WanRaptor Scripting API Documentation
For Version 2.0

East Coast Datacom

August 6, 2019

Contents

Chapter 1

Introduction

This manual contains the documentation for the REST API of WanRaptor, which is the same used by the WanRaptor Web Interface. Purpose of this documentation is to allow scripting of operations, as to make the WanRaptor more flexible to customer needs.

1.1 Endpoints

The REST endpoints available at the same management address that is used to access the Web Interface.
In this document, we will express the endpoints relatively to the management address, e.g. if management address is `192.168.1.120` and the endpoint path is `/api/auth/login`,then the HTTP request must be sent to `http://192.168.1.120/api/auth/login` Following REST best practices, endpoints are accessed through appropriate HTTP verbs (`GET`, `POST`, `PUT`, `DELETE`). Thus, in this document endpoints are presented as verb + path, e.g. `POST /apilogin`.

1.2 Design Principles

The REST API is designed according the following principles:

- URLs start with `/api/` to distinguish from Web Interface URLs
- Each URL locates a resource. As such,
  - IDs are specified as part of the path (e.g. `/api/resource/{resource_id}`).
  - GET parameters (e.g. `/api/resource?key=value`) are not used.
  - Call-specific parameters are specified through the request body.

1.3 Data Structures

The format of choice for data, in both requests and responses, is JSON. To explicit this, we suggest to set HTTP headers accordingly.

- In a request with a data payload, set `Content-Type` to `application/json`.
- As any request expects data within the response, set `Accept` to `application/json`.

In this document, we will present the JSON data structures via their schema, where properties are presented as "`<name>`": `<type>`. An optional field is indicated by a `?` following the name, i.e. "`<name>`"`?`: `<type>`.

1.4 Response Structure

To generalize the management of errors, all responses have a data payload to indicate the presence or absence of an error. This system does not replace the HTTP code system, instead it enhances it with application specific errors.

An exception to this are the Authentication endpoints, which in case of success do not follow this schema.

Schema

```
{
  ”error” : String,
  ”description”: String,
  ”content”: String”
}
```

Description of Fields

- `error`
  Error type. If `none`, the requests was successful. If `validation`, the input provided was not valid.
- `description`
  Human readable description of the error, if present.
- `content`
  If `error` is `none`, it contains the response data.
  Otherwise, it will describe validation errors.
  In all cases, it can be empty: `{}`.

In the rest of the document, we will refer to successful responses with empty `content` as *NoError*. For successful responses with non-empty `content`, we will refer to it directly as the response content.

Examples

An example of a validation error is shown in the code below. The error is related to validation of an emulation profile form. The error signals that the constant set for the delay on port1 is less than the minimum supported by that port, which is given by hardware limitations.

```
{
    ”error” : ”validation”,
    ”description” : ”Input is invalid”,
    ”content” : {
        ”delays.port1.constant” : [”min.notmet”]
    }
}
```

Chapter 2

Authentication

For most of the API, the request must provide authentication information to authorize the execution of the request. The system used is based on OAuth 2.0 specification.
This information is given in form of an `access_token` and a `token_type`, which must be provided via the `Authorization` header of the HTTP request. E.g.: `Authorization: <token_type> <access_token>`

2.1 POST /api/login

Used to authenticate and obtain the related tokens, needed for subsequent requests.
Note that these endpoints do not follow the same schema as the rest of the API. In case of success, `error` and `description` fields are not included. The response will have only the fields listed below.

Payload

```
{
  "username": String,
  "password": String
}
```

Response

```
{
  "username": String,
  "roles": String[],
  "token_type": String,
  "access_token": String,
  "expires_in": Integer,
  "refresh_token": String
}
```

- `username`
  The authenticated username.
- `roles`
  List of *roles* assigned to the user. E.g. `["ROLE_ADMIN"]`
- `token_type`
  Token type, which usually is set to `"Bearer"`.

- `access_token`
  Token to be attached to each following request as proof of authentication, using the `Authorization` header of the HTTP request.
  WARNING: If the header is not added to the request, or if the token is added incorrectly, requests will fail with error 401.
- `expires_in`
  Expiration time of the `access_token`, measured in seconds. Usually set to `3600`, i.e. 1 hour.
  WARNING: Using an expired `access_token` for requests will fail with error 401.
- `refresh_token`
  Used to refresh the `access_token` without repeating the login, with `POST /oauth/access_token`.

