Mbed Device Server Web Interfaces 2.3

Table of Contents

Introduction

Web applications interact with the Device Server using a set of RESTful Web interfaces over HTTP. These interfaces provide administration, lookup and eventing services.

The Device Server Data Model

The Device Server data model is divided into domains, groups, end-points and resources. Resources can be perceived as individual URI paths (e.g. /path, /longer/path) and as such there can be several resources assigned for a single end-point. Resources are typically used to provide access to sensors, actuators and configuration parameters on a M2M device. An end-point is the web server software running on a device, and it belongs to some domain. In short, domain contains end-points and end-points contain resources and there can be several domains configured in the Device Server.

Device Server allows end-points and resources to be associated with semantic naming, and during registration naming meta-data can be associated with them. An end-point includes a host name, which uniquely identifies the end-point in a domain. An end-point can also be associated with a node type, for example 'MotionDetector'. Finally resources can be associated with a resource type (e.g. 'LightSensor'), an interface description and other meta-data about the resource such as its content-types and observability.

Device Server endpoints can be grouped into logical groups. Each endpoint can belong to several groups.

Authentication

The security model used in Device Server is domain level based. Client application are assign to one domain and must provide credentials on each request. There are two ways to configure credentials, using local configuration file or using third party authenticator.

Internal with basic authentication

Credential configuration file, `conf/credentials.properties` contains initial accounts that are loaded into cache.

Authorization is done with HTTP basic authentication and every request must include HTTP authorization header.

```
Authorization: Basic {base64encode(username:password)}
```

Every request contains domain name in first path segment:

```
/{domain-name}/...
```

Third party authentication server

When using third party authentication server, you need to configure its address in device server configuration file.

Authorization is done with any authentication type that is supported by authentication server. For each request, devices server verifies it with authentication server.

When client makes request as below:

```
Authorization: Token {token}
```

Then device server make a lookup to authentication server:

```
GET /auth/{token}

HTTP/1.1 200 OK
Content-Type: application/json
{ "userId": "app1", "domain": "example.com" }
```

If authentication server responses with any error or response body can not be parsed, then device server denies access for such request. Authenticator server must return at least `domain` part, `userId` is optional but recomended.

For performance reason, device server caches locally authorization responses, for short period of time.

Endpoint directory lookups

The directory lookup interface provides the possibility to browse and filter through resource directory of Device Server.

List all endpoints.

URI	`/{domain}/endpoints?type={endpoint-type}&stale={true|false}`
Method	GET
Query parameters	type - (optional) filters endpoints by endpoint-type stale - (optional, default: false) if true then result list will contain stale endpoints
Response	200 - Successful response with endpoints list
Acceptable content-types	application/json application/link-format

Endpoint list json structure

```
[
  { "name": "{endpoint name}",
    "type": "{endpoint type}",
    "status": "{endpoint status: ACTIVE|STALE}"
    "q": "{queue mode: true|false (default: false)"}
]
```

Example:

```
GET /example.com/endpoints
Accept: application/json

HTTP/1.1 200 OK
Content-Type: application/json

[
  { "name":"node-001", "type":"Light", "status":"ACTIVE"},
  { "name":"node-002", "type":"Light", "status":"STALE"},
  { "name":"node-003", "type":"QueueModeNode", "status":"ACTIVE",
 "q": true}
]
```

List endpoint's resources meta-information

URI	`/{domain}/endpoints/{endpoint-name}`
Method	GET
Responses	200 - OK 404 - endpoint not found
Acceptable content types	application/json application/link-format

Example:

```
GET /example.com/endpoints/node-001
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  { "uri":"/dev/temp", "rt":"ucum:C", "obs":"true", "type":"text/
plain"},
  { "uri":"/dev/illu", "obs":"false", "type":"text/plain"}
]
```

Endpoint's resource representation

This interface provides access to endpoint's resource representation.

