

# Mixer load testing

## Overview

If server use case involves stream mixing, mixer load testing may be necessary before you put server into production. The mixer testing is carried out as follows:

1. The required number of streams are published on server (at least one stream per mixer)
2. The specified number of audiomixers are created, and streams are fed to mixers input. One stream can be fed to one mixer input only
3. Mixers work for desired time then they will be destroyed and created again until the test is finished.

The server behavior can be observed with [monitoring](#) tools while testing.

To manage mixer load testing the special REST API queries are used.

## REST API

A REST-query must be an HTTP/HTTPS POST query in the following form:

- HTTP: `http://streaming.flashphoner.com:8081/rest-api/mixer/test/start`
- HTTPS: `https://streaming.flashphoner.com:8444/rest-api/mixer/test/start`

Here:

- `streaming.flashphoner.com` - is the address of the WCS server
- `8081` - the standard REST / HTTP port of the WCS server
- `8444` - the standard HTTPS port
- `rest-api` - the required prefix
- `mixer/test/start` - the REST-method used

## REST methods

### **/mixer/test/start**

Start the test

REQUEST EXAMPLE

```
POST /mixer/test/start HTTP/1.1
Host: localhost:8081
Content-Type: application/json

{
  "feedingStreams": [
    "s1",
    "s2",
    "s3",
    "s4"
  ],
  "mixerCount": 2,
  "streamsInMixer": 2,
  "intervalInSeconds": 60
}
```

#### RESPONSE EXAMPLE

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json
```

#### RETURN CODES

Code	Reason
200	OK
500	Internal error

#### /mixer/test/stop

Stop the test

#### REQUEST EXAMPLE

```
POST /mixer/test/stop HTTP/1.1
Host: localhost:8081
Content-Type: application/json

{
}
```

#### RESPONSE EXAMPLE

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json
```

#### RETURN CODES

Code	Reason
------	--------

Code	Reason
200	OK
404	Mixer not found
500	Internal error

### /mixer/test/get\_start\_example

Return JSON object sample to pass to `/mixer/test/start` method

#### REQUEST EXAMPLE

```
POST /mixer/test/get_start_example HTTP/1.1
Host: localhost:8081
Content-Type: application/json

{
  "feedingStreams": [
    "stream1",
    "stream2",
    "stream3"
  ],
  "mixerCount": 3,
  "streamsInMixer": 1,
  "intervalInSeconds": 60
}
```

#### RESPONSE EXAMPLE

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json
```

#### RETURN CODES

Code	Reason
200	OK
500	Internal error

### Parameters

Parameter name	Description	Example
feedingStreams	Stream published list to participate in test	<pre>[ "s1", "s2", "s3", "s4" ]</pre>

Parameter name	Description	Example
mixerCount	Number of mixers created	2
streamsInMixer	Number of streams fed to each mixer input	2
intervalInSeconds	Interval in seconds to destroy mixers and create them again	60

## Configuration

To test mixer performance under high load, asynchronous media session disconnection (which is enabled by default) should be disabled

