# TAK Server WebCOP API Definition

## Overview

This document covers the API functionality needed to implement ATAK-like functionality via a Web Browser. There are multiple parts to this API, a set of REST APIs that browsers invoke directly, and then a series of messages that can be used in a WebSocket channel.

## Establishing a WebSocket / STOMP Message feed

The first step to getting the message feed is establishing a websocket and getting a topic for your user. Once the feed has been established, TAK Server will send relevant SA, Chat, Mission Change and Control messages to the browser client, using the WebSocket channel.

The current implementation infers the user from the client certificate presented, so no embedded additional authentication is needed. When requesting a topic, the websocket client may also specify callsign, team, role and TAK version. Once a topic has been assigned by the server, the topic will be tracked as a subscription in the TAK Server contact list, contact API and subscription list.

The WebSocket connection and topic must be periodically refreshed as directed by REFRESH\_TOPIC control messages send from the server to the client.

To request topic creation for your user, invoke this API:

### Get SA Topic

curl 'https://[server]:8443/Marti/api/cop/topic?callsign=bob&team=orange&role=HQ&takv=WebCOP:0.1'

Success Response Code: 200 OK

{

"version":"1.0.0",

"type":"Topic",

"data":"13876e57d2f24273aaeb2482c12eb5ca"

}

*optional parameters: callsign, team, role, takv, clientSeed*

The *clientSeed*parameter allows the browser client to specify a unique random value, which will be incorporated into the topic. If this parameter is provided, then multiple sessions for a single user credential (client cert), for example, on different browsers and computers, will be assigned a different topic (uid). If the parameter not provided, then sessions using the same credential on multiple browsers / computers will share the same topic. Whether or not the *clientSeed* is provided, the topic will remain static.

The ‘data’ from the response to the ‘topic’ call is the topic id for your user, which is needed for the next step after the WebSocket is initialized. STOMP is useful for the WebSocket handling. STOMP uses text-based frames.

Frame Object

|  |  |  |
| --- | --- | --- |
| Property | Type | Notes |
| command | String | STOMP frame type ("CONNECT", "SEND", etc.) |
| headers | JavaScript object |  |
| body | String |  |

//Sample code (from WebContent/cop/messaging.js)

var socket = new SockJS('/Marti/api/cop');

stompClient = Stomp.over(socket);

getTopicAndSubscribe();

…

var getTopicAndSubscribe = function() {

$.get("/Marti/api/cop/topic", function(data) {

var topic = data.data;

stompClient.connect({}, function(frame) {

stompClient.subscribe('/topic/' + topic, processMessage);

});

});

}

‘topic’ here is the value of the ‘data’ from the GET call above, and processMessage is a callback for handing data that comes from the server.

The above sample code sends a STOMP frame over the websocket whose raw text looks like this:

["SUBSCRIBE\nid:sub-0\ndestination:/topic/13876e57d2f24273aaeb2482c12eb5ca\n\n\u0000"]

There are some control messages used to periodically refresh the topic by tearing down the websocket and recreating it. See the processMessage function in WebContent/cop/messaging.js, specifically the ‘ControlMessage’ case statement.

When CoT events are published to the backend router, if they match the group-filtering of the current web user they will end up being formatted to come over the WebSocket as STOMP frames. An example in raw text looks like this:

["MESSAGE\ndestination:/topic/13876e57d2f24273aaeb2482c12eb5ca\ncontent-type:application/json;charset=UTF-8\nsubscription:sub-0\nmessage-id:k7xzqh80-249482\ncontent-length:279\n\n{\"messageType\":\"SituationAwarenessMessage\",\"uid\":\"142db25e-1be5-36cb-8b5b-dbd1183f2bb2\",\"type\":\"a-f-G-U-C\",\"lat\":42.3803274,\"lon\":-71.13891009999999,\"callsign\":\"admin\",\"hae\":0.0,\"start\":1497366241479,\"time\":1497366241479,\"stale\":1497452641479,\"group\":\"Cyan\",\"role\":\"Team Member\"}\u0000"]

Broken out, you can see the STOMP command is “Message”, it came from the ‘topic’ we’d subscribed to earlier, and the body is a JSON object that has the CoT fields broken out in a straightforward way.