URI	`/{domain}/endpoints/{endpoint-name}/{resource-path}?sync={true|false}&cacheOnly={true|false}&pri={dscp}&noResp={true|false}`
Method	GET | PUT | POST | DELETE
Query parameters	`sync` - (optional, default: false) indicate whether this request is synchronous or asynchronous `cacheOnly` - (optional, default: false) if true then response will come only from cache `pri` - (optional, default: 0) priority message, adds traffic-class for outgoing IPv6 message (only UDP). Accepted values are `AF11`, `AF12`, `AF13`, `AF21`, `AF22`, `AF23`, `AF31`, `AF32`, `AF33`, `AF41`, `AF42`, `AF43`, `VA`, `EF`, `CS0`, `CS1`, `CS2`, `CS3`, `CS4`, `CS5`, `CS6`, `CS7` and `DF`. Numeric values 0-7 are interpreted as matching to the corresponding CS value. `noResp` - (optional, default: false) if true then no waiting for response and no response is expected, creates CoAP Non-Confirmable requests
Responses	200 - Successful GET, PUT, DELETE operation 201 - Successful POST operation 202 - Accepted, asynchronous response id (only when sync=false) 205 - no cache available for resource 404 - Requested endpoint's resource not found 408 - Endpoint did not respond, timeout (only when sync=true) 409 - Conflict, endpoint is in queue mode and synchronous request can not be made (only when `sync=true`) or if `noResp=true` then means that request no supported 410 - Gone, endpoint not found 429 - Can not make request at the moment, already ongoing other request for this endpoint
Acceptable content types	application/json (for async response id) */* (depends on endpoint's resource)

Asynchronous: Activated when `sync` parameter is missing or false. Endpoint's response will arrive in notification channel. HTTP request will return immediately with asyn-response-id, that is used to match response. For GET requests, if representation is available in cache, then it will return it.

Example:

```
GET /example.com/endpoints/node-001/dev/temp

HTTP/1.1 202 Accepted
Content-Type: application/json
```

```
{"async-response-id": "23471#node-001@test.domain.com/path1"}
```

Synchronous (not recommended): Activated when `sync` parameter is true. Request will hold until response comes or timeout. Requests for endpoint in queue mode is not supported.

Example:

```
GET /example.com/endpoints/node-001/dev/temp?sync=true

HTTP/1.1 200 OK
Content-Type: text/plain
Cache-Control: max-age=100

24.5C
```

No response (NON): Indicated with parameter `noResp=true` and as a result, Device Server makes CoAP Non confirmable requests. REST response is with status code `204 No Content`. If underlying protocol does not support it (HTTP) or endpoint is registered in queue mode then response is with status code `409 Conflict`. Note that this type of requests are not guaranteed.

Example:

```
GET /example.com/endpoints/node-001/dev/temp?noResp=true

HTTP/1.1 204 No Content
```

Notifications

The Device Server eventing model consists of observable resources that enables end-points to deliver updated resource content, periodically or with more sophisticated solution dependent logic. Applications are able to subscribe to every individual resource or can set a pre-subscription data in order to receive notification update.

Subscribe to resource

URI	`/{domain}/subscriptions/{endpoint-name}/{resource-path}?sync={true|false}`
Method	PUT
Query parameters	sync - (optional, default: false) indicate whether this subscription request is synchronous or asynchronous
Response	200 - successfully subscribed 202 - Accepted, asynchronous response id (only when sync=false) 404 - endpoint or endpoints resource not found 412 - can not make subscription for non observable resource 429 - Can not make subscription request at the moment due to already ongoing other request for this endpoint 502 - subscription failed 504 - subscription could not be established due to timeout from endpoint

Asynchronous: Activated when 'sync' parameter is missing or false. Subscription response will arrive in notification channel. HTTP request will return immediately with asyn-response-id, that is used to match response. If subscription can be established immediately, without contacting endpoint, then this call returns subscription result.