2.2 POST /oauth/access_token

Used to refresh an `access_token` before its expiration, and avoid a re-login.
Following the OAuth 2.0 specification, requests to this protocol must use the `x-www-form-urlencoded` encoding instead of JSON, i.e. the request header will contain
`Content-Type: application/x-www-form-urlencoded`

Payload

The body of the request will, consequently, be in the following format:

```
grant_type=refresh_token&refresh_token=<refresh_token>
```

`refresh_token`, to the right of the second equal sign, must be set with the token obtained from `POST /api/login`.

Response

Follows the same response schema as `POST /api/login`.

Chapter 3

Emulation Profiles

3.1 GET /api/emulations/

Obtains the list of IDs of all emulations profiles, as an array of ID values.
Requires no argument.

3.2 GET /api/emulations/{id}

Obtains the emulation profile with the given ID, as a single `EmulationProfile` object.
Requires no argument.

3.3 POST /api/emulations/

Creates a new emulation profile.
The payload must contain a single `EmulationProfile` object.
If the emulation was correctly created, the response contains the assigned `id` as follows

```
{
    ”id”: Integer
}
```

3.4 PUT /api/emulations/{id}

Edits the emulation with the given `id`.
The payload must contain a single `EmulationProfile` object with the updated values.
If the emulation was correctly updated, the response is a *NoError*.

3.5 DELETE /api/emulations/{id}

Delete the emulation with the given `id`.
If the emulation was correctly deleted, the response is a *NoError*.

3.6 DELETE /api/emulations/

Deletes all emulations currently present.

3.7 POST /api/emulations/{id}/start

Starts the emulation with the given `id`.
If the emulation was correctly started, the response is a *NoError*.

3.8 POST /api/emulations/{id}/stop

Stops the emulation with the given `id`.
If the emulation was correctly stopped, the response is a *NoError*.

3.9 Data structures

3.9.1 EmulationProfile

Schema

```
{
    "id"?: Integer,
    "name": String,
    "operationMode": String,
    "port1": String,
    "port2": String,
    "logEnabled": Boolean,
    "jumboFrames": Boolean,
    "runningState"?: String,
    "scheduled": Boolean,
    "scheduleSettings"?: ScheduleSettings,
    "delays": {
        "port1": Delay,
        "port2": Delay
    },
    "lossRateSettings": {
        "port1": LossRate,
        "port2": LossRate
    },
    "bandwidthSettings": {
        "port1": Bandwidth,
        "port2": Bandwidth
    },
    "reorderingSettings": {
        "port1": Reordering,
        "port2": Reordering
    }
}
```

Description of fields

- `id`
  Unique id of the emulation profile.
  This field must not be present in `POST /api/emulations/` and `PUT /api/emulations/{id}` requests.
- `name`
  Unique name of the emulation profile.
- `operationMode`
  Can be one of "`bridge`" or "`route`".
- `port1`
  Name of the first interface as seen in the Web Interface.
- `port2`
  Name of the second interface as seen in the Web Interface.

- `logEnabled`
  If true, a log will be created.

- `jumboFrames`
  If true, jumbo frames support will be enabled.

- `runningState`
  Running state of the emulation. Can be one of `running` or `stopped`.
  This field is server-generated, it must not be present in `POST /api/emulations/` and `PUT /api/emulations/{id}` requests.

- `scheduled`
  If true, the emulation profile is scheduled for automatic start.

- `scheduleSettings`
  Optional `ScheduleSettings` field, must be present only if `scheduled` is true.

- `delays`
  `Delay` settings for both ports.

- `lossRateSettings`
  `LossRate` settings for both ports.

- `bandwidthSettings`
  `Bandwidth` settings for both ports.

- `reorderingSettings`
  `Reordering` settings for both ports.

3.9.2 `ScheduleSettings`

Schema

```
{
    ”startDateTime”: DateTime,
    ”endDateTime”?: DateTime
}
```

Description of fields

- `startDateTime`
  `DateTime` of scheduled start.

- `endDateTime`
  `DateTime` of scheduled end. Optional field.

3.9.3 `DateTime`

Schema

```
{
    ”date”: dd/MM/yyyy,
    ”time”: HH:mm
}
```

Description of fields

- `date`
  Date expressed in day/month/year format.

- `time`
  Time expressed in hour:minute format.