The full “SituationAwarenessMessage” type has these fields you can expect:

|  |  |
| --- | --- |
| Field | Data type |
| messageType | String: “SituationAwarenessMessage” |
| uid | String |
| type | String (CoT formatted) |
| lat | Double |
| lon | Double |
| hae | Double |
| start | CoT Timestamp |
| time | CoT Timestamp |
| stale | CoT Timestamp |
| callsign | String |
| color | String (one of ATAK’s 16 colors) |
| role | String ("Team Member”, “Team Lead”, “HQ”) |
| detail | JSON subtree (see below) |
| iconsetPath | String (e.g."34ae1613-9645-4222-a9d2-e5f243dea2865/Buildings/office.png" |
|  |  |

The correct display icon to use should follow ATAK’s conventions using GET api/iconurl :

* if ‘color’ is present, use this API with the value of the color property:
  + GET https://<server>:8443/Marti/api/iconurl?cotType=a-f-G-U-C-I&groupName=Purple&
* if ‘iconsetPath’ is present, then attempt to resolve the icon:
  + GET https://<server>:8443/Marti/api/iconurl?cotType=a-f-G-U-C-I&iconsetpath=34ae1613-9645-4222-a9d2-e5f243dea2865/Buildings/office.png&
* else, fall back to CoT type:
  + GET https://<server>:8443/Marti/api/iconurl?cotType=a-f-G-U-C-I&

If the iconurl function cannot find the specific icon being asked for by color or iconset, then it will return the CoT 2525B icon. So the ‘cotType’ parameter is required, others (such as groupName, or iconsetpath) are optional.

**Detail property**

The detail property contains a JSON representation of the detail element of the original CoT message. This property contains a wide variety of information not included in other properties. The specific contents of this property will vary based on the message type. This value of this property is itself JSON.

The detail property can be used to send / receive shapes and also points / map markers.

The following are examples of the JSON detail property, for several message types:

*Shapes: shapes are a one-to-one translation of CoT shapes. See CoT-Shape.xsd in the CoT information package for additional information.*

*Shape: circle*

{"precisionlocation":{"altsrc":"???","geopointsrc":"Calc"},"shape":{"link":{"uid":"7020dbf0-c347-4e31-8a25-7ab5baefbfe7.Style","Style":{"LineStyle":{"color":"ffffffff","width":2},"PolyStyle":{"color":"00ffffff"}},"type":"b-x-KmlStyle","relation":"p-c"},"ellipse":{"major":182752.91852647165,"minor":182752.91852647165,"angle":360}},"color":{"argb":-1},"\_flow-tags\_":{"TAKServer054c1aa7":"2018-07-26T18:28:40Z"},"contact":{"callsign":"Drawing Circle 1"}}

*Shape: concentric circles*

{"precisionlocation":{"altsrc":"???","geopointsrc":"Calc"},"shape":{"link":{"uid":"4b4f1620-5073-4303-a6da-dadc3611fda5.Style","Style":{"LineStyle":{"color":"ffffffff","width":2},"PolyStyle":{"color":"00ffffff"}},"type":"b-x-KmlStyle","relation":"p-c"},"ellipse":[{"major":6.783474303271526,"minor":6.783474303271526,"angle":360},{"major":13.566948606543052,"minor":13.566948606543052,"angle":360},{"major":20.35042290981458,"minor":20.35042290981458,"angle":360},{"major":27.133897213086104,"minor":27.133897213086104,"angle":360},{"major":33.91737151635763,"minor":33.91737151635763,"angle":360},{"major":40.70084581962916,"minor":40.70084581962916,"angle":360},{"major":47.484320122900684,"minor":47.484320122900684,"angle":360},{"major":54.26779442617221,"minor":54.26779442617221,"angle":360}]},"color":{"argb":-1},"\_flow-tags\_":{"TAKServer6bc41eef":"2018-08-08T18:04:57Z"},"contact":{"callsign":"Drawing Circle 4"}}

