Thello V3

API Documentation

Introduction 3

Management API 3

Signal-R API 3

Initializing the SignalR connection 3

SignalR client API 5

Thello functional Blocs compatibles with SignalR API 7

Queue bloc 7

# Introduction

API (Application Programming Interface) allow external (third-parties) software to access and control Thello behavior.

Depending on usage, different APIs are available

* Management API: is intended for back-office tools to configure Thello, i.e. create & manage tenants from an administrative point of view: creating new tenants, managing licensing and extracting billing information. This API is typically used by ERP to configure subscriptions and create invoices
* SignalR API: this API is intended to control in real-time the flow of calls. SignalR is a technology that allows bidirectional communication between clients such as web-clients and Thello

# Management API

This API is reachable on following url: https://api.v3.thello.cloud

# Signal-R API

## Initializing the SignalR connection

Signal-R API directly connects to Thello’s core service. Therefore the connection url depends on the Tiergroup executing the tenant’s call-flow.

Before accessing the API, the client must get authentication credentials through a JWT token or use a valid user API key in calls headers.

### JWT Bearer authentication

Authentication can be done on the Thello management API by calling Authenticate URL with JSON data {“username”: “john\_smith”, “password”: “secret\_password” }

Example of javascript code:

async function login() {

const username = $("#usernameInput").val();

const password = $("#passwordInput").val();

console.log('Current JWT Token:', token);

const response = await fetch('https://api.v3.thello.cloud/api/Authentication/Authenticate', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ userName: username, password: password })

});

if (response.ok) {

const data = await response.json();

token = data.access\_token;

console.log('JWT Token:', token);

connection.start().catch(err => console.error(err.toString()));

} else {

console.error('Authentication failed');

token = null;

}

}

Once the JWTToken is received it can be used to establish an authenticated Signal-R connection.

The SignalR service is not located on the same url as the management API and is tenant specific. The signal-R service base url can be obtained through the Management API, using “/api/Tenant/mine” request and the SignalRUrl field of the returned structure

The Signal-R connection can be created using:

connection = new signalR.HubConnectionBuilder()

.withUrl(thelloSignalRUrl, {

accessTokenFactory: () => token

})

.withAutomaticReconnect()

.configureLogging(signalR.LogLevel.Debug)

.build();

Where thelloSignalRUrl is the url obtained earlier and token is the JwtToken

### API Key authentication

Each Thello user can have one or more API keys to run calls on services and signalR. In the user profile screen in control panel, open the “API Keys” tab. The list of current key is displayed. A button “+ New” is also available to generate a new key.

Once key is known, it can be used in HTTP headers of SignalR calls.

connection = new signalR.HubConnectionBuilder()

.withUrl(thelloSignalRUrl, {

headers: { "X-API-KEY": "<the\_api\_key>" }

})

.withAutomaticReconnect()

.configureLogging(signalR.LogLevel.Debug)

.build();

## SignalR client API

public async Task SubscribeBlocNotifications(string blocId)

Subscribes for event sent by a specific bloc

Notifications are received by the signal-r client on the following callback:

Task BlocEvent(string blocId, string eventName, string callId, string fromName, string fromNumber, string to, string data, string callContextId);

Where:

* blocId the bloc ID as specified in SubscribeBlocNotifications, this allows a single callback to receive notifications from different blocs
* eventName: the notification event
* callId: SipCallId for the current channel
* fromName: CallerID name
* fromNumber: CallerID number
* to: Called number or extension
* data: optional data dependent of the bloc type sending the notification
* callContextId: A contextId that can be used with SetVariable or GetVariable

public async Task UnsubscribeBlocNotifications(string blocId)

Unsubscribes callback notification of event bloc that would have been previously subscribed using SubscribeBlocNotifications()

public async Task BlocNotify(string blocId, string eventName, string sipCallId, string fromName, string fromNumber, string to, string data, Guid contextId)

This function may be used to simulate a BlocNotification.

Use it for test purpose of your interface

public Task SetVariable(Guid callContextId, string name, object value)

Set specified KSL variable into the specified callContext. callContextId is received by bloc notifications and is constant throughout the call.

Setting a variable will set it for the stack of all blocs executed during the call, this means that new bloc will have the variable set and previous bloc (if they return from the stack) will also have the variable set. Take care to not set variables that would affect the normal behaviour of the blocs and respect the bloc documentation as described below

public async Task<object?> GetVariable(Guid callContextId, string name)

Read the specified variable from the call context stack. Only the latest value in the stack will be returned, therefore the returned value may depend on the exact time the GetVariable is executed

# Thello functional Blocs compatibles with SignalR API

## Queue bloc

A screenshot of a computer screen

Description automatically generatedNotifications sent by “Queues” blocs

* QUEUE\_ENTER
* QUEUE\_FIRST
* QUEUE\_ANSWERED
* QUEUE\_FALLBACK
* QUEUE\_HANGUP

Variables that may affect the Queue flow:

If you want to control which operator will receive queued calls, you may configure the queue as follow:

* Do not assign operators
* Clear the checkbox “fallback if no operator”

Monitor the queue notifications and save the callContextId

If you decide to send a queued call to an operator, set the variable PICKUP\_OPERATOR to the operator’s UID (pass the string guid), the specified operator will be added to the list of assigned and available operators (if this list is empty, it will be the only one). It will be ringed and then bridged with the pending call

Example:

SetVariable(callContextId, “PICKUP\_OPERATOR”, “f17618f7-56b1-4a2f-8e7f-ea97f0a995e0”)