A MITEL
PRODUCT
GUIDE

Unify OpenScape Xpert V8

Turret API

Integration Guide
08/2024

A31003-X2080-N100-01-7620

Mitel

Notices

The information contained in this document is believed to be accurate in all respects but is not warranted by Mitel Europe Limited. The information is subject to change without notice and should not be construed in any way as a commitment by Mitel or any of its affiliates or subsidiaries. Mitel and its affiliates and subsidiaries assume no responsibility for any errors or omissions in this document. Revisions of this document or new editions of it may be issued to incorporate such changes. No part of this document can be reproduced or transmitted in any form or by any means - electronic or mechanical - for any purpose without written permission from Mitel Networks Corporation.

Trademarks

The trademarks, service marks, logos, and graphics (collectively “Trademarks”) appearing on Mitel’s Internet sites or in its publications are registered and unregistered trademarks of Mitel Networks Corporation (MNC) or its subsidiaries (collectively “Mitel), Unify Software and Solutions GmbH & Co. KG or its affiliates (collectively “Unify”) or others. Use of the Trademarks is prohibited without the express consent from Mitel and/or Unify. Please contact our legal department at iplegal@mitel.com for additional information. For a list of the worldwide Mitel and Unify registered trademarks, please refer to the website: http://www.mitel.com/trademarks.

Contents

1 Introduction

Application interface (API) for OSX Client has been existed for many years, allowing 3rd party companies to integrate with OpenScape Xpert product.

The goal of the new Client API is to address shortcomings of the already existing API interface, like:

- Proprietary protocol, which is not easy to change
- 3 rd parties have to implement their own protocol parser in every language they intend to use
- Insecure transport layer

The need of an API for huge projects is very common in the industry, which has been recognized by big software companies. Therefore modern, ready-made solutions have been evolved which are freely available to use.

Using these technologies helps software product development to concentrate on defining the *service* for 3rd parties, rather than enhancing a proprietary solution towards the increasing technology and security needs.

This document describes the architecture and the currently exposed functionalities of the new *OSX Client API*, and can be used as a reference for 3rd parties, who intend to integrate with OSX Client.

2 OSX Client API technology

The new OSX Client API is based on Apache Thrift TM technology.

2.1 What is Apache Thrift TM?

Quote from https://thrift.apache.org/

“The Apache Thrift software framework, for scalable cross-language services development, combines a software stack with a code generation engine to build services that work efficiently and seamlessly between C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, OCaml and Delphi and other languages.

Apache Thrift allows you to define data types and service interfaces in a simple definition file. Taking that file as input, the compiler generates code to be used to easily build RPC clients and servers that communicate seamlessly across programming languages. Instead of writing a load of boilerplate code to serialize and transport your objects and invoke remote methods, you can get right down to business.”

2.2 Advantages

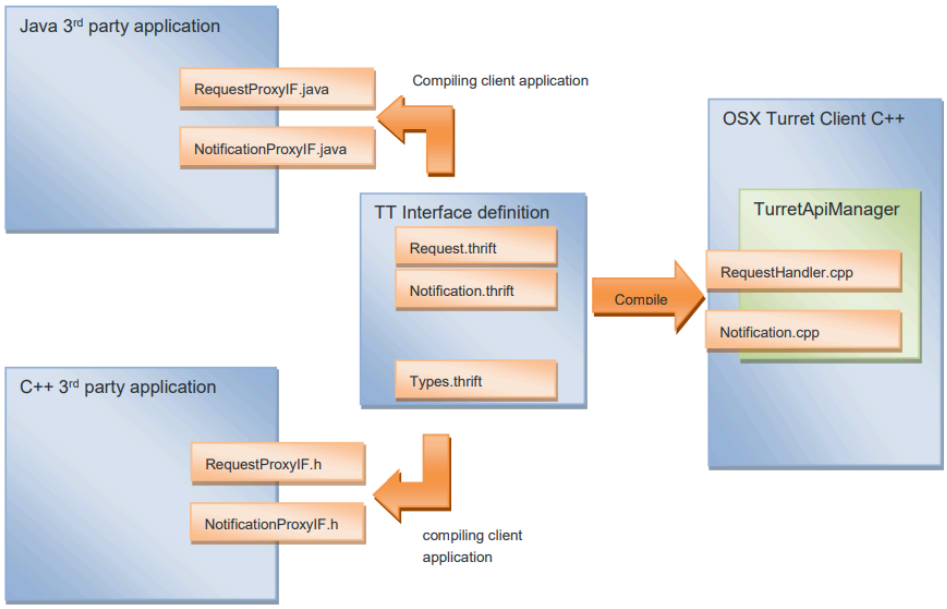
- One can connect to the OSX Turret using his/her favourite programming language without implementing custom CTI protocol (See language support: https://thrift.apache.org/docs/features)
- Language bindings are generated from IDL file (.thrift file).
- Various transport support (http, tcp)
- Secure: provides the possibility for Encrypted transport using SSL
- Various protocol options (JSON, binary, XML, plain text, etc,) for serialization/ de-serialization
- Seamless network socket connection handling
- Seamless serialization / de-serialization

2.3 Notes

- To have a duplex communication interface, two interfaces are needed on client and server side, respectively:
  - Request
  - Notification
- The consequence is that both the 3rd party client and the Turret act as server and client as well (therefore two sockets are opened).

