# Proposal for FHIRCast Websockets Prototype

## Summary

This document proposes the specifications to be followed by those parties wishing to participate in a collaborative FHIRCast/WebSocket Integration prototype. Participants can implement either/or a FHIRCast client and FHIRCast Hub. This specification also includes a proposed implement of authentication and “topic generation” which is not something currently included in the FHIRCast specification.

The proposed specification will adhere to the [current FHIRCast specifications](http://fhircast.org/), and the [proposed Websocket specifications](https://github.com/fhircast/docs/pull/57/commits/2cc6c907cdeb7e69369bd58f8af9b00a896f9633?short_path=1a523bd#diff-1a523bd9fa0dbf998008b37579210e12) in all cases, unless otherwise specified herein. All communication from client to server will conform to WebSub and FHIRCast RESTful API specifications. Only Hub notifications to the client will be over Websockets.

## FHIRCast Hub

The Hub will provide APIs that allow FHIRCast clients that share context with it to subscribe to imaging study events, and to notify other clients (publish) the same imaging study events. Per the FHIRCast specification, these events are named, but we will support a limited set at this time:

* open-imaging-study
* switch-imaging-study
* close-imaging-study
* user-logout

## FHIRCast Client

The client can be any application that will authenticate, subscribe to a topic, send context changes, and receive context changes via Websockets using the “FHIRCast Hub” as described in this proposal.

## Required Hub Implementation

### Authentication

The hub must provide a topic to each client, which is based on an identity (radiologist). This requires some form of persistent storage (configuration file/database) in the hub. It will enable the authentication process to identify the application user based on their credentials and return them a topic (essentially the session id).

From a customer implementation perspective, configuring and maintaining passwords at the user level would be tedious and difficult. In this prototype we will use an application-wide secret. However, we still need to configure users and their grouping. For example, if I am "gkustas61" on a PACS but "georgek" on the RIS, someone needs to configure that. When authentication occurs, the association is made so that they both receive the same topic.

***Security Consideration****: The Hub (or authorization server) MUST use the best available randomness methods to generate the secrets and topics. GUIDs are acceptable.*

Example data model and sample data:

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

### GetTopic REST Method

The hub SHALL implement a REST method using the base URL and “GetTopic” as the endpoint to authenticate and authorize the client, and return to it a topic (session id). Example:

GET https://hub.example.com/gettopic?username=joe&secret=61B584A8-C5AD-4A87-A40F-19E448EEBBAD

In this example, based on the sample data from above, the topic “1A3DF21C-1451-4DC5-8B59-3F824D3A7ED7” would be returned along with a status code of 200 (or any other valid 200s code).

If I launch another client, I can authenticate using:

GET https://hub.example.com/authenticate?username=joe2&secret=61B584A8-C5AD-4A87-A40F-19E448EEBBAD

And receive the same topic as I did for user “joe”, who is connecting from a different client application.

### Subscribe REST Method

The hub SHALL implement a REST method using the base URL as the endpoint. The specification conforms to WebSub specifications. The client will use the topic returned during authentication. Example:

POST https://hub.example.com

Authorization: Bearer 61B584A8-C5AD-4A87-A40F-19E448EEBBAD

Content-Type: application/x-www-form-urlencoded

hub.topic=1A3DF21C-1451-4DC5-8B59-3F824D3A7ED7

&hub.events=open-imaging-study,close-imaging-study,switch-imaging-study,user-logout

&hub.mode=subscribe

&hub.channel.type=websocket

Notes:

1. the example shows a hub channel type of “websocket”, but “websub” is also valid and the hub SHOULD support the traditional WebSub REST interface, although not required for this prototype.
2. The application-wide secret is sent in the header as the authorization, and the topic is in the body (data) of the message

### WebSocket Raw JSON Format

The hub will send notifications to Websocket clients in raw json format, conforming to FHIR specifications, with one exception:

Two encompassing objects will be used:

1. Header: will contain any valid HTTP header
2. Body: will contain the standard FHIR resource
3. The client app SHALL respond to all Websocket notifications from the hub. The response includes timestamp, status and status code value as shown in the example below. TODO: should the message follow the header/body format?
4. The response message from the hub will contain one object containing the status, status – no header or body differentiation (see example below)

Example Notification (Hub to client):

TODO: update example user R4 FHIR spec changes for “accession”

“header”:

{

“Authorization”: “Bearer 61B584A8-C5AD-4A87-A40F-19E448EEBBAD”,

},

“body”:

{

  "timestamp": "2018-01-08T01:40:05.14",

  "id": "wYXStHqxFQyHFELh",

  "event": {

    "hub.topic": "1A3DF21C-1451-4DC5-8B59-3F824D3A7ED7",

    "hub.event": "close-imaging-study",

    "context":

[

{

      "key": "patient",

      "resource": {

        "resourceType": "Patient",

        "id": "ewUbXT9RWEbSj5wPEdgRaBw3",

        "identifier": [

          {

            "system": "urn:MRN",

            "value": "2667"

          }

        ]

      }

    },

    {

      "key": "study",

      "resource": {

        "resourceType": "ImagingStudy",

        "id": "8i7tbu6fby5ftfbku6fniuf",

        "uid": "urn:oid:2.16.124.113543.6003.1154777499.30246.19789.3503430045",

        "identifier": [

          {

            "system": "urn:accession",

            "value": "185444"

          }

        ],

        "patient": {

          "reference": "Patient/ewUbXT9RWEbSj5wPEdgRaBw3"

        }

      }

    }

  }

]

Example Hub response:

{

"timestamp": "2018-01-08T01:40:05.14",

"status": "OK",

"statusCode": "200",

}

### Establishing the WebSocket Connection

The FHIRCast WebSocket client will attempt to connect to the hub after:

1. Obtaining a topic during the authentication/authorization process (GetTopic)
2. Calling the Subscribe REST method.

The hub will listen for the connection on the base URL, using an endpoint containing the topic. Example (client to hub):

wss:*//hub.example.com/1A3DF21C-1451-4DC5-8B59-3F824D3A7ED7*

Example Response (hub to client):

{

"timestamp": "2018-01-08T01:40:05.14",

"status": "OK",

"statusCode": "200",

}

### Get Current Context

The hub SHALL implement a REST method using the base URL and path to the topic as the endpoint. The specification conforms to FHIR specifications. Client can at any time (and SHOULD upon connection to the hub) Example:

GET https://hub.example.com/1A3DF21C-1451-4DC5-8B59-3F824D3A7ED7

Authorization: Bearer 61B584A8-C5AD-4A87-A40F-19E448EEBBAD

Potential response (context is a key/value array of FHIR Resources):

{

"timestamp":"2018-01-08T01:40:05.14",

"id":"wYXStHqxFQyHFELh",

"event":{

"hub.topic":"*1A3DF21C-1451-4DC5-8B59-3F824D3A7ED7*",

"context":[

]

}

}