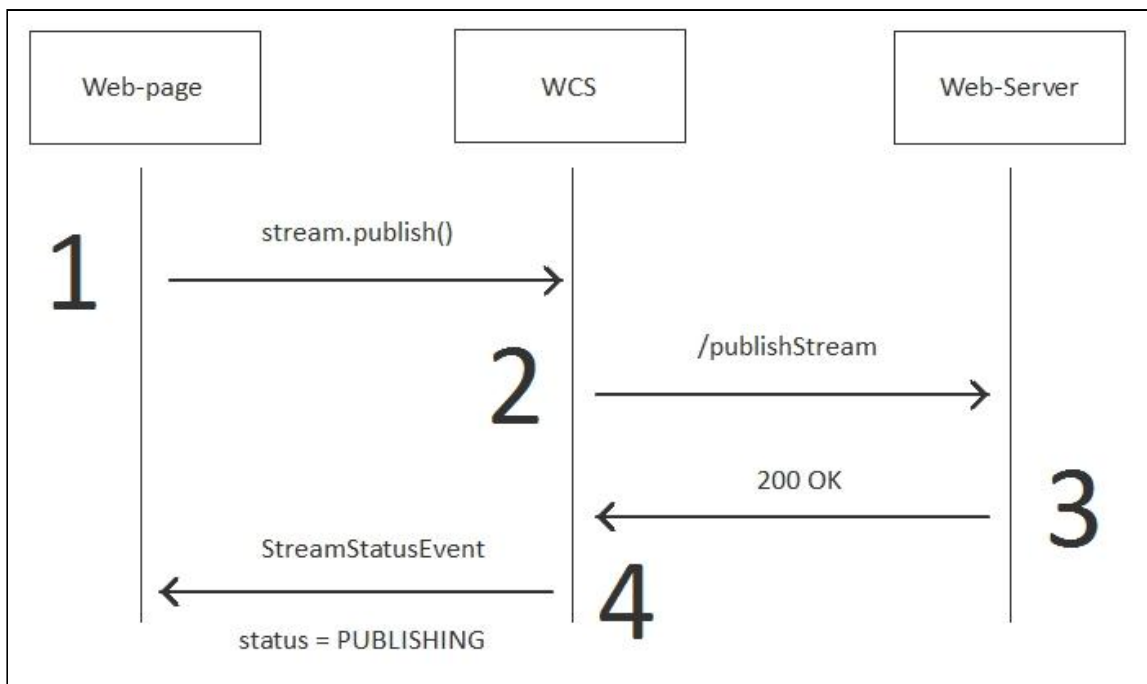


Type 2 - the direct invoke

Workflow example using `publishStream` hook

The `publishStream` REST hook is a direct invoke method, because a client invokes this method with the `Stream.publish()` command attempting to publish a video stream from the web camera. This operation can be authorized – cancelled or permitted – and parameters of this operation can be redefined by backend server. For example, the field `name: "stream1"` may be changed to `name: "stream2"`, and if such a replace succeeds, WCS will publish the stream with the new name `stream2`.

The invocation flow:



Example:

REST hook

```
POST /rest/my_api/publishStream HTTP/1.1
Accept: application/json
Content-Type: application/json;charset=UTF-8
User-Agent: Java/1.8.0_111
Host: 192.168.1.101
Connection: keep-alive
Content-Length: 3611
```

```
{
```

```
"nodeId": "Hw47CFMBEchVOpBMDr29IxjudnJ1sj0Y@192.168.1.101",
"appKey": "defaultApp",
"sessionId": "/192.168.1.102:4388/192.168.1.101:8443",
"mediaSessionId": "56141d10-fddc-11e6-ac3a-4d67d5b3360d",
"name": "b4e7",
"published": true,
"hasVideo": true,
"hasAudio": true,
"status": "PENDING",
"sdp": ".....",
"record": false,
"width": 0,
"height": 0,
"bitrate": 0,
"quality": 0,
"mediaProvider": "WebRTC"
}
```

