DynamR API

Documentation

v2.4

Last updated: 12/27/2019

The purpose of this document is to explain how one can leverage DynamR to controls the lights, tame sharks and make big sheets.

Table of Contents

In case of emergency:

- Tom <tom@dynamr.com>
- David <david@dynamr.com>

Date	Version	Publisher	Comment
2017-05-28	2.0	Tom	Initial release
2017-07-26	2.1	Tom	Add session status
2019-07-18	2.2	Tom	Add round speed controls Add prefixes info Rename message to payload Add info about responses
2019-11-23	2.3	Tom	Distinguish DynamR server & API communication Add network configuration API
2019-12-27	2.4	Tom	Add light endpoints

Overview

Abstract

DynamR is a embedded autonomous program that allow to live-judge wind tunnel dynamic flying competition and controls the light system required to do so. It adds some laser, sharks & sheets too.

Terms

In this documentation, each item involved has a dedicated term and signification:

- `server` : this is the DynamR part, where one have to connect to.
- `client` : the software part that has to be implemented on your own, to communicate with the `server`.
- `session` : represent the time where a customer (first flyers, teams, pro flyer, ...) is using the wind tunnel and for which lights are managed.
- `preset` : a predefined configuration for lights (ie. right stripe blue, left stripe red, ...). This is managed by DynamR.
- `round_speed` : a speed round of dynamic flying.
- `light` : the LED stripes.

Communication

Due to technical constraints, there is 2 ways to communicate with DynamR: the server & the API. We strive to keep all the things the most consistent possible, but some times you may need to go a little bit “low level”. Unless explicitly specified, you should use the server communication,

Communication with DynamR is based on the REST philosophy and use JSON representation for the data.

DynamR server

This is the main way of communication.

The server is accessible by an `hostname/ip` and a `port` couple. In this document, the server is located at `dynamr.local:8080`, please refer to your technical staff to get the correct value.

The `client` MUST send its `payload` using an `HTTP POST request` to the `/call` endpoint of the `server`, ie. `http://dynamr.local:8080/call`. The `payload` MUST be a valid JSON string and `Content-Type` MUST be `application/json`. Depending of the result, the DynamR `server` will return a confirmation or throw an exception (both always JSON-serialized).

Warning: Response code is always 200, even if there is an error while processing the payload. You must check error from within the payload.

```
curl -H "Content-Type: application/json" -d
'{"procedure":"com.dynamr.something","args":["payload","goes",
{"here":true}],"kwargs":{"metas":{"caller":"windtunnel"}}}'
http://dynamr.local:8080/call
```

Payload format

When the `client` send a payload to the `server`, it MUST respect this format:

Key	Type	Description	Mandatory
`procedure`	string	The remote procedure (prefix + method)	Yes
`args`	array	The payload to send	No
`kwargs`	hashmap	Extras informations	No

Each key will be defined in respective parts of this document. Depending of the procedure, `args`/`kwargs` could be mandatory.

Sample payload

```
{
  "procedure": "com.dynamr.something",
  "args": [
    "payload",
    "goes",
    {"here": true}
  ],
  "kwargs": {
    "metas": {
      "foo": "bar"
    }
  }
}
```

Response format

Success

Response will mostly contains 2 types of data:

1. status resulting from a command (start/stop/...)
2. data specifically retrieved

When getting status, data try to be “stateless”, it should contains everything you need to react to.

Key	Type	Description
args	array	Response value

Sample

```
{
  "args": [
    "response",
    "goes",
    {"here": true}
  ]
}
```

Failure

Key	Type	Description
error	string	Error message
args	array	Error description
kwargs	hashmap	

Sample

```
{
  "error": "wamp.error.no_such_procedure",
  "args": [
    "no callee registered for procedure <com.foo.bar>"
  ],
  "kwargs": {}
}
```

API

The API is accessible by an `hostname/ip` and a `port`. In this document, the server is located at `dynamr.local:5000`, please refer to your technical staff to get the correct value.

It’s a standard JSON REST API.