Example:

```
PUT /example.com/endpoints/node-001/dev/temp

HTTP/1.1 202 Accepted
Content-Type: application/json

{"async-response-id": "5734979#node-001@test.domain.com/path1"}
```

Remove subscription

URI	`/{domain}/subscriptions/{endpoint-name}/{resource-path}`
Method	DELETE
Response	204 - successfully removed subscription 404 - endpoint or endpoints resource not found

Remove all subscriptions

URI	`/{domain}/subscriptions`
Method	DELETE
Response	200 - successfully removed subscriptions

Read subscription status

URI	`/{domain}/subscriptions/{endpoint-name}/{resource-path}`
Method	GET
Response	200 - resource is subscribed 404 - resource is not subscribed

Read endpoints subscriptions

URI	`/{domain}/subscriptions/{endpoint-name}`
Method	GET
Response	200 - list subscribed resources 404 - endpoint not found
Acceptable content type	text/uri-list

Example:

```
GET /example.com/subscriptions/node-001

HTTP/1.1 200 OK
Content-Type: text/uri-list

/example.com/subscriptions/node-001/dev/temp
/example.com/subscriptions/node-001/dev/power
```

Remove endpoints subscriptions

URI	`/{domain}/subscriptions/{endpoint-name}`
Method	DELETE
Response	204 - successfully removed 404 - endpoint not found

Set pre-subscription data

Pre-subscription means that application sets a pattern and will be subscribed automatically to all endpoint's all resources that matches the pattern. Pattern may include the endpoint type, a list of resources or an expressions with an `*` character at the end. Pre-subscription concerns of all the endpoints that are registered already, and all endpoints that will be registering in the future. Changing the pre-subscription data removes all previously existing subscriptions.

Example

```
PUT /{domain}/subscriptions
Content-Type: application/json

[
   {
     endpoint-type: "Light",
     resource-path: ["/sen/*"]
   },
   {
     endpoint-type: "Sensor"
   },
   {
     resource-path: ["/dev/temp","/dev/hum"]
   }
]

-->
HTTP/1.1 200 OK
```

Get pre-subscription data

Pre-subscription data can be retrieved by a GET operation. Server returns with the same JSON structure as described above. If there is no pre-subscribed resources then it returns with an empty array.

`GET /{domain}/subscriptions`

Receiving notifications

Notifications can be delivered in two different mechanism. The 'Long polling' or using subscription server where Device Server actively sends the notifications. Notifications are delivered in JSON format.

Client's push url

Notifications are delivered as PUT message to HTTP server defined by client with subscription server message. Following REST request (subscription) is needed by a client:

`PUT /{domain}/notification/push-url`

Entity of a message must contain URL location where client listens for incoming notifications. If special host name is used: `REMOTE_HOST`, then Device Server will replace it with source IP address.

Example:

```
PUT /example.com/notification/push-url
Content-length: 30

http://192.168.5.5:8089/events

HTTP/1.1 200 OK
```

`GET /{domain}/notification/push-url`
Returns client's url

To delete push url

URI	`/{domain}/notification/push-url`
Method	DELETE
Response	204 - successfully removed 404 - push url does not exist

This deletes push url AND removes all subscriptions

Long polling for notifications

Notifications are delivered through HTTP long-poll requests. The HTTP request is kept open until an event notification or batch of event notifications are delivered to the client or the request timeouts. In both cases the client should open a new polling connection after the previous one closes. Long polling connections are handled on a per domain basis so each open connection delivers notifications for a single domain. The interface for receiving event notification has a URL of the form:

`GET /{domain}/notification/pull`

Notification data structure

Notification message contains server event types.

Notification type	Description
notifications	Contains resource notifications
registrations	List of endpoints that have registered (with resources)
reg-updates	List of endpoints that have updated registration
de-registrations	List of endpoints that are removed in controlled manner
registrations-expired	List of endpoints that are removed because the registration has expired
async-responses	Responses to asynchronous proxy request

