# Yamcs Server API

YAMCS-SA-API-0.28.0

October, 14th
2015

www.yamcs.org

# Yamcs Server API

YAMCS-SA-API-0.28.0

This version was published October, 14th 2015
A later version of this document may be available at www.yamcs.org

# Table of Contents

# Chapter 1. Web API Configuration

In `yamcs.yaml` define the exposed port. Traffic on this port is multiplexed between web tools such as the REST API, the WebSocket API and the web displays.

```
webPort: 80902
```

In `yamcs.(instance).yaml` activate the `YamcsWebService`.

```
services:
      - [...]
      - org.yamcs.web.YamcsWebService
```

# Chapter 2. REST API

Yamcs provides a RESTful web service allowing remote tools to interface with a growing set of internals.

All operations accept and deliver both JSON and Protobuf. The `.proto` files that define the contract of the REST endpoint are included in the yamcs source code. Using the `protoc` compiler, client code can be generated for Java, Python, C++ and more.

All operations documented in the following pages assume the default *simulator* instance. They also assume that the client is sending and asking JSON messages. However, the server is also capable of reading and writing binary data using Google Protocol Buffers. If -as a client- you are sending in the Protobuf wire format, change the HTTP `Content-Type` header to `application/octet-stream`. If you also want the server to respond with Protobuf messages, set the HTTP `Accept` header to `application/octet-stream`.

## Error Handling

When an exception is caught while handling a REST request, the server will try to give some feedback to the client by wrapping it in a generic exception message like this:

```
{
  "exception" : {
    "type": "<short>",
    "msg": "<long>"
  }
}
```

Clients of the REST API should check on whether the status code is `200 OK`, and if not, deserialize the response as a `RestExceptionMessage` instead.

Note that the exception handling still requires more thoughtwork in terms of when we return an applicative exception as part of a 200 response, and when we return an actual exception.

## 2.1. Alarms

### List Active Alarms

Returns a list of current active alarms. For each alarm you get full information on the value occurrence that initially triggered the alarm, the most severe value since it originally triggered, and the latest value at the time of your request.

```
curl -XGET http://localhost:8090/simulator/api/alarms?pretty
```

```
{
  "alarms" : [ {
```

```
    "id" : 3,
    "triggerValue" : {
      "id" : {
        "name" : "/YSS/SIMULATOR/O2TankTemp"
      },
      "rawValue" : {
        "type" : 2,
        "uint32Value" : 227
      },
      "engValue" : {
        "type" : 2,
        "uint32Value" : 227
      },
      "acquisitionTime" : 1440576556724,
      "generationTime" : 1440576539714,
      "acquisitionStatus" : 0,
      "processingStatus" : true,
      "monitoringResult" : 21,
      "acquisitionTimeUTC" : "2015-08-26T08:08:40.724",
      "generationTimeUTC" : "2015-08-26T08:08:23.714",
      "watchLow" : 10.0,
      "watchHigh" : 12.0,
      "warningLow" : 30.0,
      "warningHigh" : 32.0,
      "distressLow" : 40.0,
      "distressHigh" : 42.0,
      "criticalLow" : 60.0,
      "criticalHigh" : 62.0,
      "severeLow" : 80.0,
      "severeHigh" : 82.0,
      "expirationTime" : 1440576558224,
      "expirationTimeUTC" : "2015-08-26T08:08:42.224"
    },
    "mostSevereValue" : {
      "id" : {
        "name" : "/YSS/SIMULATOR/O2TankTemp"
      },
      "rawValue" : {
        "type" : 2,
        "uint32Value" : 227
      },
      "engValue" : {
        "type" : 2,
        "uint32Value" : 227
      },
      "acquisitionTime" : 1440576556724,
      "generationTime" : 1440576539714,
      "acquisitionStatus" : 0,
      "processingStatus" : true,
      "monitoringResult" : 21,
```

```
      "acquisitionTimeUTC" : "2015-08-26T08:08:40.724",
      "generationTimeUTC" : "2015-08-26T08:08:23.714",
      "watchLow" : 10.0,
      "watchHigh" : 12.0,
      "warningLow" : 30.0,
      "warningHigh" : 32.0,
      "distressLow" : 40.0,
      "distressHigh" : 42.0,
      "criticalLow" : 60.0,
      "criticalHigh" : 62.0,
      "severeLow" : 80.0,
      "severeHigh" : 82.0,
      "expirationTime" : 1440576558224,
      "expirationTimeUTC" : "2015-08-26T08:08:42.224"
    },
    "currentValue" : {
      "id" : {
        "name" : "/YSS/SIMULATOR/O2TankTemp"
      },
      "rawValue" : {
        "type" : 2,
        "uint32Value" : 223
      },
      "engValue" : {
        "type" : 2,
        "uint32Value" : 223
      },
      "acquisitionTime" : 1440577186414,
      "generationTime" : 1440577169410,
      "acquisitionStatus" : 0,
      "processingStatus" : true,
      "monitoringResult" : 21,
      "acquisitionTimeUTC" : "2015-08-26T08:19:10.414",
      "generationTimeUTC" : "2015-08-26T08:18:53.410",
      "watchLow" : 10.0,
      "watchHigh" : 12.0,
      "warningLow" : 30.0,
      "warningHigh" : 32.0,
      "distressLow" : 40.0,
      "distressHigh" : 42.0,
      "criticalLow" : 60.0,
      "criticalHigh" : 62.0,
      "severeLow" : 80.0,
      "severeHigh" : 82.0,
      "expirationTime" : 1440577187914,
      "expirationTimeUTC" : "2015-08-26T08:19:11.914"
    },
    "violations" : 102
  } ]
}
```

