R&S®PRISMON
Audio/Video Content Monitoring and Multiviewer Solution
REST API Description rev.2.22

ROHDE&SCHWARZ
Make ideas real

Contents

Remote Control & Automation API

NOTICE

Only available with license R&S PRM-KXCORE (PRISMON SW-INST EXT CORE)

Besides the visual monitoring, the R&S PRISMON offers the possibility to allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations. This Representational state transfer (REST) or RESTful Web services are one way of providing interoperability between computer systems on the Internet. In a RESTful Web service, requests made to a resource's URI will elicit a response that may be in XML, HTML, JSON or some other defined format. The response may confirm that some alteration has been made to the stored resource, and it may provide hypertext links to other related resources or collections of resources. Using HTTP, as is most common, the kind of operations available include those predefined by the HTTP verbs `GET, POST, PUT, DELETE` and so on. By making use of a stateless protocol and standard operations, REST systems aim for fast performance, reliability, and the ability to grow, by re-using components that can be managed and updated without affecting the system as a whole, even while it is running.

1 Authorization (Access & Refresh Tokens)

PRISMONs running a version >= V2.06.0 support authentication using theOpenID Connect protocol. You can find all relevant OpenID Connect metadata on your PRISMON:

https://<PRISMON_IP>/auth/realms/prismon/.well-known/openid-configuration

Workflow overview

1. The client requests a refresh token using the PRISMON Web UI or via the specified HTTP-Request
2. The client sends the refresh token to the token endpoint
3. The server responds with new a new access token and a new refresh token
4. The access token can be used to request the desired resource

1.1 Request Refresh Token using the Web UI

Requesting a Refresh Token is only available to users in the admin group. Secure your token as you would secure your password. The token can be used to login to your PRISMON device.

Navigate to System & Hardware / User Management. Click Request Token and authenticate the monitoring account.

Figure 1-1: R&S PRISMON: Remote Access Token

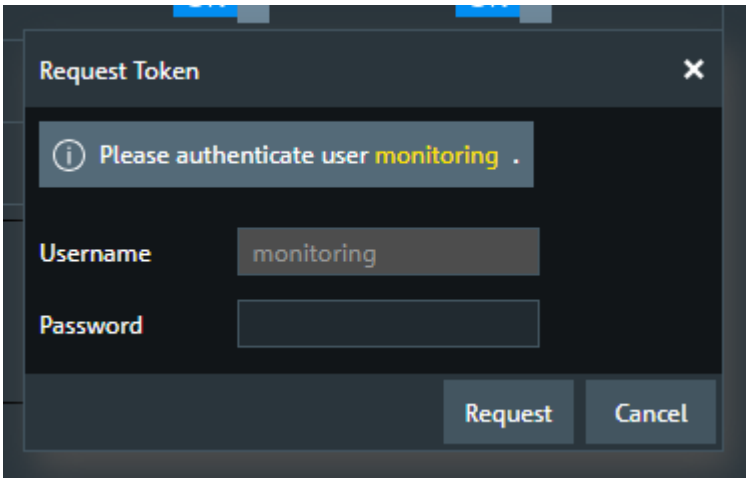

Figure 1-2: R&S PRISMON: Request Token

Figure 1-3: R&S PRISMON: Request Token Response

1.2 Request Access and Refresh Token without access to the Web UI

Your refresh token is valid indefinitely. If you want to integrate the Remote Control & Automation API into another application, you should use short lived tokens. Request those using scope = openid instead of scope = offline_token

You can directly request an Offline Remote Access Token using this HTTP-Request

Request:

```
curl --location --request POST \

'https://<PRISMON-IP>/auth/realms/prismon/ \

protocol/openid-connect/token' \

--header 'Content-Type: application/x-www-form-urlencoded' \

--data-urlencode 'client_id=webui' \

--data-urlencode 'username=monitoring' \

--data-urlencode 'password=yourpassword' \

--data-urlencode 'grant_type=password' \

--data-urlencode 'scope=offline_access'
```

Response:

```
{
    "access_token":"<your_access_token>",
    "expires_in":300,
    "refresh_expires_in":0,
    "refresh_token":"<your_refresh_token>",
    "token_type":"bearer",
```

```
    "not-before-policy":1584525611,
    "session_state":"<session_state_guid>",
    "scope":"email profile offline_access"
}
```

The response includes a short-lived access token, which can be used to request a desired API endpoint. Once the short-lived token expires you can use the refresh token to request a new short-lived access token.