```
handler_async_disconnect=false
```

WCS should be restarted to apply.

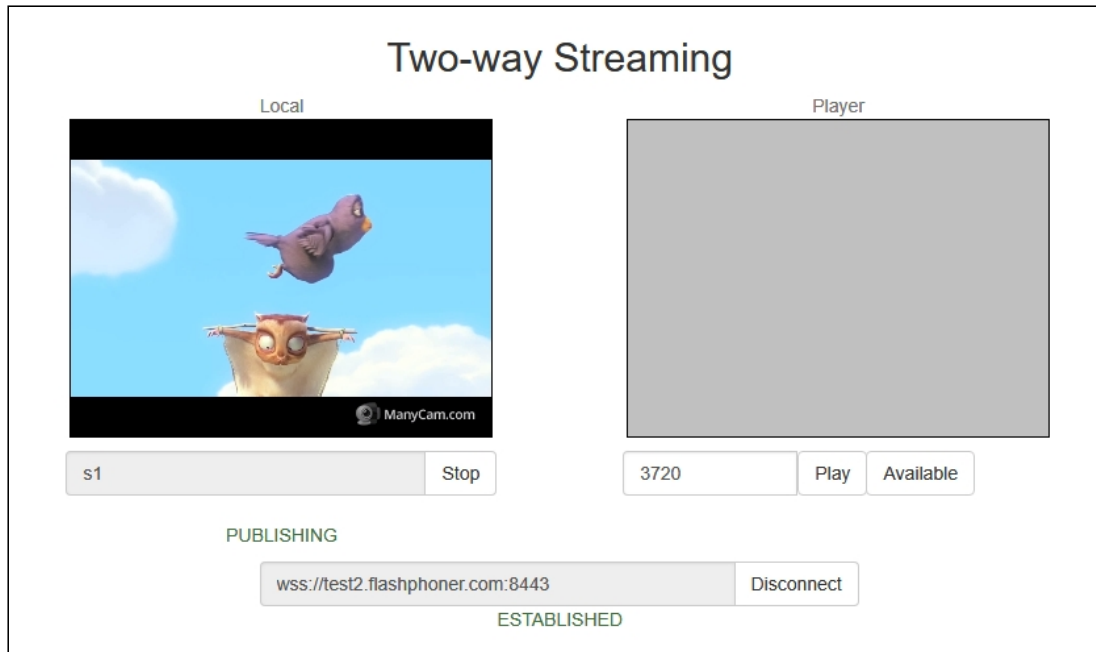
When test is finished, this setting must be removed from configuration file.

## Testing

1. For test we use:

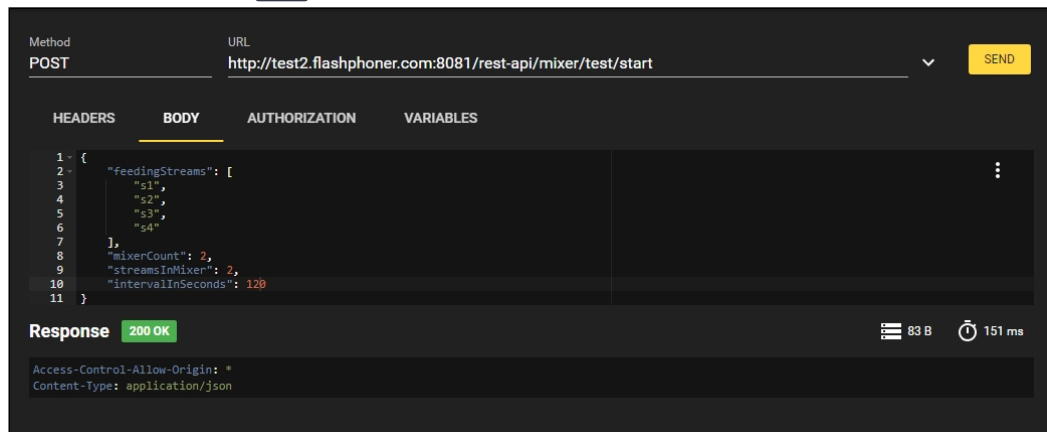
- WCS server
- Chrome browser and [REST-client](#) to send queries
- `Two Way Streaming` web application to publish streams

2. Publish streams named `s1`, `s2`, `s3`, `s4`

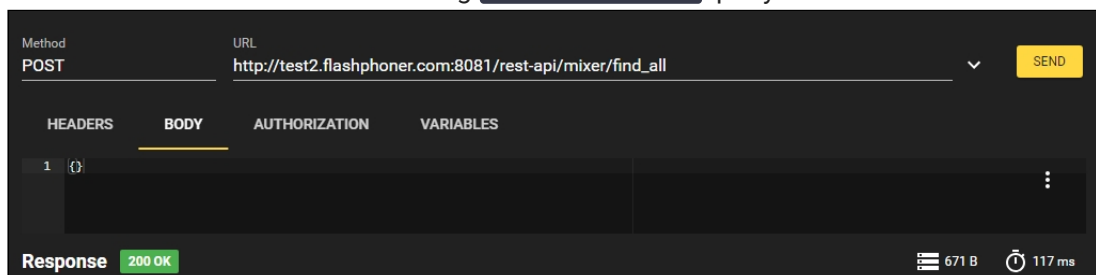


3. Open REST client. Send `/mixer/test/start` query with the following parameters:

- streams published list: `s1, s2, s3, s4`
- number of mixers: `2`
- number of streams per mixer: `2`
- mixer work interval: `120` seconds



4. Make sure mixers are created sending `/mixer/find_all` query



Two mixers `mixer0` and `mixer1` should be found

```
Access-Control-Allow-Origin: *
Content-Type: application/json
Content-Length: 567

{
  "localMediaSessionId": "70783957-e03f-46cc-975e-a9fe174950a7",
  "localStreamName": "mixer1",
  "uri": "mixer://mixer1",
  "status": "PROCESSED_LOCAL",
  "hasAudio": true,
  "hasVideo": true,
  "record": false,
  "mediaSessions": [
    "d619b1d0-a5b2-11e8-8348-ebb0edd6d2e8",
    "ca05f570-a5b2-11e8-8508-b1c91dbedb18"
  ]
},
{
  "localMediaSessionId": "7be960fc-08fe-444f-86ee-c3ceb1d87ac5",
  "localStreamName": "mixer0",
  "uri": "mixer://mixer0",
  "status": "PROCESSED_LOCAL",
  "hasAudio": true,
  "hasVideo": true,
  "record": false,
  "mediaSessions": [
    "af49ef20-a5b2-11e8-bf17-ab9c5e415872",
    "bba6c270-a5b2-11e8-8b25-f7438838b4d6"
  ]
}
```

5. Server resource consumption can be observed while testing with [Java Mission Control](#), [load and resource usage information](#) and [error information](#) pages, and [server logs](#).
6. Stop the test with `/mixer/test/stop` query

```
Method: POST
URL: http://test2.flashphoner.com:8081/rest-api/mixer/test/stop
SEND

HEADERS:
BODY: {}
AUTHORIZATION:
VARIABLES:

Response: 200 OK
83 B 61 ms
Access-Control-Allow-Origin: *
Content-Type: application/json
```

## Tuning recommendations

1. If large CPU load was detected during testing, follow [server tuning recommendations](#).
2. If resource leak was detected during testing, send detailed test description and [report collected](#) to [support@flashphoner.com](mailto:support@flashphoner.com)