Presets

Prefix is: “com.dynamr.light_preset”.

Entity

Variable	Type	Description
id	integer	
name	string	Human name
colors	hashmap	key: channel, value: #rrggbb
locked	boolean	

Retrieve all presets

Payload

Key	Type	Description	Mandatory
procedure	string	com.dynamr.light_preset.**all**	Yes

Sample payload

```
{
  "procedure": "com.dynamr.light_preset.all"
}
```

Response

An array of `preset`.


```
{
  "args": [
    [
      {
        "name": "active",
        "colors": {
          "1": "#f660ab",
          "5": "#f660ab",
          "9": "#f660ab",
          "13": "#f660ab"
        },
        "locked": true,
        "id": 1
      },
      {
        "name": "bust",
        "colors": {
          "1": "#ff0000",
          "5": "#ff0000",
          "9": "#ff0000",
          "13": "#ff0000"
        },
        "locked": true,
        "id": 6
      }
    ]
  ]
}
```

Light

Status

Lights can be set by two way: a preset or a RGB color. The status endpoint gives the details.

Payload

Notice the "`.output`" in procedure.

Key	Type	Description	Mandatory
`procedure`	string	`com.dynamr.`**`output.light.status`**	Yes

Sample payload

```
{
  "procedure": "com.dynamr.output.light.status"
}
```

Response

When set by a preset

```
{
    "args": [
        {
            "type": "preset",
            "value": "round.active"
        }
    ]
}
```

When set by RGB

```
{
    "args": [
        {
            "type": "color",
            "value": [
                33,
                222,
                11
            ]
        }
    ]
}
```

Set the lights

By Preset

Payload

Key	Type	Description	Mandatory
`Procedure`	string	`com.dynamr.`**`light.preset`**	Yes
`args`	array<string> of 1 element	The `preset` name, cf presets.	Yes

Sample payload

```
{
  "procedure": "com.dynamr.output.light.preset",
  "args": ["round.speed"]
}
```

Response

```
{
    "args": [
        "round.speed"
    ]
}
```

By color

Payload

Key	Type	Description	Mandatory
`Procedure`	string	`com.dynamr.`**`light.color`**	Yes
`args`	array<string> of 1 element	#RRGGBB	Yes

Sample payload

```
{
  "procedure": "com.dynamr.output.light.color",
  "args": ["#885566"]
}
```

Response

```
{
    "args": [
        136,
        85,
        102
    ]
}
```

Sessions

Prefix is: “`com.dynamr.session.`”.

This is dedicated to tell the server the different states a session is:

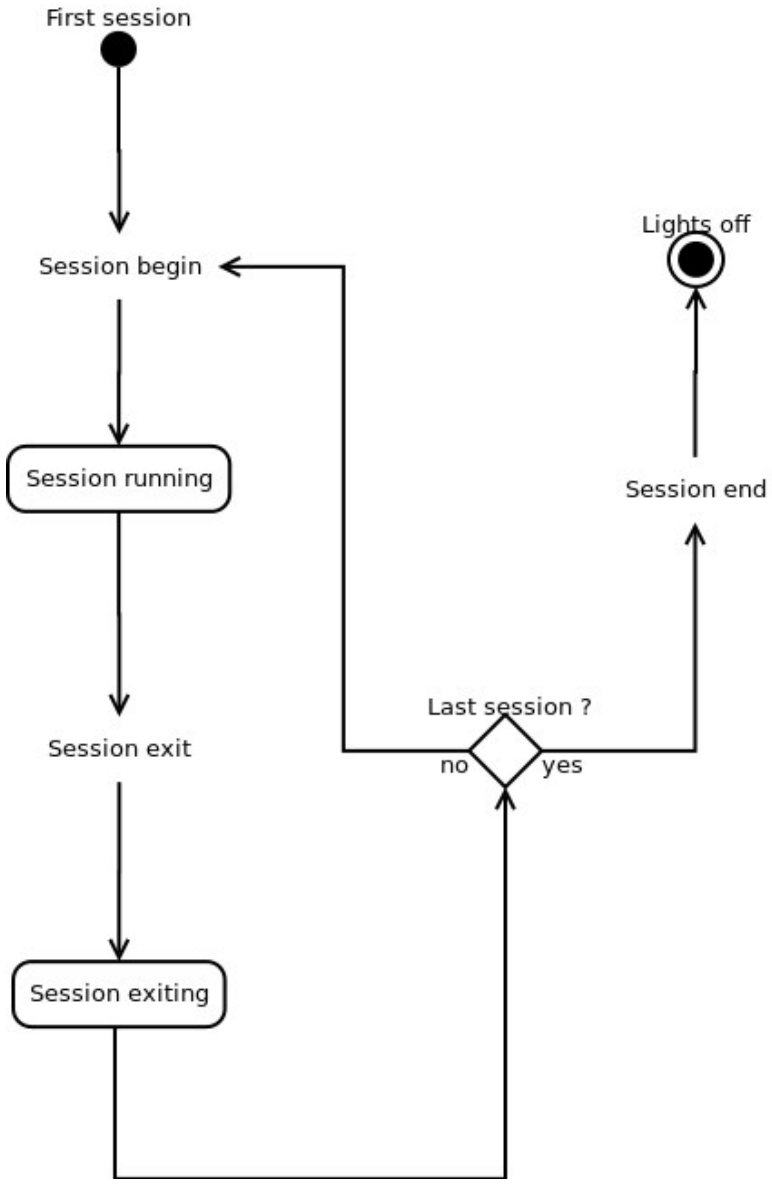
- `running`
- `exiting`

In a `running` state, lights will display the wanted preset.

In a `exiting` state, lights will be blinking.

System is able to switch in theses states without condition. It’s not server’s part of the job to know if the state change is allowed or not.

Normal workflow is:



Begin

Payload

Key	Type	Description	Mandatory
`procedure`	string	`com.dynamr.session.`**`begin`**	Yes
`args`	array<string> of 1 element	The `preset` name, cf presets.	Yes

Sample payload

```
{
  "procedure": "com.dynamr.session.begin",
  "args": ["speed"]
}
```

Response

The `preset` name.

```
{
  "args": [
    "speed"
  ]
}
```

Exit

Payload

Key	Type	Description	Mandatory
procedure	string	com.dynamr.session.**exit**	Yes

Sample payload

```
{
  "procedure": "com.dynamr.session.exit"
}
```

Response

true

```
{
  "args": [
    true
  ]
}
```

End

Payload

Key	Type	Description	Mandatory
procedure	string	com.dynamr.session.**end**	Yes

Sample payload

```
{
  "procedure": "com.dynamr.session.end"
}
```

Response

true

```
{
  "args": [
    true
  ]
}
```

Status

Payload

Key	Type	Description	Mandatory
`procedure`	string	`com.dynamr.session.`**`status`**	Yes

Sample payload

```
{
  "procedure": "com.dynamr.session.status"
}
```

Response

There is 2 cases:

1. The server is in `session` mode
2. The server is NOT in `session` mode

Server in session mode

An hashmap with the current `state` and `type` (`preset`).

`state` IS one of

- `running`
- `exiting`
- `stopped`

`type` is the type of the round (aka. `preset` name)

```
{
    "args": [
        {
            "state": "running",
            "type": "speed"
        }
    ]
}
```

Server NOT in session mode

If the `server` is NOT in `session` mode, `response` will be `null`.

```
{
  "args": [
    null
  ]
}
```

Speed round

Prefix is: “`com.dynamr.round_speed.`”.

You can control the life cycle of a round speed, ie. create a round, stop and bust it.

Note: As a bust must last as long as a the judge want, you need to “enter” & “exit” a bust.

Reload

Round is force-reloaded. It will stop the current mode if needed.