When using Protobuf, the response can be deserialized as `Rest.GetAlarmsResponse`.

### Acknowledge an alarm

```
curl -XPOST
http://localhost:8090/simulator/api/alarms/acknowledge/(alarmId)/my/quali
-d '{"message": "bla bla"}'
```

For example to acknowledge the alarm above:

```
curl -XPOST
http://localhost:8090/simulator/api/alarms/acknowledge/3/YSS/SIMULATOR/O2
pretty -d '{"message": "bla bla"}'
```

If you get an empty response, the acknowledge was successful:

```
{}
```

If the alarm could not be acknowledged, you get a message why that is. For instance:

```
{
  "errorMessage" : "Parameter /YSS/SIMULATOR/O2TankTemp is not in state
of alarm"
}
```

When using Protobuf, send a request body of type `Rest.AcknowledgeAlarmRequest`. The response can be deserialized as `Rest.AcknowledgeAlarmResponse`

## 2.2. Archive

### Fetch archive data `api/archive`

Fetches Parameters, Packets, Processed Parameters, Events and/or Command History from the Yamcs Archive. The API of this method is likely to change in the near future.

Filters are specified in the request body, and should always include at least a `start` and a `stop` value. These are to be encoded in the internal Yamcs time format (use `TimeEncoding` from `yamcs-api`). We might make this configurable in the future.

Request the command history:

```
curl -XGET http://localhost:8090/simulator/api/archive -d '{"start":
1425686400,"stop": 1426636800,"commandHistoryRequest": {}}'
```

In practice the Yamcs archive could be very big. Therefore, responses are by default limited to about 1MB. Anything above will generate a server error. If your client is expected to fetch more data than that, you should use the `stream` -option like so:

Request parameters:

```
curl -XGET http://localhost:8090/simulator/api/archive -d '{"start":
1425686400,"stop": 1426636800,"commandHistoryRequest": {}}'
```

Notice in the response that we now receive multiple individually delimited JSON messages, therefore your JSON client does not need to wait for the full response before starting processing. For Protobuf clients, the individual message are prefixed with their byte-size (Protobuf is not a self-delimiting message format).