Notification message is delivered in JSON format as below:

```
{
   "notifications":[
     {
        "ep":"{endpoint-name}",
        "path":"{uri-path}",
        "ct":"{content-type}",
        "payload":"{base64-encoded-payload}",
        "timestamp":"{timestamp}"
        "max-age":"{max-age}"
     }
   ],

   "registrations":[
     {
       "ep": "{endpoint-name}",
       "ept": "{endpoint-type}",
       "q": "{queue-mode, default: false}",
       "resources": [ {
```

```
          "path": "{uri-path}",
          "if": "{interface-description}",
          "rf": "{resource-type}"
          "ct": "{content-type}",
          "obs": "{is-observable (true|false) }"
        } ]
      }
    ],

    "reg-updates":[
      {
        "ep": "{endpoint-name}"
        "ept": "{endpoint-type}"
        "q": "{queue-mode, default: false}",
        "resources": [ {
          "path": "{uri-path}",
          "if": "{interface-description}",
          "rf": "{resource-type}"
          "ct": "{content-type}",
          "obs": "{is-observable (true|false) }"
        } ]
      }
    ],

    "de-registrations":[
      {
        "{endpoint-name}",
        "{endpoint-name2}"
      }
    ],

    "registrations-expired":[
      {
        "{endpoint-name}",
        "{endpoint-name2}"
      }
    ],

    "async-responses":[
      {
         "id": "{async-response-id}",
         "status":  {http-status-code},
         "error":  {error-message},
         "ct":  "{content-type}",
         "max-age": {max-age},
         "payload":  "{base64-encoded-payload}"
      }
    ]
}
```

Groups

Putting and getting a group

First we create a minimal group by putting the json representation of it to the group's URI.

```
PUT http://localhost:8080/domain/groups/group0 HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=
Content-type: application/json

{"endpoints":[],"subGroups":[]}

HTTP/1.1 204 No Content
Date: Fri, 10 Oct 2014 11:02:26 GMT
```

```
Server: DeviceServer/DEVELOPMENT
```

Now we can get the json representation of the group by getting from the same URI.

```
GET http://localhost:8080/domain/groups/group0 HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=
Accept: application/json


HTTP/1.1 200 OK
Date: Fri, 10 Oct 2014 11:02:26 GMT
Content-Length: 31
Content-Type: application/json
Server: DeviceServer/DEVELOPMENT
{"subGroups":[],"endpoints":[]}
```

Trying to get a nonexistent group causes a 404 response.

```
GET http://localhost:8080/domain/groups/nonexistent HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=
Accept: application/json


HTTP/1.1 404 Not Found
Date: Fri, 10 Oct 2014 11:02:26 GMT
Content-Length: 0
Server: DeviceServer/DEVELOPMENT
```

We can add a description to the group by putting a modified json representation of it.

```
PUT http://localhost:8080/domain/groups/group0 HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=
Content-type: application/json

{"description":"a group","endpoints":[],"subGroups":[]}

HTTP/1.1 204 No Content
Date: Fri, 10 Oct 2014 11:02:26 GMT
Server: DeviceServer/DEVELOPMENT
```

Subgroups

Adding subgroups

Assuming we have created a groups called 'group0sub0' and 'group0sub1', we can now make them subgroups of 'group0' by sending a modified group0 that the subgroup names in the subGroups field.

```
PUT http://localhost:8080/domain/groups/group0 HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=
Content-type: application/json

{"description":"a group","endpoints":[],"subGroups":["group0sub0"
,"group0sub1"]}

HTTP/1.1 204 No Content
Date: Fri, 10 Oct 2014 11:02:26 GMT
Server: DeviceServer/DEVELOPMENT
```

A nonexistent group cannot be added as a subgroup.

```
PUT http://localhost:8080/domain/groups/group0 HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=
Content-type: application/json

{"description":"a group","endpoints":[],"subGroups":["group0sub0"
,"group0sub1","nonexistent"]}

HTTP/1.1 400 Bad Request
Date: Fri, 10 Oct 2014 11:02:26 GMT
Content-Length: 0
Server: DeviceServer/DEVELOPMENT
```

Removing subgroups

We can remove a subgroup by sending a modified version of the parent that no more contains a reference to the subgroup to remove.

```
PUT http://localhost:8080/domain/groups/group0 HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=
Content-type: application/json

{"description":"a group","endpoints":[],"subGroups":["group0sub0"
]}

HTTP/1.1 204 No Content
Date: Fri, 10 Oct 2014 11:02:26 GMT
Server: DeviceServer/DEVELOPMENT
```

Endpoint members

Adding and removing endpoint members works like subgroups.

