete('https://111.111.111.111:443/UCBusinessRest/api/EEAMFGroups/3')

# FeatureGroupAdd
    payload = {
        "COMPANYID": 1,
        "FGNUMBER": 56,
        "NAME": "TestFG"
    }
    headers = {'content-type': 'application/json'}   #important, otherwise content-type is form encoded
    r = s.post('http://111.111.111.111/UCBusinessRest/api/EEAMFGroups/', data = json.dumps(payload), headers =
      headers)

# FeatureGroupUpdate
    r = s.get('https://111.111.111.111:443/UCBusinessRest/api/EEAMFGroups/2')
    headers = {'content-type': 'application/json'}
    payload = r.json()
    payload['Name'] = 'New ' + payload['Name']
    r = s.put('https://111.111.111.111:443/UCBusinessRest/api/EEAMFGroups/2', data = json.dumps(payload), headers =
      headers)
```

######## Mailbox Functions ########

 # MailboxAdd

```
payload = {
    "UserName": "test@esna.com",
    "MailboxNo": "55",
    "CompanyID": 1,
    "FirstName": "TestFirstNAme",
    "LastName": "TestLastName",
    "FGroupID": 1
}
headers = {'content-type': 'application/json'}   #important, otherwise content-type is form encoded
r = s.post('http://111.111.111.111/UCBusinessRest/api/EEAMMailboxes/', data = json.dumps(payload), headers =
  headers)
```

 # MailboxDelete

```
r = s.delete('https://111.111.111.111:443/UCBusinessRest/api/EEAMMailboxes/3')
```

 # MailboxUpdate

```
r = s.get('http://localhost:10784/api/EEAMMailboxes/4');
payload = r.json()
payload['FirstName'] = 'New ' + payload['FirstName']
r = s.put('http://localhost:10784/api/EEAMMailboxes/4', data = json.dumps(payload), headers = headers)
```

 # GetMailbox

```
r = s.get('https://111.111.111.111:443/UCBusinessRest/api/EEAMMailboxes/4')
logger.debug('response after REST api call: %r' % r.__dict__)
jsonObject = r.json()
```

 # GetListOfMailboxesByPage

```
r = s.get('https://111.111.111.111:443/UCBusinessRest/api/EEAMMailboxes?companyid=1,page=1')
logger.debug('response after REST api call: %r' % r.__dict__)
```

 # GetDepartment

```
# r = s.get('https://111.111.111.111:443/UCBusinessRest/api/EEAMDepartments/1', verify = False)
```

```
######## Department Functions ########

# GetListOfDepartmentsInCompany
    # r = s.get('https://111.111.111.111:443/UCBusinessRest/api/EEAMDepartments?companyid=1', verify = False)

# DeleteDepartment
    # r = s.delete('https://111.111.111.111:443/UCBusinessRest/api/EEAMDepartments?id=1&parentid=0', verify = False)

# DepartmentAdd
    payload = {
        "CompanyID": 1,
        "DepartmentName": 'APITEST'
    }
    headers = {'content-type': 'application/json'}   #important, otherwise content-type is form encoded
    r = s.post('https://111.111.111.111:443/UCBusinessRest/api/EEAMDepartments', data = json.dumps(payload),
      headers = headers, verify = False)

# DepartmentUpdate
    r = s.get('https://111.111.111.111:443/UCBusinessRest/api/EEAMDepartments/1', verify = False);
    headers = {'content-type': 'application/json'}   #important, otherwise content-type is form encoded
    payload = r.json()
    payload['DepartmentName'] = 'New' + payload['DepartmentName']
    r = s.put('https://111.111.111.111:443/UCBusinessRest/api/EEAMDepartments/1', data = json.dumps(payload),
      headers = headers, verify = False)
```

```
######## Authorization#######
######## Note that this is sample code and must be modified for each system #######

import config
import logging
from datetime import timedelta, datetime
from requests_oauth2 import OAuth2
import requests, json
import inspect

class OAuth2Token(object):
    def __init__(self, access_token, expires_in, refresh_token, scope, token_type = u'bearer'):
        self.access_token = access_token
        self.expirationUtc = datetime.utcnow() + timedelta(seconds=expires_in-5)
        self.refresh_token = refresh_token
        self.scope = scope
        self.token_type = token_type

def fetch_UC_oauth2token():
    # get OAuth2 token from remote url
    func = '[' + inspect.currentframe().f_code.co_name + ']'

    url = "https://111.111.111.111/UCOAuth2/"