## 2.3. Commanding

### Validate a command `/commanding/validator`

Empty response means the command was successfully validated.

```
curl -XGET http://localhost:8090/simulator/api/commanding/validator?
pretty -d '
  {
    "commands": [
      {
        "id": {"name": "/SIMULATOR/SIMULATOR/SWITCH_VOLTAGE_ON"},
        "arguments":[{"name":"voltage_num", "value": "5"}]
      }
    ]
  }'
```

```
{
  "exception" : {
    "type" : "BadRequestException",
    "msg" : "Cannot assign value to voltage_num: Value 5 is not in the
range required for the type IntegerDataType name:voltage_num
sizeInBits:32 signed: false, validRange: [1,3], encoding:
IntegerDataEncoding(sizeInBits:8, encoding:unsigned,
defaultCalibrator:null byteOrder:BIG_ENDIAN)"
  }
}
```

### Send a command `/commanding/queue`

Will perform validation first, and if successful will queue the command for further dispatching. As soon as it's queued, you get back an empty JSON object `{}`. Clients are required to specify a unique `sequenceNumber` for every command. The unicity is not currently enforced by the server, since it's basically there to help the client, and not necessarily the server.

```
curl -XPOST http://localhost:8090/simulator/api/commanding/queue?pretty
-d '
  {
    "commands": [
      {
        "id": {"name": "/SIMULATOR/SIMULATOR/SWITCH_VOLTAGE_ON"},
        "arguments":[{"name":"voltage_num", "value": "2"}],
        "sequenceNumber": 1
      }
    ]
  }'
```

## 2.4. Mission Data Base

### List Parameters `/mdb/parameters`

Retrieve qualified parameter names:

```
curl -XGET http://localhost:8090/simulator/api/mdb/parameters?pretty
```

```
{
  "ids" : [ {
    "name": "/YSS/SIMULATOR/ccsds-apid",
  }, {
    "name": "/YSS/SIMULATOR/packet-type",
  } ]
}
```

Retrieve parameter aliases for specific namespaces:

```
curl -XGET http://localhost:8090/simulator/api/mdb/parameters?pretty -d
'{
  "namespaces": ["MDB:OPS Name"]
}'
```

```
{
  "ids" : [ {
    "name": "SIMULATOR_ccsds-apid",
    "namespace": "MDB:OPS Name"
  } ]
}
```

### Get parameter info `/mdb/parameterInfo`

Get information about a parameter:

```
curl -XGET 'http://localhost:8090/simulator/api/mdb/parameterInfo?
pretty&name=SIMULATOR_ccsds-apid&namespace=MDB:OPS Name'
```

The result: **TBW**

Get information about a list of parameters:

```
curl -XGET http://localhost:8090/simulator/api/mdb/parameterInfo?pretty
```

```
-d '{
    "list": [
     {"name": "SIMULATOR_ccsds-apid",
      "namespace": "MDB:OPS Name"}
    ]
}'
```

The result: **TBW**

### Dump the Mission Database `/mdb/dump`

Return the entire dump of the mission database. This is a very bad method which we would like to make more RESTful and explorable in the future, but for now it returns a java-serialized dump of `org.yamcs.xtce.XtceDb` for the specified yamcs instance. You'll need a dependency on yamcs-xtce to interpret this dump.

```
curl -XGET http://localhost:8090/simulator/api/mdb/dump?pretty
```

```
{
  "rawMdb": "<blob encoded as Base64>"
}
```