3.9.4 `Delay`

Schema

```
{
    "type": String,
    "constant": Decimal,
    "minimum": Decimal,
    "maximum": Decimal,
    "average": Decimal
}
```

Description of fields

Aim of the data structure as a whole is to represent the probability distribution of the delay. Meaning of the fields depend on the value of `type`, as described below. All numeric fields are intended in milliseconds. For a given `type`, only cited values are considered, others are ignored.

- `constant`
  The delay is constant and set to the value of `constant`.

- `uniform`
  The delay is uniformly distributed between `minimum` and `maximum`.

- `exponential`
  The delay is exponential distributed with base value `minimum` and constant `average`.

- `inter_packet`
  Packets are delayed by `constant` and, additionally, the delay between two consecutive packets is uniformly distributed between `minimum` and `maximum`.

3.9.5 `LossRate`

Schema

```
{
    "type": String,
    "rate": Decimal
}
```

Description of fields

- `type`
  Can be either `plr` or `ber`, which stand respectively for Packet Loss Rate and Bit Error Rate.

- `rate`
  Error rate. If `type` is set to `plr`, it is between 0 and 100. If `type` is set to `ber`, it is between 0 and 1.

3.9.6 Bandwidth

Schema

```
{
    "bandwidth": Decimal
    "unit": String
}
```

Description of fields

- `bandwidth`
  Bandwidth value.
- `unit`
  Unit of measure for the bandwidth. Can be one of `gbps`, `mbps` or `kbps`.

3.9.7 Reordering

Schema

```
{
    "delay": Decimal
    "probability": Decimal
}
```

Description of fields

- `delay`
  Delay value in milliseconds.
- `probability`
  Probability of reordering, from 0 to 100.

3.9.8 Example EmulationProfile

```
{
    ”id”: 6,
    ”name”: ”profile_name ”,
    ”operationMode”: ”bridge”,
    ”port1”: ”enp3s0f0”,
    ”port2”: ”enp3s0f1”,
    ”logEnabled”: true,
    ”jumboFrames”: false,
    ”runningState”: ”stopped”,
    ”scheduled”: false,
    ”delays”: {
        ”port1”: {
            ”type”: ”constant”,
            ”constant”: 29.0
        },
        ”port2”: {
            ”type”: ”exponential”,
            ”minimum”: 1.0,
            ”average”: 5.0
        }
    },
    ”lossRateSettings”: {
        ”port1”: {
            ”rate”: 26.0,
            ”type”: ”plr”
        },
        ”port2”: {
            ”rate”: 1.0E-5,
            ”type”: ”ber”
        }
    },
    ”bandwidthSettings”: {
        ”port1”: {
            ”bandwidth”: 1.0,
            ”unit”: ”gbps”
        },
        ”port2”: {
            ”bandwidth”: 5.0,
            ”unit”: ”mbps”
        }
    },
    ”reorderingSettings”: {
        ”port1”: {
            ”delay”: 17.0,
            ”probability”: 10.0
        },
        ”port2”: {
            ”delay”: 0.0,
            ”probability”: 0.0
        }
    }
}
```

Chapter 4

Emulation Logs

4.1 GET /api/logs

Obtains the IDs of logs, as an array of log IDs.

4.2 GET /api/logs/active

Obtains the IDs of logs currently active, as an array of log IDs.

4.3 GET /api/logs/{id}

Obtains the log with the given ID, as a single `Log` object.

4.4 DELETE /api/logs/{id}

Deletes the log with the given `id`, if it is not active.
If the log was correctly deleted, the response is a *NoError*.

4.5 DELETE /api/logs/

Deletes all logs currently present which are not active.
If all non-active logs were correctly deleted, or no non-active log is present, the response is a *NoError*.

4.6 GET /api/logs/{id}/statistics

Obtains the list of statistics for the log with the given `id`, as an array of `Statistics` objects.

4.7 GET /api/logs/{id}/statistics/last

Obtains the last statistics for the log with the given `id`, as a single `Statistics` object.

4.8 POST /api/logs/{id}/reset

Resets all statistics counters for the log with the given `id`.
If the reset was correctly executed, the response is a *NoError*.

4.9 GET /api/logs/{id}/export

Exports the log data in csv formatted file.

4.10 Data structures

4.10.1 Log

Schema

```
{
    "id": Integer,
    "start": DateTime,
    "active": Boolean,
    "configuration": EmulationProfile
}
```

Description of fields

- `id`
  ID of log.
- `start`
  `DateTime` when the log started.
- `active`
  True if the log is currently active.
- `configuration`
  Snapshot of the `EmulationProfile` that is logged by this log.