First we add endpoints 'ep0' and 'ep1' as members of 'group0'.

```
PUT http://localhost:8080/domain/groups/group0 HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=
Content-type: application/json

{"description":"a group","endpoints":["ep0","ep1"],"subGroups":[
"group0sub0"]}

HTTP/1.1 204 No Content
Date: Fri, 10 Oct 2014 11:02:26 GMT
Server: DeviceServer/DEVELOPMENT
```

Then we remove 'ep1'.

```
PUT http://localhost:8080/domain/groups/group0 HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=
Content-type: application/json

{"description":"a group","endpoints":["ep0"],"subGroups":[
"group0sub0"]}

HTTP/1.1 204 No Content
Date: Fri, 10 Oct 2014 11:02:26 GMT
Server: DeviceServer/DEVELOPMENT
```

A nonexistent endpoint cannot be added as a member.

```
PUT http://localhost:8080/domain/groups/group0 HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=
Content-type: application/json

{"description":"a group","endpoints":["ep0","nonexistent"],
"subGroups":["group0sub0"]}

HTTP/1.1 400 Bad Request
Date: Fri, 10 Oct 2014 11:02:26 GMT
Content-Length: 0
Server: DeviceServer/DEVELOPMENT
```

Listing groups

We can get a list of either all groups or only root groups (groups that are not a child group of another group).

(Remember, 'group0sub1' is now a root group since we removed it from 'group0'.)

```
GET http://localhost:8080/domain/groups HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=
Accept: application/json


HTTP/1.1 200 OK
Date: Fri, 10 Oct 2014 11:02:26 GMT
Content-Length: 23
Content-Type: application/json
Server: DeviceServer/DEVELOPMENT
["group0sub1","group0"]
```

```
GET http://localhost:8080/domain/groups?all=true HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=
Accept: application/json


HTTP/1.1 200 OK
Date: Fri, 10 Oct 2014 11:02:26 GMT
Content-Length: 36
Content-Type: application/json
Server: DeviceServer/DEVELOPMENT
["group0sub1","group0sub0","group0"]
```

Deleting a group

Deleting a group also deletes its subgroups. We now delete 'group0' and its subgroup 'group0sub0'.

```
DELETE http://localhost:8080/domain/groups/group0 HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=


HTTP/1.1 204 No Content
Date: Fri, 10 Oct 2014 11:02:26 GMT
Server: DeviceServer/DEVELOPMENT
```

Now 'group0sub1' is the only existing group.

```
GET http://localhost:8080/domain/groups?all=true HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=
Accept: application/json
```

```
HTTP/1.1 200 OK
Date: Fri, 10 Oct 2014 11:02:26 GMT
Content-Length: 14
Content-Type: application/json
Server: DeviceServer/DEVELOPMENT
["group0sub1"]
```

Trying to delete a nonexistent group causes a 404 response.

```
DELETE http://localhost:8080/domain/groups/nonexistent HTTP/1.1
Authorization: Basic dXNlcjpzZWNyZXQ=


HTTP/1.1 404 Not Found
Date: Fri, 10 Oct 2014 11:02:26 GMT
Content-Length: 0
Server: DeviceServer/DEVELOPMENT
```

Device Server Administration

Administration can be handled with HTML user interface (admin-ui) or REST API (admin-api). Those interfaces can be access by default on https://localhost:8081, if not changed in device-server.properties.

Admin web console

- **Status** - Shows connectors information such like protocol, port, warnings.
- **Endpoints** - Shows endpoints with filtering. Enables removing endpoint.
- **Security** - Manages eDTLS credentials.
- **Clients** - Manages client application credentials.

Admin API

This chapter describes all administration operations that can be done with REST API.

Directory

Get all domains:

URI	`/admin-api/rd`
Method	GET
Response	200 - List of all domains

Create a new domain:

URI	`/admin-api/rd`
Method	PUT
Response	204 - Success 400 - Invalid domain name 400 - Domain already exists

Delete a domain. Domain to be deleted must have no endpoints.

URI	`/admin-api/rd`
Method	DELETE
Response	204 - Success 400 - Domain already has endpoints 404 - Domain not found

`GET /admin-api/rd/{domain}?name={endpoint-name}&status={ACTIVE|STALE}&type={endpoint-type}&maxEntries={max}`
Filter endpoints by name, type or status.