Backend response

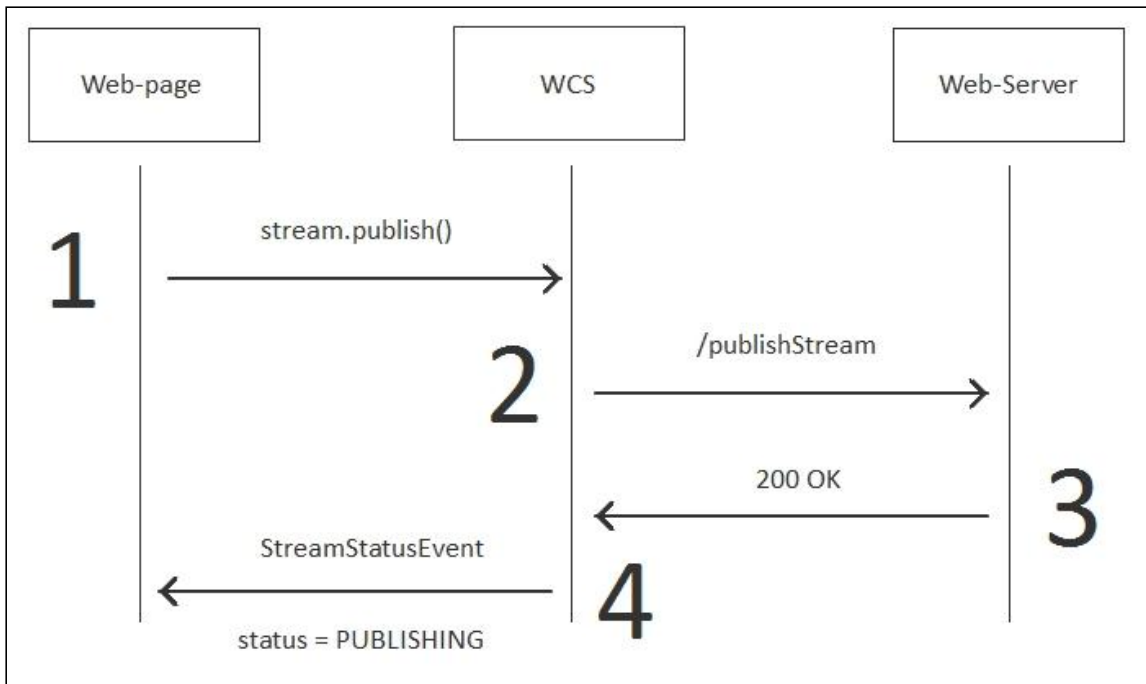
```
HTTP/1.1 200 OK
Date: Tue, 28 Feb 2017 17:35:43 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.3.3
Content-Length: 3653
Connection: close
Content-Type: application/json

{
  "nodeId": "Hw47CFMBEchVOpBMDr29IxjudnJ1sj0Y@192.168.1.101",
  "appKey": "defaultApp",
  "sessionId": "\/192.168.1.102:4388\/192.168.1.101:8443",
  "mediaSessionId": "56141d10-fddc-11e6-ac3a-4d67d5b3360d",
  "name": "b4e7",
  "published": true,
  "hasVideo": true,
  "hasAudio": true,
  "status": "PENDING",
  "sdp": "",
  "record": false,
  "width": 0,
  "height": 0,
  "bitrate": 0,
  "quality": 0,
  "mediaProvider": "WebRTC"
}
```

Authentication

Authentication is similar to `connect` method. You can send a token or a password here too, as well as permit/forbid the operation based on other parameters.

For authentication you need to configure the `publishStream` method at the connection stage (`connect`) and set `restOnError: FAIL` in the `restClientConfig` object.



Example:

REST hook

```

POST /rest/my_api/publishStream HTTP/1.1
Accept: application/json
Content-Type: application/json;charset=UTF-8
User-Agent: Java/1.8.0_111
Host: 192.168.1.101
Connection: keep-alive
Content-Length: 3639

{
  "nodeId": "Hw47CFMBEchVOpBMDr29IxjudnJ1sj0Y@192.168.1.101",
  "appKey": "defaultApp",
  "sessionId": "/192.168.1.102:17749/192.168.1.101:8443",
  "mediaSessionId": "0e17ab50-fdbc-11e6-8a47-c5292ef61cc0",
  "name": "3a88",
  "published": true,
  "hasVideo": true,
  "hasAudio": true,
  "status": "PENDING",
  "sdp": ".....",
  "record": false,
  "width": 0,
  "height": 0,
  "bitrate": 0,
  "quality": 0,
  "mediaProvider": "WebRTC",
  "custom": {
    "token": "abcdef"
  }
}
  