## 2.5. Parameters

Retrieve multiple parameters `/parameter/_get`

By default, the parameters are returned after they have been freshly updated (i.e. no cache).
The timeout option indicates how long should wait until the call shall return (with partial or no data).
Instead of the timeout, the `fromCache:true` can be used to retrieve parameters from cache (if available).
See [[Parameter Cache]] for description on how to configure YProcessor parameter cache.

```
curl -XGET http://localhost:8090/simulator/api/parameter/_get?pretty -d
'
  {
    "list": [
      {"name":"/YSS/SIMULATOR/Longitude"},
      {"name":"/YSS/SIMULATOR/Latitude"},
      {"name":"/YSS/SIMULATOR/Altitude"}]
     ,"timeout":2000}
  }'
```


```
{
  "parameter" : [ {
    "id" : {
      "name" : "/SIMULATOR/SIMULATOR/Longitude"
    },
    "rawValue" : {
      "type" : 0,
      "floatValue" : -60.55
    },
    "engValue" : {
      "type" : 0,
      "floatValue" : -60.55
    },
    "acquisitionTime" : 1429799497246,
    "generationTime" : 1429799481242,
    "acquisitionStatus" : 0,
    "processingStatus" : true,
    "monitoringResult" : 1,
    "acquisitionTimeUTC" : "2015-04-23T14:31:02.246",
    "generationTimeUTC" : "2015-04-23T14:30:46.242"
  }, {
    "id" : {
      "name" : "/SIMULATOR/SIMULATOR/Latitude"
    },
    "rawValue" : {
      "type" : 0,
      "floatValue" : 53.3
    },
```

```
    "engValue" : {
      "type" : 0,
      "floatValue" : 53.3
    },
    "acquisitionTime" : 1429799497246,
    "generationTime" : 1429799481242,
    "acquisitionStatus" : 0,
    "processingStatus" : true,
    "monitoringResult" : 1,
    "acquisitionTimeUTC" : "2015-04-23T14:31:02.246",
    "generationTimeUTC" : "2015-04-23T14:30:46.242"
  }, {
    "id" : {
      "name" : "/SIMULATOR/SIMULATOR/Altitude"
    },
    "rawValue" : {
      "type" : 0,
      "floatValue" : 39759.0
    },
    "engValue" : {
      "type" : 0,
      "floatValue" : 39759.0
    },
    "acquisitionTime" : 1429799497246,
    "generationTime" : 1429799481242,
    "acquisitionStatus" : 0,
    "processingStatus" : true,
    "monitoringResult" : 1,
    "acquisitionTimeUTC" : "2015-04-23T14:31:02.246",
    "generationTimeUTC" : "2015-04-23T14:30:46.242"
  } ]
}
```

Retrieve single parameter `/parameter/parameter_fqname`

```
curl -XGET
http://localhost:8090/simulator/api/parameter/YSS/SIMULATOR/BatteryVoltage
pretty
```

```
{
  "id" : {
    "name" : "/YSS/SIMULATOR/BatteryVoltage2"
  },
  "rawValue" : {
    "type" : 2,
    "uint32Value" : 244
  },
```

```
  "engValue" : {
    "type" : 2,
    "uint32Value" : 244
  },
  "acquisitionTime" : 1429801967511,
  "generationTime" : 1429801951507,
  "acquisitionStatus" : 0,
  "processingStatus" : true,
  "monitoringResult" : 1,
  "acquisitionTimeUTC" : "2015-04-23T15:12:12.511",
  "generationTimeUTC" : "2015-04-23T15:11:56.507"
}
```

Set multiple parameters `/parameter/_set`

Set single parameter

POST /parameter/_set

# Chapter 3. WebSocket API

Yamcs provides a WebSocket-API for subscribing to data. A typical use case would be a display tool subscribing to parameter updates. All operations support binary WebSocket frames encoded using Google Protocol Buffers, as well as textual WebSocket frames encoded in JSON. The choice between Protobuf and JSON is currently determined based on the type of the first received client frame. If it's binary, all further message will be Protobuf, if it's text, all further messages will be in JSON.

### Wrapper

Assuming the default *simulator* instance, WebSocket calls should be directed to a URL of the form `http://localhost:8090/simulator/_websocket`. The frame must contain a text array like this: `[x,y,z,{"<request-type>":"<request>"}]` where:

- `x` is the version of the protocol (currently at 1)
- `y` is the message type. One of:
  - `1` Request
  - `2` Reply
  - `3` Exception
  - `4` Data
- `z` is a sequence counter. Basically this allows clients to couple a response with the original request.

## 3.1. Alarm Notices

The `alarms` resource type within the [[ WebSocket API ]] allows subscribing to alarm updates.

We recommend using the LGPL Java WebSocket client distributed as part of the yamcs-api jar, but if that is not an option, this is the protocol:

### Subscribe to Alarm Notices

Within the WebSocket request envelope use these values:
* request-type `alarms`
* request `subscribe`

This will make your web socket connection receive updates of the type `ProtoDataType.ALARM`.

Directly after you subscribe, you will also get the active set of alarms – if any.