`DELETE /admin-api/rd/{domain}/{endpoint-name}`
Removes endpoint from Device Server.

Security eDTLS PSK credentials

`GET /admin-api/security/credentials`
List of all credentials, contains originalKey (to be used in UI when editing actual key), key (Hex), secret (Hex), name, authorization (bool), network (bool), registration (bool). Note that auth and nw parameters are used for PANA server as described in M2M interfaces document, in Authentication interface chapter. If registration and authorization parameter is set to false, then endpoint eDTLS registration with given key-id will be rejected.

Example return value:

```
 [
  {"originalKey":"00","key":"00","auth":true,"nw":true,"reg":true
,"name":"name","secret":"07d0"},
  {"originalKey":"01","key":"01","auth":true,"nw":true,"reg":true
,"name":"name","secret":"07cf"},
  {"originalKey":"02","key":"02","auth":true,"nw":true,"reg":true
,"name":"name","secret":"07ce"},
  {"originalKey":"03","key":"03","auth":true,"nw":true,"reg":true
,"name":"name","secret":"07cd"}
 ]
```

`GET /admin-api/security/credentials?key={key-id (hex)}`
Filters credentials by key-id

`GET /admin-api/security/credentials?page={x}&rows={y}`
Get supports paging with default jqgrid syntax. When using paging the return value contains also page information.

Example:

```
 {
   "total":10,"page":1,"records":100,
   "rows":[
     {"originalKey":"00","key":"00","auth":true,"nw":true,"reg":
true,"name":"name","secret":"07d0"}
   ]
 }
```

Where total = total number of pages, page = current page number, records = total number of records.

`GET /admin-api/security/credentials?sidx={x}&sord={y}`
Get supports sorting with default jqgrid syntax. This can be used with paging. Sidx = field to be sort with ('originalKey', 'name', 'key', 'secret'), sord = order ('asc', 'desc')

`POST /admin-api/security/credentials`
Create new credential, fields are submited as form parameters: key, secret, auth, nw, reg, name.

`PUT|DELETE /admin-api/security/credentials/{key-id}`
Updates/removes credential with provided key-id. When updating data, fields are submited as form parameters: key, secret, auth, nw, reg, name. Key-id in path MUST match the original key value.

Security eDTLS Certificates

`GET /admin-api/security/certificate/device-server`
Returns human readable server public certificate information

`GET /admin-api/security/certificate/root`
Returns human readable trusted CA public certificate information

`Delete /admin-api/security/certificate`
Deletes all certificate information

`POST /admin-api/security/certificate`
Sets certificate information. Data format is the plain text of base64 output of .cer and .key data created and exported using Device Server Key Tool. Input data must contain trusted CA public certificate (default alias 'ca' in Device Server Key Tool), the signed server certificate (default alias 'ds-signed') and server private key (default alias 'ds') for example:

```
-----BEGIN PRIVATE KEY-----
ME0CAQAwEwYHKoZIzj0CAQYIKoZIzj0DAQcEMzAxAgEBBCDS3kGK3uQ1vh3keRDV
8xecNcyZp9D5sXVeP4jP9GFMb6AKBggqhkjOPQMBBw==
-----END PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIBSDCB76ADAgECAgQFe1/XMAoGCCqGSM49BAMCMDQxCzAJBgNVBAYTAkZJMRQwEgYDVQQKEwtB
Uk0gRmlubGFuZDEPMA0GA1UEAxMGUm9vdENBMCAXDTE0MDExMDA4MzAxNFoYDzIxMTMxMjE3MDgz
MDE0WjAjMQswCQYDVQQGEwJGSTEUMBIGA1UEChMLQVJNIEZpbmxhbmQwWTATBgcqhkjOPQIBBggq
hkjOPQMBBwNCAAQuBjNUDYkEtpjarHCvERQz2aT8W3eq5SaXylhBugd150UPxXWUphf0CUz+E4pi
9MY9UUpzGeM+v3Q5VnNcD04lMAoGCCqGSM49BAMCA0gAMEUCIQDsb59glhKlYFcuu3AQCoCQcJ0l
zOkyUiCbI08d5iOA5wIgXjlYyjF+GbPTnNs6eQOXoV3F0ZqGut5hRY3ZFml/d7M=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIBgDCCASWgAwIBAgIEIZcGAjAMBggqhkjOPQQDAgUAMDQxCzAJBgNVBAYTAkZJMRQwEgYDVQQK
EwtBUk0gRmlubGFuZDEPMA0GA1UEAxMGUm9vdENBMCAXDTE0MDExMDA4MzAxMVoYDzIxMTMxMjE3
MDgzMDExWjA0MQswCQYDVQQGEwJGSTEUMBIGA1UEChMLQVJNIEZpbmxhbmQxDzANBgNVBAMTBlJv
b3RDQTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABHCtdJ5qFd35pnzW42LPBgHCBrwh9z4LSL+g
kYxfWLhcAKBGMzoNSkf3WSWV9YzRvFBshqZSIPMlV+VlSaAJ8ZWjITAfMB0GA1UdDgQWBBT+Zvp3
VVRwjzIi4ni7u+FG9b3ypTAMBggqhkjOPQQDAgUAA0cAMEQCIBBWEPmziyy2SUWyIrziaaZouln7
EJiJ20MMD0YPCm28AiBAvGSVx7gBliQe1acsGbkKbqLSrlhadFk+5kAKcyaX6Q==
-----END CERTIFICATE-----
```