*Shape: telestration*

{"precisionlocation":{"altsrc":"???","geopointsrc":"???"},"\_flow-tags\_":{"TAKServer054c1aa7":"2018-07-26T18:31:05Z"},"contact":"","link":[{"detail":{"fillColor":{"value":16711680},"precisionlocation":{"altsrc":"???","geopointsrc":"???"},"contact":{"callsign":"Freehand 3"},"link":[{"point":"34.15199190157241,-89.81985480163021"},{"point":"33.715945619140086,-87.5875794561014"},{"point":"32.87739507600101,-87.22108648892502"},{"point":"26.135448709162823,-86.6213707244546"},{"point":"25.632318383279376,-86.02165495998418"},{"point":"25.93419657880944,-84.98881114339622"},{"point":"28.952978534110123,-79.35814646586832"},{"point":"29.154230664463505,-78.4252552766921"},{"point":"24.76022581841473,-79.3914640083389"},{"point":"22.64707844970426,-79.35814646586832"},{"point":"22.110406102095247,-79.09160612610368"},{"point":"21.674359819662925,-78.15871493692747"},{"point":"21.640817797937366,-74.96023085975185"},{"point":"22.378742275899754,-72.72795551422304"},{"point":"22.44582631935088,-72.39478008951724"},{"point":"22.31165823244863,-72.22819237716435"},{"point":"21.171229493779478,-72.49473271692898"},{"point":"20.164968842012584,-72.9611783115171"},{"point":"18.185989560204362,-74.39383263775201"},{"point":"17.213270930163034,-75.8598045064575"},{"point":"17.079102843260785,-76.45952027092792"},{"point":"17.280354973614152,-77.95880968210399"},{"point":"17.917653386399863,-79.82459206045642"},{"point":"18.957456059892316,-84.05591995421999"},{"point":"19.594754472678012,-85.85506724763127"},{"point":"20.3326789506404,-87.35435665880735"},{"point":"21.13768747205392,-88.28724784798356"},{"point":"22.68062047142982,-89.12018640974804"},{"point":"24.793767840140298,-89.78653725915963"},{"point":"26.73920510022296,-90.21966531127715"},{"point":"28.18151203442217,-90.28630039621832"},{"point":"29.389024816542445,-90.153030226336"},{"point":"31.133209946271727,-89.61994954680674"},{"point":"32.87739507600101,-88.52047064527761"}],"strokeWeight":{"value":6},"strokeColor":{"value":-65536}}},{"detail":{"fillColor":{"value":16711680},"precisionlocation":{"altsrc":"???","geopointsrc":"???"},"contact":{"callsign":"Freehand 3"},"link":[{"point":"34.15199190157241,-89.81985480163021"},{"point":"33.715945619140086,-87.5875794561014"},{"point":"32.87739507600101,-87.22108648892502"},{"point":"26.135448709162823,-86.6213707244546"},{"point":"25.632318383279376,-86.02165495998418"},{"point":"25.93419657880944,-84.98881114339622"},{"point":"28.952978534110123,-79.35814646586832"},{"point":"29.154230664463505,-78.4252552766921"},{"point":"24.76022581841473,-79.3914640083389"},{"point":"22.64707844970426,-79.35814646586832"},{"point":"22.110406102095247,-79.09160612610368"},{"point":"21.674359819662925,-78.15871493692747"},{"point":"21.640817797937366,-74.96023085975185"},{"point":"22.378742275899754,-72.72795551422304"},{"point":"22.44582631935088,-72.39478008951724"},{"point":"22.31165823244863,-72.22819237716435"},{"point":"21.171229493779478,-72.49473271692898"},{"point":"20.164968842012584,-72.9611783115171"},{"point":"18.185989560204362,-74.39383263775201"},{"point":"17.213270930163034,-75.8598045064575"},{"point":"17.079102843260785,-76.45952027092792"},{"point":"17.280354973614152,-77.95880968210399"},{"point":"17.917653386399863,-79.82459206045642"},{"point":"18.957456059892316,-84.05591995421999"},{"point":"19.594754472678012,-85.85506724763127"},{"point":"20.3326789506404,-87.35435665880735"},{"point":"21.13768747205392,-88.28724784798356"},{"point":"22.68062047142982,-89.12018640974804"},{"point":"24.793767840140298,-89.78653725915963"},{"point":"26.73920510022296,-90.21966531127715"},{"point":"28.18151203442217,-90.28630039621832"},{"point":"29.389024816542445,-90.153030226336"},{"point":"31.133209946271727,-89.61994954680674"},{"point":"32.87739507600101,-88.52047064527761"}],"strokeWeight":{"value":6},"strokeColor":{"value":-65536}}}]

*Map Marker (point) with Attachment*

Points with attachments contain a “fileshare” tag with details about the attached file – which is available from the sync api.