Here's example output in JSON (with Protobuf, there's an applicable getter in the `WebSocketSubscriptionData`). Notice how we are first getting an `ALARM` of type `1` ( `ACTIVE` ) that triggered somewhere before we connected, and only then a further update on that alarm of type `4` ( `PVAL_UPDATED` ) (because the parameter value was updated).

```
[1,2,3]
[1,4,0,{"dt":"ALARM","data":{"id":0,"type":1,"triggerValue":{"id":
```

```
{"name":"/YSS/SIMULATOR/O2TankTemp"},"rawValue":
{"type":2,"uint32Value":227},"engValue":
{"type":2,"uint32Value":227},"acquisitionTime":1440576556724,"generationT
08-26T08:08:40.724","generationTimeUTC":"2015-08-
26T08:08:23.714","watchLow":10.0,"watchHigh":12.0,"warningLow":30.0,"warn
08-26T08:08:42.224"},"mostSevereValue":{"id":
{"name":"/YSS/SIMULATOR/O2TankTemp"},"rawValue":
{"type":2,"uint32Value":227},"engValue":
{"type":2,"uint32Value":227},"acquisitionTime":1440576556724,"generationT
08-26T08:08:40.724","generationTimeUTC":"2015-08-
26T08:08:23.714","watchLow":10.0,"watchHigh":12.0,"warningLow":30.0,"warn
08-26T08:08:42.224"},"currentValue":{"id":
{"name":"/YSS/SIMULATOR/O2TankTemp"},"rawValue":
{"type":2,"uint32Value":258},"engValue":
{"type":2,"uint32Value":258},"acquisitionTime":1440576955780,"generationT
08-26T08:15:19.780","generationTimeUTC":"2015-08-
26T08:15:02.777","watchLow":10.0,"watchHigh":12.0,"warningLow":30.0,"warn
08-26T08:15:21.280"},"violations":65}}]
[1,4,1,{"dt":"ALARM","data":{"id":0,"type":4,"triggerValue":{"id":
{"name":"/YSS/SIMULATOR/O2TankTemp"},"rawValue":
{"type":2,"uint32Value":227},"engValue":
{"type":2,"uint32Value":227},"acquisitionTime":1440576556724,"generationT
08-26T08:08:40.724","generationTimeUTC":"2015-08-
26T08:08:23.714","watchLow":10.0,"watchHigh":12.0,"warningLow":30.0,"warn
08-26T08:08:42.224"},"mostSevereValue":{"id":
{"name":"/YSS/SIMULATOR/O2TankTemp"},"rawValue":
{"type":2,"uint32Value":227},"engValue":
{"type":2,"uint32Value":227},"acquisitionTime":1440576556724,"generationT
08-26T08:08:40.724","generationTimeUTC":"2015-08-
26T08:08:23.714","watchLow":10.0,"watchHigh":12.0,"warningLow":30.0,"warn
08-26T08:08:42.224"},"currentValue":{"id":
{"name":"/YSS/SIMULATOR/O2TankTemp"},"rawValue":
{"type":2,"uint32Value":280},"engValue":
{"type":2,"uint32Value":280},"acquisitionTime":1440576962013,"generationT
08-26T08:15:26.013","generationTimeUTC":"2015-08-
26T08:15:09.011","watchLow":10.0,"watchHigh":12.0,"warningLow":30.0,"warn
08-26T08:15:27.513"},"violations":66}}]
```


## Unsubscribe

Within the WebSocket request envelope use these values:
* request-type `alarms`
* request `unsubscribe`

This will stop your WebSocket connection from getting further alarm updates.

## 3.2. Event Updates

The `events` resource type within the [[ WebSocket API ]] allows subscribing to event updates.

We recommend using the LGPL Java WebSocket client distributed as part of the yamcs-api jar. But for deeper understanding, this is the protocol:

### Subscribe

Within the WebSocket request envelope use these values:
* request-type `events`
* request `subscribe`

This will make your web socket connection receive updates of the type `ProtoDataType.EVENT`.