Clients credentials

`GET /admin-api/clients`
Returns list of clients

`POST /admin-api/clients`
Create client credential

`PUT /admin-api/clients/{domain}/{client-name}`
Update client's credential (password)

Trusted certificates

Admin can upload certificates that are accepted by the Device Server while sending notifications to the web applications through a secure channel. If there are no trusted certificates added then Device Server trusts all web applications. Trusted certificates can be uploaded using the Admin API or initial certificates can be put as files under `conf/trusted` in binary DER or Base64

encoded PEM format. Those files are read during startup and certificates found on them are being trusted.

List of aliases

`GET /admin-api/security/trusted`
Returns list of trusted aliases as a JSON array.

```
GET /admin-api/security/trusted

==>
200 OK
Content-Type: application/json

["webapp-1.der", "webapp-2.cer"]}}}
```

List of aliases with CN

`GET /admin-api/security/trusted/*`
Returns list of trusted aliases with CN as a JSON array.

```
GET /admin-api/security/trusted

==>
200 OK
Content-Type: application/json

[{"alias":"webapp-1.der","cn":"CN=domain/webapp-1"},
{"alias":"webapp-2.cer","cn":"CN=webapp-2"}]
```

Add new trusted certificate

`PUT /admin-api/security/trusted/{alias}`
Adds new trusted certificate with the given alias.

```
PUT /admin-api/security/trusted/webapp-2.cer
Content-Type: text/plain

-----BEGIN CERTIFICATE-----
MIIBTDCB8qADAgECAgRUJATUMAoGCCqGSM49BAMCMEoxCzAJBgNVBAYTAkZJMQ0w
CwYDVQQHDARPdWx1MQwwCgYDVQQKDANBUk0xDDAKBgNVBAsMA0lvVDEQMA4GA1UE
AwwHQVJNIElvVDAeFw0xNDA5MjUxMjA0NDNaFw0xNTA5MjUxMjA0NDNaMBIxEDAO
BgNVBAMMB25vZGUwMDEwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAS9kcAimwGI
MY92jfyehY8aNIcPKLH1dcW4yah/YbXUxjDk+iuQlbz1PBG+CEDXI3cjhTw9mqI9
2VNvs2v4zUnqMAoGCCqGSM49BAMCA0kAMEYCIQCK9nKX/50RsW5j2bGNXYvLyc00
40KGCzyw0BK7kI5h7QIhAM7E08jouO3D0B679hRJpwatyCOFRClGvL903Pm0VELk
-----END CERTIFICATE-----

==>
204 OK
```