{"precisionlocation":{"altsrc":"GPS","geopointsrc":"GPS"},"ackrequest":{"uid":"f911502c-8da6-447b-8ecd-4ccfc896a619","endpoint":"10.187.124.61:4242:tcp","ackrequested":true,"tag":"20180817\_165502.jpg"},"\_flow-tags\_":{"TAKServer2d6a88bb":"2018-08-17T20:57:39Z"},"fileshare":{"senderUid":"ANDROID-358982072593830","senderUrl":"https://192.168.43.68:8443/Marti/sync/content?hash=1264f50b0a2e4646181b73940a1a4d279913f59486fc79ca70ded477151f4cda","filename":"20180817\_165502.jpg.zip","sizeInBytes":1056484,"sha256":"1264f50b0a2e4646181b73940a1a4d279913f59486fc79ca70ded477151f4cda","name":"20180817\_165502.jpg","senderCallsign":"PSS5"}}

### Sending your own SA

Once the STOMP WebSocket is initialized, the browser may send his own messages though the websocket. Example Javascript code for this:

geolocation.on('change:position', function() {

console.debug("Geolocation position!");

var coordinates = geolocation.getPosition();

if(coordinates) {

positionFeature.setGeometry(coordinates ? new ol.geom.Point(coordinates) : null);

var wgsCoord = ol.proj.transform(coordinates, 'EPSG:900913', 'EPSG:4326');

if(myuid != undefined) {

stompClient.send("/cop/cop", {}, JSON.stringify({

'lat' : wgsCoord[1],

'lon' : wgsCoord[0],

'type' : 'a-f-G-U-C',

'uid' : myuid,

'callsign' : mycallsign

}));

}

}

});

This code results in a message sent over the websocket that looks like this:

["SEND\ndestination:/cop/cop\ncontent-length:126\n\n{\"lat\":42.3803274,\"lon\":-71.13891009999999,\"type\":\"a-f-G-U-C\",\"uid\":\"142db25e-1be5-36cb-8b5b-dbd1183f2bb2\",\"callsign\":\"admin\"}\u0000"]

Note that for a self marker, a consistent UID should be used. Currently only a small selection of CoT fields are supported, namely uid, callsign, type, lat, and lon.

This same API can be used for the point dropper tool.

## Chat API

### Chat Message Format

|  |  |  |
| --- | --- | --- |
| Field | Data type | Description |
| messageType | String | “ChatMessage” |
| from | String | Client UID of message sender |
| addresses | String[] | Set of destination addresses, specified in the format:  <type>:<identifier>  valid types include: uid || mission || group || special  The identifier is a string specifying the uid, mission or group. For the *special* type, the only supported identifier that is currently supported is **AllStreaming**. In future API updates, additional *special* identifiers may be supported. |
| lat | Number | latitude |
| lon | Number | longitude |
| hae | Number | altitude |
| body | String | Chat message body |
| timestamp | Number | Timestamp when the message was sent |

As with Situation Awareness messages, chat messages are sent and received over the bi-directional WebSocket channel using the STOMP protocol. The same WebSocket channel may be used for both Situation Awareness messages and Chat messages, establishing a connection and obtaining a topic as described above. When a message is sent from TAK Server to the browser, the STOMP frame type is MESSAGE. When a message is sent from the browser to TAK Server, the STOMP frame type is SEND.

Based on the identity of the user accessing the WebCOP API, TAK Server internally tracks group membership. Messages are sent and received to ATAK and WebCop clients based on common group membership. Additional filtering is in addition to the group filtering, further restricting the set of clients who will receive a message.

When sending a message from the WebCOP to TAK Server, messages may be addressed to any combination of ATAK client UIDs, missions, or TAK Server groups, or websocket topics. Websocket topics and ATAK uids may be used interchangeably. A websocket message may be sent to an ATAK client by addressing the message to the uid of the ATAK device. An ATAK device may send a message to a websocket user by addressing the message to the uid (topic) of the websocket client. Messages may also be sent between websocket clients in the same manner. See Mission API documentation, and Contacts and Groups API, below. The Mission API may be used to subscribe WebCOP topics to a mission, and to obtain mission information as REST resources.