```

Backend response

```
HTTP/1.1 403 Forbidden
Date: Tue, 28 Feb 2017 13:44:39 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.3.3
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

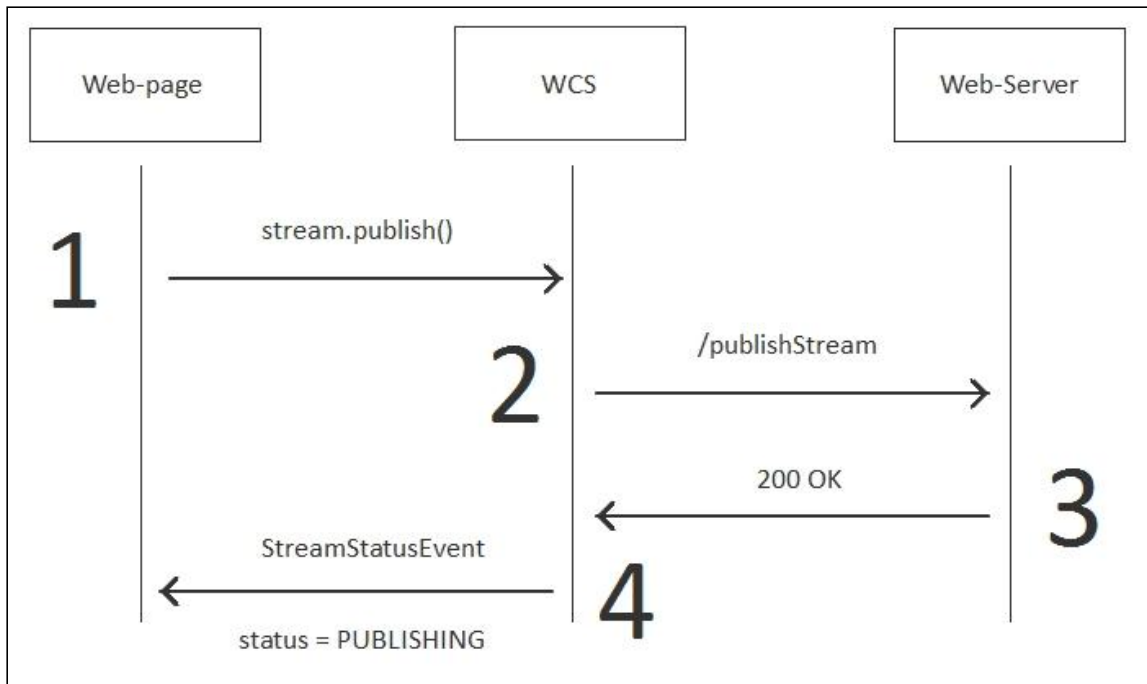
Redefining fields

Redefining fields works when the method operates normally and the `200 OK` response is returned. The purpose of redefining fields is, for instance, to hide the real name of the stream on the page from the end user.

For example, the user publishes `stream1`, but the backend server changes the name to `stream2`. The feature can be also used to hide an original RTSP stream address from viewers.

To make such redefining, do the following:

1. Enable `"restPolicy": "OVERWRITE"` in `restClientConfig` for the `publishStream` when connecting to the server.
2. Put a comma-separated list of fields to overwrite in `"restOverwrite": ""`, for example, `"restOverwrite": "name"` to `restClientConfig`. In this case, only the name field – the name of a stream – is overwritten.
3. In the JSON body of the `200 OK` response return the modified `name` field, and return other fields as they were received from the WCS server.



Example:

REST hook

```

POST /rest/my_api/publishStream HTTP/1.1
Accept: application/json
Content-Type: application/json;charset=UTF-8
User-Agent: Java/1.8.0_111
Host: 192.168.1.101
Connection: keep-alive
Content-Length: 3612