Payload

Key	Type	Description	Mandatory
`procedure`	string	`com.dynamr.round_speed.`**`reload`**	Yes

Sample payload

```
{
  "procedure": "com.dynamr.round_speed.reload"
}
```

Response

The round current configuration and status.

```
{
    "args": [
        {
            "configuration": {
                "countdown": 0,
                "bust": 5,
                "entry_fault": 5,
                "skip": 20,
                "page_count": 3,
                "moves_per_page": 3,
                "type": "speed"
            },
            "status": {
                "current_buster": null,
                "bust_count": 0,
                "skip_count": 0,
                "entry_fault": false
            }
        }
    ]
}
```

Busting

Bust enter

Payload

Key	Type	Description	Mandatory
procedure	string	com.dynamr.round_speed.**bust_enter**	Yes

Sample payload

```
{
  "procedure": "com.dynamr.round_speed.bust_enter"
}
```

Response

The round current status (with the caller as the current buster).

```
{
    "args": [
        {
            "current_buster": {
                "who": "handle",
                "name": null,
                "type": null
            },
            "bust_count": 2,
            "skip_count": 0,
            "entry_fault": false,
            "exit_last": 53.07606220245361
        }
    ]
}
```

Busts leave

Payload

Key	Type	Description	Mandatory
procedure	string	com.dynamr.round_speed.**bust_leave**	Yes

Sample payload

```
{
  "procedure": "com.dynamr.round_speed.bust_leave"
}
```

Response

The round current status.

```
{
    "args": [
        {
            "current_buster": null,
            "bust_count": 2,
            "skip_count": 0,
            "entry_fault": false,
            "exit_last": 53.07606220245361
        }
    ]
}
```

Stop

Payload

Key	Type	Description	Mandatory
`procedure`	string	`com.dynamr.round_speed.`**`stop`**	Yes

Sample payload

```
{
  "procedure": "com.dynamr.round_speed.stop"
}
```

Response

The round data.


```
{
    "args": [
        {
            "configuration": {
                "countdown": 0,
                "bust": 5,
                "entry_fault": 5,
                "skip": 20,
                "page_count": 3,
                "moves_per_page": 3,
                "type": "speed"
            },
            "times": {
                "begin": 956237862.0282004,
                "end": 956237931.4168363,
                "raw_original": 69.3886358737946,
                "raw_final": 79.3886358737946
            },
            "moves": [],
            "busts": [
                {
                    "buster": {
                        "who": "wm_1",
                        "name": "Bob",
                        "type": "Line judge"
                    },
                    "begin": 956237869.3857775,
                    "end": 956237877.41333
                },
                {
                    "buster": {
                        "who": "192.168.10.42",
                        "name": null,
                        "type":  null
                    },
                    "begin": 956237911.1516032,
                    "end": 956237922.0063996
                }
            ],
            "maybusts": [],
            "skips": [],
            "exits": [],
            "has_entry_fault": false,
            "metas": {
                "created": 956237000.7485876,
                "activated": 956237010.9040902,
                "stopped": 956237999.4168584
            },
            "id": 666
        }
    ]
}
```

Network

It uses the API.

This part allows you to define:

- the ODE (Open DMX Ethernet) endpoint
- the ethernet port configuration

ODE

URL	http://dynamr.local:5000**/api/network/ode**
Method	POST

Payload

It's the IP v4 represented as a string.

Sample payload

```
"192.168.20.254"
```

Response

Success

Body	*null*
Status	200

Error

Body	{
	"code": 400,
	"details": "foo does not appear to be a valid IPv4 address",
	"status": "error",
	"title": "Invalid data",
	"type": "api.network"
	}

Status	400

eth0

You have 2 options: use DHCP or static addressing.

URL	http://dynamr.local:5000**/api/network/eth0**
Method	POST

Payload

With DHPC:

```
{
  "dhcp": true
}
```

Static addressing:

```
{
  "dhcp": false,
  "address": "192.168.10.1",
  "netmask": "255.255.255.0"
}
```

all fields are mandatory.

Response

Success

Body	*null*
Status	200

Error

Body	{
	"code": 400,
	"details": "...",
	"status": "error",
	"title": "Invalid data",
	"type": "api.network"
	}

Status	400