Here's example output in JSON (with Protobuf, there's an applicable getter in the `WebSocketSubscriptionData`).

```
[1,2,3]
[1,4,0,{"dt":"EVENT","data":
{"source":"CustomAlgorithm","generationTime":1440823760490,"receptionTime

[1,4,1,{"dt":"EVENT","data":
{"source":"CustomAlgorithm","generationTime":1440823765491,"receptionTime

[1,4,2,{"dt":"EVENT","data":
{"source":"CustomAlgorithm","generationTime":1440823770490,"receptionTime
```

### Unsubscribe

Within the WebSocket request envelope use these values:
* request-type `events`
* request `unsubscribe`

This will stop your WebSocket connection from getting further event updates.

## 3.3. Management Updates

The `management` resource type within the [[ WebSocket API ]] groups general Yamcs info that does not fit under a specific other resource type.

We recommend using the LGPL Java WebSocket client distributed as part of the yamcs-api jar, but if that is not an option, this is the protocol:

### Subscribe to Management Updates

Within the websocket request envelope use these values:
* request-type `management`
* request `subscribe`

This will make your web socket connection receive updates on the following Yamcs data types (aka `ProtoDataType`):

CLIENT_INFO

Updates on a client that is or was connected to Yamcs. Directly after sending a `management/subscribe` request, you will also get an update for all the clients that are connected at that point. As soon as a client is disconnected, you will also get an update on that.

Here's example output in JSON (with Protobuf, there's an applicable getter in the `WebSocketSubscriptionData`), where there are two anonymous clients connected to a non-secured deployment of Yamcs:

```
[1,2,3]
[1,4,1,{"dt":"CLIENT_INFO","data":
{"instance":"simulator","id":1,"username":"unknown","applicationName":"Un

[1,4,2,{"dt":"CLIENT_INFO","data":
{"instance":"simulator","id":3,"username":"unknown","applicationName":"Un
```

After the empty initial ACK, we receive two data messages, one for each connected client at that point. Notice the `currentClient` field which indicates whether this client info concerns your own web socket session. The `state` field indicates either CONNECTED (0) or DISCONNECTED (1). In case the other use would close its connection, we would thus get an update on that too.

PROCESSOR_INFO

Updates on the lifecycle of Yamcs TM/TC processors. Directly after sending a `management/subscribe` request, you will also get an update for all the processors at that point.

Here's an example output in JSON (with Protobuf, there's an applicable getter in the `WebSocketSubscriptionData`), where there is just one processor `realtime`.

```
[1,2,3]
[1,4,0,{"dt":"PROCESSOR_INFO","data":
{"instance":"simulator","name":"realtime","type":"realtime","creator":"sy
```

PROCESSING_STATISTICS

General statistics on processors. For every high-level packet, shows you the the current processing time, an increasing data count, and many more info. Updates on this data type are a bit noisier than the client_info/processor_info updates, so we might decide to move this out to a different subscription request in the future.

Here's an example output in JSON (with Protobuf, there's an applicable getter in the `WebSocketSubscriptionData`):

```
[1,4,3,{"dt":"PROCESSING_STATISTICS","data":
{"instance":"simulator","yProcessorName":"realtime","tmstats":
[{"packetName":"RCS","receivedPackets":2378,"lastReceived":1438235693322,
{"packetName":"FlightData","receivedPackets":73718,"lastReceived":1438235
{"packetName":"ccsds-
default","receivedPackets":2378,"lastReceived":1438235693322,"lastPacketT
{"packetName":"Power","receivedPackets":2378,"lastReceived":1438235693322
{"packetName":"DHS","receivedPackets":2378,"lastReceived":1438235693322,"

[1,4,4,{"dt":"PROCESSING_STATISTICS","data":
{"instance":"simulator","yProcessorName":"realtime","tmstats":
[{"packetName":"RCS","receivedPackets":2378,"lastReceived":1438235693322,
{"packetName":"FlightData","receivedPackets":73723,"lastReceived":1438235
{"packetName":"ccsds-
default","receivedPackets":2378,"lastReceived":1438235693322,"lastPacketT
{"packetName":"Power","receivedPackets":2378,"lastReceived":1438235693322
{"packetName":"DHS","receivedPackets":2378,"lastReceived":1438235693322,"
```

## Get Client-Info

Within the websocket request envelope use these values:
* request-type `management`
* request `getClientInfo`

This will return you a *one-time* data frame containing your own client info. The format is exactly the same as for the above described subscription updates.