{
  "nodeId": "Hw47CFMBEchV0pBMDr29IxjudnJ1sj0Y@192.168.1.101",
  "appKey": "defaultApp",
  "sessionId": "/192.168.1.102:12514/192.168.1.101:8443",
  "mediaSessionId": "abbe7e90-fdcd-11e6-9831-bd76c6e822a1",
  "name": "2a0c",
  "published": true,
  "hasVideo": true,
  "hasAudio": true,
  "status": "PENDING",
  "sdp": ".....",
  "record": false,
  "width": 0,
  "height": 0,
  "bitrate": 0,
  "quality": 0,
  "mediaProvider": "WebRTC"
}
  
```

Backend response

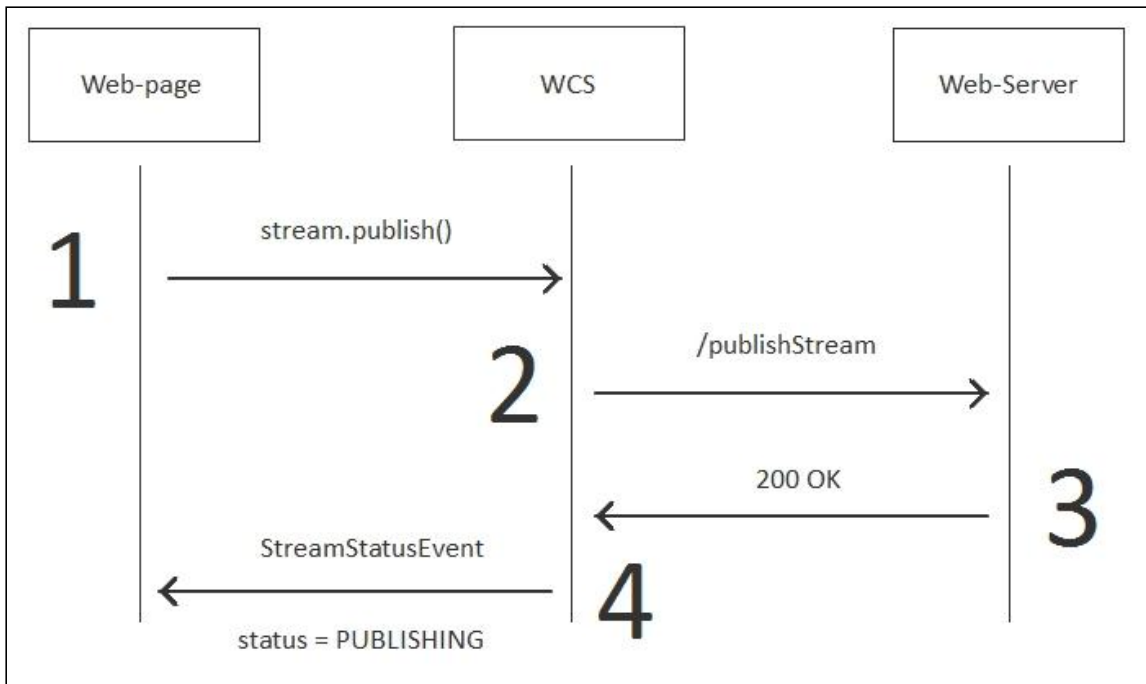
```
HTTP/1.1 200 OK
Date: Tue, 28 Feb 2017 15:50:44 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.3.3
Content-Length: 3669
Connection: close
Content-Type: application/json

{
  "nodeId": "Hw47CFMBEchVOpBMDr29IxjudnJ1sj0Y@192.168.1.101",
  "appKey": "defaultApp",
  "sessionId": "\/192.168.1.102:12514\/192.168.1.101:8443",
  "mediaSessionId": "abbe7e90-fdcd-11e6-9831-bd76c6e822a1",
  "name": "streamChangedName",
  "published": true,
  "hasVideo": true,
  "hasAudio": true,
  "status": "PENDING",
  "sdp": ".....",
  "record": false,
  "width": 0,
  "height": 0,
  "bitrate": 0,
  "quality": 0,
  "mediaProvider": "WebRTC"
}
```

Excluding fields from sending

This method works in one direction only, from WCS to the backend server. This means you can exclude one or more fields that the WCS server sends to the backend server in the JSON body of the response on step 2 below.

By default, `publishStream` sends all fields. To exclude field, put a comma-separated list of fields you want to exclude to the `"restExclude": ""` parameter in `restClientConfig` when establishing connection.



In the request/response example below the name field – the name of a stream – is excluded:

REST hook

```

POST /rest/my_api/publishStream HTTP/1.1
Accept: application/json
Content-Type: application/json;charset=UTF-8
User-Agent: Java/1.8.0_111
Host: 192.168.1.101
Connection: keep-alive
Content-Length: 3602

{
  "nodeId": "Hw47CFMBEchVOpBMDr29IxjudnJ1sj0Y@192.168.1.101",
  "appKey": "defaultApp",
  "sessionId": "/192.168.1.102:9232/192.168.1.101:8443",
  "mediaSessionId": "084db2f0-fdd5-11e6-9ba5-6ba06f30ad92",
  "published": true,
  "hasVideo": true,
  "hasAudio": true,
  "status": "PENDING",
  "sdp": ".....",
  "record": false,
  "width": 0,
  "height": 0,
  "bitrate": 0,
  "quality": 0,
  "mediaProvider": "WebRTC"
}
  
```

Backend response