3 OSX Client API Architecture

Consider the following figure (it contains only the API handler part from OSX Client)



The services and supporting components are defined in the *TurretApi.thrift* file:

- *TurretRequestService*: for example login, logout, disconnect
- *TurretNotificationService*: asynchronous messages coming from the Turret, like line state changes
- Types, structs and enumerations defined for OSX Client API, like *LoginState, LineState* etc.
- *Error codes are defined as constants on the API*

During official production of OSX software, Apache Thrift TM compiler generates server side bindings for the OSX Turret client in C++ language. These are incorporated into the OSX Client software and handle the communication between the 3rd party application and the OSX client.

The .thrift interface files are distributed for 3rd party development. The client application needs to use Apache thrift compiler to generate language specific bindings which then have to be included in the client application build process.

4 OSX Client API Architecture

The OSX solution provides wide range of features. It's not the scope of this document to describe all functionalities of OSX, however the ones which are exposed on the API are described in the HTML-based documentation.

For further information for features, please refer to the *OpenScape Xpert Feature Description manual*.

4.1 Profile / configuration

The basic usage of OSX client starts with loading a profile. The API provides functionality to login to a specific profile, logout from the actually loaded profile and query the current login state of the OSX client.

NOTICE: Currently, there is no possibility to query the available configurations through API.

4.2 Basic call related functionalities

The following basic call related functionalities can be accomplished using the API.

4.2.1 Placing a call

- using a specific line to initiate the call from, or let the OSX client select one
- select the Speech device
- dial the destination number

4.2.2 Answering an incoming call on a line, on a specific device

- select the Speech device
- select the Line, or pick up the call which is at the top of the call queue

4.2.3 Disconnect a call, on a specific device

- select the Speech device which handles the call that has to be disconnected

4.3 Call queue

Call queue is a feature where incoming call, and held lines can be monitored (depending on the configuration).

The API provides

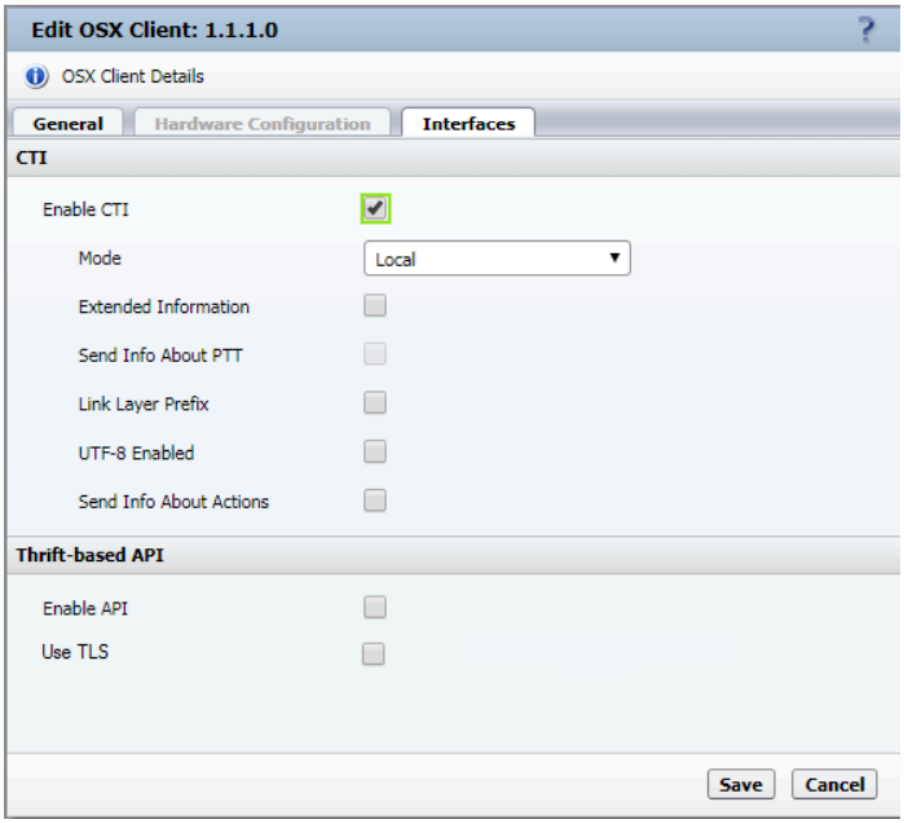
- Notification, when the incoming call at the top of the call queue has changed

5 Setting up thrift API

5.1 Enable API connections for Turret

- Thrift API can be enabled on the System Manager's Client window (OSX Clients -> Edit OSX client - > Interfaces -> Enable API).

  API connection is by default unsecured. To enable TLS for the API connection, check the "Use TLS" checkbox. After the transmission of the changes, all the existing API connections will be dropped and the clients have to reconnect using appropriate certificates. Regarding OSX clients, the certificates are placed in \api_tls or /etc/cert/htems/api_tls. The API connections will be also dropped if the "Use TLS" or "Enable API" flag is changed from enabled to disabled.



The changes have to be released before any 3rd party software could connect to the OSX client.

6 Turret API Description

You can see a detail description of the TurretApi by clicking on this link:

https://networks.unify.com/openscape-documentation/V6R1/OSX/Turret_API/Turret_Thrift_API_reference.html#Const_InvalidSpeechUnit

7 Limitations

- *Only one* 3 rd party server/client application can connect to an OSX client, but this client can notify several 3rd party client/server applications. “Multiple connections” is an enhancement possibility for the future.
- The turret’s server uses the port 9007 at the moment.
- The actually used programming language might have further limitations, which is not covered in this document. Please refer to the https://thrift.apache.org/, before using a specific programming language.