1.3 Requesting an Access Token using your Refresh Token

Request:

```
curl --location --request POST \
'https://<PRISMON-IP>/auth/realms/prismon/ \
protocol/openid-connect/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'client_id=webui' \
--data-urlencode 'refresh_token=<your access token>'\
--data-urlencode 'grant_type=refresh_token'
```

Response:

```
{
    "access_token":"<your_access_token>",
    "expires_in":300,
    "refresh_expires_in":0,
    "refresh_token":"<your_refresh_token>",
    "token_type":"bearer",
    "not-before-policy":1584525611,
    "session_state":"<session_state_guid>",
    "scope":"email profile offline_access"
}
```

This returns a new short-lived access token, which can be used to request the desired API endpoint (e.g. /-/spu.inputs/02/DESCRIPTION)

Request:

```
curl --location --request GET \

'https://<PRISMON-IP>/-/spu.inputs/02/DESCRIPTION' \

--header 'Content-Type: application/json; charset:utf-8' \

--header 'Authorization: Bearer <your access token>'
```

Example Response:

```
“Source Singal 02”
```

1.4 Automatic Token Handling

Integrate your API-Requests using a library supporting OpenID Connect

(You can find certified openid connect libraries on the openid webpage: https://openid.net/developers/certified/).

You can find all relevant OpenID connect metadata on your PRISMON:

```
https://<prismon-ip>/auth/realms/prismon/.well-known/openid-configuration
```

2 Service Control Web-API

2.1 Errors

If an error occurs while executing Service-Control the response-code will be 400 and the output of stderr will be contained in the response-body.

POST-requests

- If you issue a POST-request the return value will be a representation of Service-Control's output. This is in most cases an empty string. If you set multiple values at once you will get all of Service-Control's responses in the reply.
- When setting a value an action can be added to the URL as a query parameter. `(e.g. POST http://<ip>/-/spu.inputs/01/DESCRIPTION?restart.)` This action will be performed once for every configured service after all configurations are applied.

2.2 Performance

Simple-Mode

Machine-to-machine communication should mostly use Simple-Mode because this produces almost no overhead. Every request leads to exactly one call to Service Control.

Recursive-Mode

In Recursive-Mode (as the name suggests) one API-call results in multiple calls to Service-Control. Depending on the request's depth, this can take up a lot of processing time. It is advised to use this for requests that are only issued once in a while to get or set a large number of configurations at once.

Pattern-Mode

The Pattern-Mode is the least performant of all. It not only underlies the same restrictions as the Recursive-Mode but also must evaluate which parts of the entire configuration are matching the pattern.

Use cases where this can still be a good choice might be human-interaction or in cases where machine-to-machine communication needs only a limited amount of configurations to reset them at a later point.

Do not use this for long-polling or similar cases!

2.2.1 Simple Mode

Simple-Mode is the default. If you do not anything else this is what the API will behave like.

GET

The simple GET is designed to behave like a regular REST-API `GET`.

Example:

```
GET http://192.168.0.100/-/spu.inputs
=> ['01', '02', '03' ...]
GET http://192.168.0.100/-/spu.inputs/01
=> ['DESCRIPTION', 'SERVICE_STATUS', ...]
GET http://192.168.0.100/-/spu.inputs/01/DESCRIPTION
=> "HLS_Decoder_1"
```

POST

To set a value there is only one way in simple-mode. You must issue a POST for the exact value you want to change with the new value as request-body.

NOTICE

The http-response will always contain Service-Control's response which is in almost all cases an empty string.

Example:

```
POST http://192.168.0.100/-/spu.inputs/01/DESCRIPTION (Body: "My First Input")
=> ""
```

There is a special POST command (simple type) to switch a viewer layout:

```
POST http://<ip>/-/spu.bmm/<viewer instance>/LAYOUT?reload (Body:"<selected layout>")
```

Example:

```
POST http://192.168.0.100/-/spu.bmm/01/LAYOUT?reload (Body:"2")
```

There is a special POST command (simple type) to switch the description tile text of a viewer layout:

```
POST http://<ip>/-/spu.bmm/<viewer
instance>/DESCRIPTIONTILE_TEXT_<descriptionTileId>?reload
(Body:"<text>")
```

Where "descriptionTileId" should be between 1 and 10.