Missions may be utilized in the same manner as chatrooms in other chat protocols, such as XMPP.

### Mission Change Notification Message Format

|  |  |
| --- | --- |
| Field | Data type |
| messageType | String: “MissionChange” |
| missionName | String |
| cotType | String  Change to mission log: "t-x-m-c-l"  Other mission change: "t-x-m-c" |
| time | CoT Timestamp |

Once a topic has been subscribed to a mission, it will receive notifications of changes to the mission.

Change notifications will occur for the following events

* [Create Mission](https://dsl-external.bbn.com/tracsvr/marti/wiki/datasync_api#create_mission)
* [Delete Mission](https://dsl-external.bbn.com/tracsvr/marti/wiki/datasync_api#delete_mission)
* [Add Content](https://dsl-external.bbn.com/tracsvr/marti/wiki/datasync_api#add_content) to Mission (by hash or uid)
* Create, Update or Delete Mission Log Entry. For mission log changes, the cotType will be "t-x-m-c-l"

**Contacts and Groups API**

**Resource: contacts**

The Contacts API provides information about currently connected TAK Server clients, represented as REST resources. Clients UIDs and callsigns are provided by the API, and may be sorted according to query parameters.

api/contacts

**GET all contacts**

curl "http://<username>:<password>@localhost:8080/Marti/api/contacts/all"

curl "http://<username>:<password>@localhost:8080/Marti/api/contacts/all ?sortBy=UID&direction=DESCENDING"

Success Response Code: 200 OK

[

{

"notes": "Anonymous\_329994173",

"callsign": "anon1",

"uid": "anon1"

},

{

"notes": "",

"callsign": "Anonymous\_COTRMI",

"uid": "Anonymous\_COTRMI"

}

]

**Query Parameters**

*optional*

sortBy : CALLSIGN or UID

*sort the result list by callsign, or client UID*

direction : ASCENDING or DESCENDING

*specify the ordering of the result list*

**Resource: groups**

The Groups API provides information about groups that have been assigned to an actual TAK Server user, or specified in the TAK Server configuration. These groups are persisted in the TAK Server database.

api/groups

**GET all groups**

curl "http://<username>:<password>@localhost:8080/Marti/api/groups/all"

Success Response Code: 200 OK

{

"version": "1.0.0",

"type": "com.bbn.marti.remote.groups.Group",

"data": [

{

"name": "CN=TAK2,OU=Groups,DC=cam,DC=bbn,DC=com",

"direction": "OUT",

"created": "2017-04-01",

"type": "SYSTEM",

"bitpos": 9

},

{

"name": "GroupC",

"direction": "OUT",

"created": "2017-01-24",

"type": "SYSTEM",

"bitpos": 7

},

{

"name": "\_\_ANON\_\_",

"direction": "OUT",

"created": "2016-09-27",

"type": "SYSTEM",

"bitpos": 0

},

{

"name": "blue",

"direction": "OUT",

"created": "2016-09-27",

"type": "SYSTEM",

"bitpos": 4

},

{

"name": "c",

"direction": "OUT",

"created": "2017-04-01",

"type": "SYSTEM",

"bitpos": 12

},

{

"name": "green",

"direction": "OUT",

"created": "2016-09-27",

"type": "SYSTEM",

"bitpos": 3

},

{

"name": "yellow",

"direction": "OUT",

"created": "2016-09-27",

"type": "SYSTEM",

"bitpos": 5

},

{

"name": "zoo",

"direction": "OUT",

"created": "2017-04-01",

"type": "SYSTEM",

"bitpos": 14

}

]

}

**Query Parameters**

*none*

**Properties API**

The TAK Server properties API can be used as a server-side in-memory repository of user settings and preferences. Settings and preferences can be stored and retrieved on a “per-UID” basis. An arbitrary number of preference keys can be stored per UID, and each key can be associated with one or more values. Deletion methods are provided to clear all data for a UID with a single call, or clear all of the values associated with a single key for a given UID.