```

HTTP/1.1 200 OK
Date: Tue, 28 Feb 2017 16:43:26 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.3.3
Content-Length: 3649
Connection: close
Content-Type: application/json

{
  "nodeId": "Hw47CFMBEchVOpBMDr29IxjudnJ1sj0Y@192.168.1.101",
  "appKey": "defaultApp",
  "sessionId": "\/192.168.1.102:9232\/192.168.1.101:8443",
  "mediaSessionId": "084db2f0-fdd5-11e6-9ba5-6ba06f30ad92",
  "published": true,
  "hasVideo": true,
  "hasAudio": true,
  "status": "PENDING",
  "sdp": ".....",
  "record": false,
  "width": 0,
  "height": 0,
  "bitrate": 0,
  "quality": 0,
  "mediaProvider": "WebRTC"
}

```

Parsing stream URL parameters

When RTMP stream is published or played on WCS, RTMP connection and stream parameters may be set in stream URL like this:

```

rtmp://host:1935/live?connectParam1=val1&connectParam2=val2/streamName?
streamParam1=val1&streamParam2=val2

```

Where

- `host` is WCS server hostname;
- `connectParam1`, `connectParam2` are RTMP connection parameters;
- `streamName` is stream name on server;
- `streamParam1`, `streamParam2` are stream parameters.

WCS server passes the parameters to backend server in `custom` field, for example:

1. Connection parameters (REST hook `/connect`)

```

POST /apps/EchoApp/connect HTTP/1.1
HOST: localhost:8081

{
  "nodeId" : "Qb3rAjf31zoy6PE11WZkUhrG1DsTykgj@192.168.1.1",

```



```

    "appKey" : "flashStreamingApp",
    "sessionId" : "/127.0.0.1:5643/192.168.1.1:1935",
    "useWsTunnel" : false,
    "useWsTunnelPacketization2" : false,
    "useBase64BinaryEncoding" : false,
    "keepAlive" : false,
    "custom" : {
      "connectParam1" : "val1",
      "connectParam2" : "val2"
    }
  }
}

```

2. Publishing parameters (REST hook `/publishStream`)

```

POST /apps/EchoApp/publishStream HTTP/1.1
HOST: localhost:8081

{
  "nodeId" : "Qb3rAjf3lzoy6PE11WZkUhRG1DsTykgj@192.168.1.1",
  "appKey" : "flashStreamingApp",
  "sessionId" : "/127.0.0.1:5643/192.168.1.1:1935",
  "mediaSessionId" : "627990f9-8fe5-4e92-bb2a-863cc4eb43de",
  "name" : "stream1",
  "published" : true,
  "hasVideo" : false,
  "hasAudio" : true,
  "status" : "NEW",
  "record" : true,
  "width" : 0,
  "height" : 0,
  "bitrate" : 0,
  "minBitrate" : 0,
  "maxBitrate" : 0,
  "quality" : 0,
  "mediaProvider" : "Flash",
  "custom" : {
    "streamParam1" : "val1",
    "streamParam2" : "val2"
  }
}

```

3. Playback parameters (REST hook `/playStream`)

```

POST /apps/EchoApp/playStream HTTP/1.1
HOST: localhost:8081

{
  "nodeId" : "Qb3rAjf3lzoy6PE11WZkUhRG1DsTykgj@192.168.1.1",
  "appKey" : "flashStreamingApp",
  "sessionId" : "/127.0.0.1:5643/192.168.1.1:1935",
  "mediaSessionId" : "stream1/127.0.0.1:5643/192.168.1.1:1935",
  "name" : "stream1",
  "published" : false,
  "hasVideo" : true,
  "hasAudio" : true,
  "status" : "NEW",

```

```
"record" : false,  
"width" : 0,  
"height" : 0,  
"bitrate" : 0,  
"minBitrate" : 0,  
"maxBitrate" : 0,  
"quality" : 0,  
"mediaProvider" : "Flash",  
"custom" : {  
  "streamParam1" : "val1",  
  "streamParam2" : "val2"  
}  
}
```

This feature can be used for example to authenticate client on backend server while publishing or playing RTMP-stream on WCS server.

`playHLS` and `playRTSP` methods

`playHLS` and `playRTSP` methods are intended for stream playback authentication via [HLS](#) and [RTSP](#), accordingly. These methods are called without first calling of `/connect` method, so it is impossible to change error handling policy and other parameters via `restClientConfig`, the following policy is always used:

```
"restOnError": "FAIL"
```