## 3.4. Sreams

The `stream` resource type within the [[ WebSocket API ]] groups low-level publish/subscribe operations on [[ Yamcs Streams ]].

The documented operations work on one of the built-in streams (like `tm`, `tm_realtime`, `tm_dump`, `pp_realtime`, `cmdhist_realtime`, etc). Or, if your Yamcs deployment defines any other streams, they would work as well.

We recommend using the LGPL Java WebSocket client distributed as part of the yamcs-api jar, but for deeper understanding, this is the protocol:

### Subscribe to a Stream

Within the websocket request envelope use these values:
* request-type `stream`
* request `subscribe`
* data
* With Protobuf: an object of type `Yamcs.StreamSubscribeRequest` where at least the `stream` name is filled in.
* With JSON: an object literal where at least the `stream` key is set.

Here's a full request example in JSON-notation

```
[1,1,3,{"stream": "subscribe", "data": {"stream": "tm_realtime"}}]
```

We are thinking of maybe adding other options, like a way to limit the data by filtering on one of the stream's columns, but this is pending an actual use case.

As a result of the above call you will get updates whenever anybody publishes data to the specified stream. With Protobuf, the data can be fetched with the `getStreamData()`-method on in the `WebSocketSubscriptionData` object. With JSON, you might see something like this sample output:

```
[1,2,3]
[1,4,0,{"dt":"STREAM_DATA","data":
{"stream":"tm_realtime","columnValue":[{"columnName":"gentime","value":
{"type":6,"timestampValue":1438608491320}},
{"columnName":"seqNum","value":{"type":3,"sint32Value":134283264}},
{"columnName":"rectime","value":
{"type":6,"timestampValue":1438608508323}},
{"columnName":"packet","value":
{"type":4,"binaryValue":"CAEAAAAPQuou2FJFAAAABOcAAAAAAA=="}}]}}]
[1,4,1,{"dt":"STREAM_DATA","data":
{"stream":"tm_realtime","columnValue":[{"columnName":"gentime","value":
{"type":6,"timestampValue":1438608491320}},
{"columnName":"seqNum","value":{"type":3,"sint32Value":134283264}},
{"columnName":"rectime","value":
```

```
{"type":6,"timestampValue":1438608508323}},
{"columnName":"packet","value":
{"type":4,"binaryValue":"CAEAAAAPQuou2FJFAAAABOcAAAAAAA=="}}]}}]
```

In the case we were receiving some simulated data from the `tm_realtime` stream, this is a built-in stream with columns `gentime`, `rectime`, `seqNum` and `packet`. This last column is of binary format (it's the raw TM packet), which is why it is Base64-encoded in the JSON output.

Other streams would have different columns.

### Publish to a Stream

Within the websocket request envelope use these values:
* request-type `stream`
* request `publish`
* data
* With Protobuf: an object of type `Yamcs.StreamData` where the `stream` key is set to the targeted stream, and where column-values are added for every column of that stream's definition.
* With JSON: an object literal where the `stream` key is set to the targeted stream, and where `columnValue` is an array of `columnName` - `value` literals.

```
The provided type has to match the type of the actual stream's
definition.
```

Here's a full publish-request to the `tm_realtime`-stream in JSON-notation. Notice the similarities with the return type on the subscribe-operation.

```
[1,1,3,{"stream":"publish","data":
{"stream":"tm_realtime","columnValue":[{"columnName":"gentime","value":
{"type":6,"timestampValue":1438608491320}},
{"columnName":"seqNum","value":{"type":3,"sint32Value":134283264}},
{"columnName":"rectime","value":
{"type":6,"timestampValue":1438608508323}},
{"columnName":"packet","value":
{"type":4,"binaryValue":"CAEAAAAPQuou2FJFAAAABOcAAAAAAA=="}}]}}]
```

As as result of the above call the server will write that data to the requested stream (hence notifying anybody that might be subscribed to that stream), and return with just an empty ACK:

```
[1,2,3]
```