**Resource: properties**

api/properties

**PUT a key-value pair for a UID**

PUT https://gov.atakserver.com:8443/Marti/api/properties/alice

Request Body:

A single JSON object associating a key with a value. This call associated the value “HQ” with the key “chatroom”. If another PUT is executed for the same UID and key, but with a different value, that value is added to the map for that key, as an additional value.

{"chatroom": "HQ"}

Success Response Code: 200 OK

{"version":"2","type":"KeyValuePair","data":{"chatroom":"HQ"}}

**GET all key-value maps for a UID**

GET https://gov.atakserver.com:8443/Marti/api/properties/alice/all

Success Response Code: 200 OK

{

"version": "2",

"type": "Properties",

"data": {

"chatroom": [

"HQ"

],

"team": [

"Blue",

"Green"

]

}

}

This result shows multiple key-values can be stored for a UID (alice), and multiple values can be stored for a key (team).

If no values are stored for this UID, the result is 404 Not Found, with a JSON error body.

**GET all values for a key, per UID**

GET https://gov.atakserver.com:8443/Marti/api/properties/alice/team

Success Response Code: 200 OK

{

"version": "2",

"type": "Properties",

"data": [

"Blue",

"Green"

]

}

If no values are stored for this UID and key, the result is 404 Not Found, with a JSON error body.

**DELETE all values for a key, per UID**

DELETE https://gov.atakserver.com:8443/Marti/api/properties/alice/team

The key “team” and all values with which it is associated are removed, for the UID “alice”

Success Response Code: 200 OK

This result shows multiple key-values can be stored for a UID (alice), and multiple values can be stored for a key (team).

**DELETE all data for a UID**

DELETE https://gov.atakserver.com:8443/Marti/api/properties/alice/all

All keys are values that are stored for the UID “alice” are deleted.

Success Response Code: 200 OK

This result shows multiple key-values can be stored for a UID (alice), and multiple values can be stored for a key (team).

**Utility API**

**Resource: user roles**

api/ util/user/roles

**GET all roles for current user**

GET https://gov.atakserver.com:8443/Marti/api/util/user/roles

Success Response Code: 200 OK

["ROLE\_ANONYMOUS", "ROLE\_READONLY"]

The response will show any roles assigned to the user by TAK Server.

**Video Feed API**

**POST create or edit one or more video feeds**

POST https://<host>:8443/Marti/vcm

**URL parameter:**

**action = add | setActive**

default *action* is add. The setActive action will set the feed(s) to active status, if they are inactive.

**Request Body: XML VideoConnection object, containing one or more Feed objects**

Content-Type: application/xml

If the feed object already exists, it will be updated based on the contents of the POSTed object.

**Example**

<?xml version="1.0" encoding="UTF-8"?>

<videoConnections>

<feed>

<active>false</active>

<address>rtmp://1.2.3.4/rtplive/test.stream</address>

<alias>Test Stream 5</alias>

<buffer>-1</buffer>

<ignoreEmbeddedKLV>false</ignoreEmbeddedKLV>

<timeout>2000</timeout>

<path />

<port>-1</port>

<protocol>raw</protocol>

<roverPort>-1</roverPort>

<rtspReliable>0</rtspReliable>

<type>VIDEO</type>

</feed>

</videoConnections>

**setActive**

**GET All video connections**

GET https://<host>:8443/Marti/vcm

Success Response Code: 200 OK

**Response Body**

<?xml version="1.0" encoding="UTF-8"?>

<videoConnections>

<feed>

<active>true</active>

<address>rtmp://8.15.251.50/rtplive/vabeachcam036.stream</address>

<alias>5th and Atlantic</alias>

<buffer>-1</buffer>

<id>2</id>

<ignoreEmbeddedKLV>false</ignoreEmbeddedKLV>

<preferredMacAddress>null</preferredMacAddress>

<timeout>5000</timeout>

<path />

<port>-1</port>

<protocol>raw</protocol>

<roverPort>-1</roverPort>

<rtspReliable>0</rtspReliable>

<type>VIDEO</type>

<uid>a1cca03d-2fd2-49a5-98c4-e518942e3c82</uid>

</feed>

<feed> … </feed>

</videoConnections>

**DELETE video connection**

DELETE https://<host>:8443/Marti/vcm?id=<id>

Success Response Code: 200 OK

**URL parameter:**

**id=<id>**

delete the feed matching the provided *id*. (see GET video connection)