    client_id = "SampleConsumer"
    client_secret = "SampleSecret"
    username = "SampleUserName"
    pwd = "SamplePassword"

    oauth2_handler = OAuth2(client_id, client_secret,
        url,  '', 'oauth2/authorize', 'oauth2/token' )
    response = oauth2_handler.get_token('', verify=True, grant_type='password', username=username, password=pwd,
      scope='http://localhost/')
# scope should be hardcode to 'http://localhost/' for tolee rest api
token = None
if response.get('access_token') is None:
    logger.debug('{0} did not get access_token'.format(func))
else:
    token = OAuth2Token(response['access_token'], response['expires_in'], response['refresh_token'],
      response['scope'],response.get('token_type'))
    logger.info('{0} created token object: {1}'.format(func, token.__dict__))

return token;

def refresh_UC_oauth2token(refresh_token):
    # refresh OAuth2 token from remote url
    func = '[' + inspect.currentframe().f_code.co_name + ']'
```

```
    url = "https://111.111.111.111/UCOAuth2/"

    client_id = "SampleConsumer"
    client_secret = "SampleSecret"
    username = "SampleUserName"
    pwd = "SamplePassword"

    oauth2_handler = OAuth2(client_id, client_secret,
        url,  '', 'oauth2/authorize', 'oauth2/token' )

response = oauth2_handler.get_token('', verify=True, grant_type='refresh_token', refresh_token = refresh_token)
logger.debug('{0} after refreshToken: {1}'.format(func, response))
token = None
if response.get('access_token') is None:
    logger.debug('{0} refreshing did not get access_token'.format(func))
else:
    token = OAuth2Token(response['access_token'], response['expires_in'], response['refresh_token'],
      response['scope'],response.get('token_type'))
    logger.info('{0} refreshing created token object: {1}'.format(func,token.__dict__))

return token;

def get_UC_oauth2token():
    func = '[' + inspect.currentframe().f_code.co_name + ']'
    save_token = False
    token = read_UC_oauth2token()

    if token is None or \
        token.access_token is None or \
        token.access_token == '' or \
        token.refresh_token is None or \
        token.refresh_token == '' :
        logger.debug("{0} token is not valid, fetching token".format(func))
        token = fetch_UC_oauth2token()
        logger.debug("{0} fetched token: {1}".format(func, token))
        save_token = True
    else:
    if token.expirationUtc < datetime.utcnow():
        logger.debug("{0} token expired, refreshing token".format(func))
        token = refresh_UC_oauth2token(token.refresh_token)
        save_token = True
        logger.debug("{0} after refreshing token".format(func))
        if token is None:
          logger.debug("{0} refreshing token failed, getting new token".format(func))
          token = fetch_UC_oauth2token()
          save_token = True
```

```
        if token is not None and save_token:
            logger.debug("{0} save token to configuration file: {1}".format(func, token.__dict__))
    return token

if __name__ == "__main__":
        for i in range(1, 76458):
            logger.debug('loop {0}'.format(i))
            get_UC_oauth2token()
```

OAuth2####

```
import requests
from urllib import quote, urlencode
from urlparse import parse_qs
try:
    import simplejson as json
except ImportError:
    import json

class OAuth2(object):
    authorization_url = '/oauth/authorize'
    token_url = '/oauth/token'

    def __init__(self, client_id, client_secret, site, redirect_uri, authorization_url=None, token_url=None):
        """
        Initializes the hook with OAuth2 parameters
        """
        self.client_id = client_id
        self.client_secret = client_secret
        self.site = site
        self.redirect_uri = redirect_uri
        if authorization_url is not None:
          self.authorization_url = authorization_url
        if token_url is not None:
          self.token_url = token_url

    def authorize_url(self, scope='', **kwargs):
        """
        Returns the url to redirect the user to for user consent
        """
        oauth_params = {'redirect_uri': self.redirect_uri, 'client_id': self.client_id, 'scope': scope}
        oauth_params.update(kwargs)
        return "%s%s?%s" % (self.site, quote(self.authorization_url), urlencode(oauth_params))

    def get_token(self, code, verify=True, **kwargs):
        """
        Requests an access token
        """
        url = "%s%s" % (self.site, quote(self.token_url))
        data = {'redirect_uri': self.redirect_uri, 'client_id': self.client_id, 'client_secret': self.client_secret, 'code': code}
        data.update(kwargs)
        response = requests.post(url, data=data, verify=verify)

    if isinstance(response.content, basestring):
        try:
          content = json.loads(response.content)
        except ValueError:
```

```
            content = parse_qs(response.content)
        else:
            content = response.content

    return content
```