4.10.2 Statistics

Schema

```
{
    "time": Integer,
    "producer1": StatisticsStructure,
    "producer2": StatisticsStructure,
    "consumer1": StatisticsStructure,
    "consumer2": StatisticsStructure
}
```

Description of fields

- `time`
  Time in seconds passed from log start when these statistics were sampled.
- `producer1`
  `StatisticsStructure` for `producer1`.
- `producer2`
  `StatisticsStructure` for `producer2`.
- `consumer1`
  `StatisticsStructure` for `consumer1`.
- `consumer2`
  `StatisticsStructure` for `consumer2`.

4.10.3 StatisticsStructure

Schema

```
{
    ”packets”: Integer,
    ”bytes”: Integer,
    ”dropPackets”: Integer,
    ”dropBytes”: Integer,
    ”reorderedPackets”: Integer,
    ”reorderedBytes”: Integer,
}
```

Description of fields

- `packets`
  Total number of packets processed.
- `bytes`
  Total number of bytes processed.
- `dropPackets`
  Total number of packets dropped.
- `dropBytes`
  Total number of bytes dropped.
- `reorderedPackets`
  Total number of packets reordered.
- `reorderedBytes`
  Total number of bytes reordered.

Chapter 5

Scripting Example

Being a HTTP interface, the API can be used from any language that can send/receive HTTP messages. In this chapter we provide an example of this, using Python 3.

The following example script authenticates into the device, lists the emulation profiles configured, picks one and runs it. The emulation is then edited while running, and after some time it is stopped. After the script has run, emulation data will be available to be visualized using the WanRaptor GUI, or processed with custom solutions based on the Emulation Logs APIs.

```
import requests
import json
import time

# This is the base url on which the GUI is reachable
baseUrl = "http://192.168.1.100:8080"

headers = {'content-type': 'application/json'}

# Request for login
loginCredentials = {'username': 'admin', 'password': 'WanRaptor'}

r = requests.post(
    url = baseUrl + "/api/login",
    data = json.dumps(loginCredentials),
    headers=headers)

if r.status_code == 200:
    resp = json.loads(r.text)

    # Save data from the login response
    token_type = resp['token_type']
    access_token = resp['access_token']
    refresh_token = resp['refresh_token']

    # Add Authorization header
    headers['Authorization'] = token_type + " " + access_token

# Request for list of emulation profile ids
r = requests.get(
    url = baseUrl + "/api/emulations",
    headers = headers)

if r.status_code == 200:
    ids = json.loads(r.text)
    print(ids)

# For following commands, suppose we select a specific emulation ID out of the list
id = 6

# Request for given emulation profile
r = requests.get(
    url = baseUrl + "/api/emulations/" + id,
    headers = headers)
```

```
if r.status_code == 200:
    profile = json.loads(r.text)
    print(profile)

# Request to start an emulation profile
r = requests.post(
    url = baseUrl + "/api/emulations/" + id + "/start",
    headers = headers)

resp = json.loads(r.text)
print(resp)

# The emulation is now running
time.sleep(5)

# You can change any property of the emulation profile, like you would on the GUI
# You can edit both when the emulation is stopped and when is running
profile['delays']['port1']['constant'] = 25

r = requests.put(
    url = baseUrl + "/api/emulations/" + id + "edit",
    headers = headers,
    data = json.dumps(profile))

resp = json.loads(r.text)
print(resp)

# Suppose half an hour has passed, it is good practice to refresh tokens early

# According to OAuth 2.0 specification, this request must be in x-www-form-urlencoded format
refreshCredentials = 'grant_type=refresh_token&refresh_token=' + refresh_token

r = requests.post(
    url = baseUrl + "/oauth/access_token",
    data = refreshCredentials,
    headers = {'content-type': 'application/x-www-form-urlencoded'})

if r.status_code == 200:
    resp = json.loads(r.text)

    # Update tokens from the refresh response
    token_type = resp['token_type']
    access_token = resp['access_token']
    refresh_token = resp['refresh_token']

    # Update Authorization header
    headers['Authorization'] = token_type + " " + access_token

# The emulation is still running
time.sleep(5)

# Request to stop an emulation profile
r = requests.post(
    url = baseUrl + "/api/emulations/" + id + "/stop",
    headers = headers)

resp = json.loads(r.text)
print(resp)

# The emulation is now stopped, and the script concludes.
# Emulation data is available on the GUI or through /api/logs/ endpoints
```