Example:

```
POST http://192.168.0.100/-
/spu.bmm/01/DESCRIPTIONTILE_TEXT_1?reload (Body:"new text")
```

2.2.2 Recursive Mode

Recursive-Mode is designed to give an overview over a certain section with one request.

You are using Recursive-Mode if the request-URL starts with `/r/`.

GET

Depending on the depth of your search this request can take some processing time.

For example: It is not advised to use the `/r/`-Route within a long-polling context.

The recursive GET will always answer with a JSON-object that represents the entire resource from the root. Meaning that even if you only request one single value the response will contain the service-type and service-index this very value belongs to.

Example:

```
GET http://192.168.0.100/-/r/spu.inputs/01/DESCRIPTION
=> { "spu.inputs": { "01": { "DESCRIPTION":
"HLS_Decoder_1" } } }
GET http://192.168.0.100/-/r/spu.inputs/01/
=> { "spu.inputs: { "01": { "DESCRIPTION":
"HLS_Decoder_1", "SERVICE_STATUS": "0", ... }, }
GET http://192.168.0.100/-/r/spu.inputs
=> { "spu.inputs: { "01": { "DESCRIPTION":
"HLS_Decoder_1", "SERVICE_STATUS": "0", ... }, "02":
{ ... }, ... } }
GET http://192.168.0.100/-/r/
=> { "spu.inputs: { "01": { "DESCRIPTION":
"HLS_Decoder_1", "SERVICE_STATUS": "0", ... }, "02":
```

```
{ ... }, ... }, "spu.sources": { ... }, ... }
```

POST

Hence the recursive `GET` always responds with an object representing the entire resource, there is only one route for setting values recursively: `/r/`, the root itself.

NOTICE

- Configurations that are not in the request-body will not be touched.
- The http-response will be the exact same object you handed in but the values will be exchanged with Service-Control's output.
- In case of an error a partial configuration is possible!

2.2.3 Pattern Mode

GET

To use Pattern-Mode prefix the URL with `/p/`.

Responses will be formatted in the same way Recursive-Mode's responses are formatted. Therefore Pattern-Mode's responses can be used in Recursive-Mode POSTrequests.

NOTICE

Pattern-Mode does not require a separation of services and indexes via a / but doing so may result in server-side optimizations that lead to a better performance.

Example:

Get descriptions of everything that has a description

```
GET http://<IP>/-/p/*/DESCRIPTION
```

Get all configurations that are related to SDI

```
GET http://<IP>/-/p/*SDI*
```

POST

Issuing a POST to a Pattern-URL does not alter any configurations.

This is convenient if all you want to do is perform an action like starting or stop services.

The response's-body will be a JSON-object similar the one from a Recursive-Mode POST.

Thereby you can identify what services the server has performed the action for.

Example:

Stopping sources 10 to 19

```
POST http://192.168.0.100/-/p/spu.sources/1*?stop
=> { "spu.sources": { "10": "", "11": "", "12": "", "13":
"", "14": "", "15": "", "16": "", "17": "", "18": "", "19":
"" } }
```

2.3 Routes to access media data

R&S PRISMON REMOTE CONTROL & AUTOMATION API has additional routes to provide media data such as (preview) images and video streaming.

There are 3 types of routes as follow.

Example:

R&S PRISMON's video wall image, where `<INPUT>` is the video wall's ID

```
GET https://<IP>/webui/image/view/<INPUT>/video/1
```

Image of a service's videodecoder, where `<INPUT>` is the service number

```
GET https://<IP>/webui/image/service/<INPUT>/video/1
```

Image of one of the service's audio tracks as an audio bar, where <INPUT> is the service number and `<INPUT_AUDIO>` is the audio track

```
GET https://<IP>/webui/image/service/<INPUT>/audio/<INPUT_AUDIO>
```

NOTICE

<INPUT> is expected to be two digits. For inputs 1-9 it should be prefixed by zero i.e. '01' or '09'.

<INPUT_AUDIO> does not need to be padded to two digits.

2.4 Additional REST-API trees

```
GET https://<IP>/api/v1/
```