Retrieve trusted certificate

`GET /admin-api/security/trusted/{alias}`
Returns trusted certificate of the given alias.

```
GET /admin-api/security/trusted/webapp-1.der

==>
200 OK
Content-Type: text/plain

[
[
  Version: V3
  Subject: CN=webapp-1
  Signature Algorithm: SHA256withECDSA, OID = 1.2.840.10045.4.3.2

  Key:  Sun EC public key, 256 bits
  public x coord: 85744647274891019041152188477351379318998259828071898365442213
  public y coord: 22115584473857231286236512143669893103167010110141601736416097
  parameters: secp256r1 [NIST P-256, X9.62 prime256v1] (1.2.840.10045.3.1.7)
  Validity: [From: Thu Sep 25 15:04:43 EEST 2014,
              To: Fri Sep 25 15:04:43 EEST 2015]
  Issuer: CN=ARM IoT, OU=IoT, O=ARM, L=Oulu, C=FI
  SerialNumber: [    542404d4]

]
  Algorithm: [SHA256withECDSA]
  Signature:
0000: 30 46 02 21 00 8A F6 72   97 FF 9D 11 B1 6E 63 D9  0F.!...r.....nc.
0010: B1 8D 5D 8B CB C9 CD 34   E3 42 86 0B 3C B0 D0 12  ..]....4.B..<...
0020: BB 90 8E 61 ED 02 21 00   CE C4 D3 C8 E8 B8 ED C3  ...a..!.........
0030: D0 1E BB F6 14 49 A7 06   AD C8 23 85 44 29 46 BC  .....I....#.D)F.
0040: BF 74 DC F9 B4 54 42 E4                            .t...TB.

]
```

Delete trusted certificate

`DELETE /admin-api/security/trusted/{alias}`
Removes the trusted certificate with the given alias.

```
DELETE /admin-api/security/trusted/webapp-1.der

==>
204 OK
```

Blacklist

`GET /admin-api/black-list`

Returns blacklisted endpoints. No endpoint, with name that is listed, will be allowed to register. Such an attempt will result in CoAP 4.03 Forbidden. In case of certificate based secure communication, handshake will be rejected.

`POST /admin-api/black-list`

Add endpoints to black list. Note that if added endpoint exists in system.

`POST /admin-api/black-list?remove=true`

Remove endpoints from black list.

`GET /admin-api/black-list/{endpoint-name}`

Check if endpoint is in black list (200 - exists; 404 does not exist).

`DELETE /admin-api/black-list/{endpoint-name}`

Remove endpoint from black list.

Examples

```
POST /admin-api/black-list
Content-Type: application/json

["node-1241", "node-4543"]
=>
204 No Content
```

```
GET /admin-api/black-list

=>
200 OK
Content-Type: application/json

["node-1241", "node-4543"]
```

```
POST /admin-api/black-list?remove=true
Content-Type: application/json

["node-4543"]
=>
204 No Content
```

Device Server Monitoring and Management

Device Server provides Key Performance Indicators realtime accessible through JMX. For a quick check we recommend to use Oracle's Java Mission Control tool (jmc) that is included in the JDK from version 7 Update 40. Connect to the JVM that you want to monitor and select MBean Browser tab for browsing all the MBean features. Device Server related mbeans can be found

under `com.arm.mbed.deviceserver` / `Monitoring`. The following mbeans and attributes are exposed on this interface:

Coap

IncomingTpsCoapUdp	Incoming TPS received on CoAP/UDP interface
IncomingTpsCoapTls	Incoming TPS received on CoAP/TLS interface
IncomingTpsCoapTcp	Incoming TPS received on CoAP/TCP interface
IncomingTpsCoapEdtls	Incoming TPS received on CoAP/eDTLS interface
OutgoingTpsCoapUdp	Outgoing TPS sent on CoAP/UDP interface
OutgoingTpsCoapTls	Outgoing TPS sent on CoAP/TLS interface
OutgoingTpsCoapTcp	Outgoing TPS sent on CoAP/TCP interface
OutgoingTpsCoapEdtls	Outgoing TPS sent on CoAP/eDTLS interface
CoapRetransmissions	Number of CoAP retransmissions
CoapTimeouts	Number of CoAP timeouts
CoapParsingFailures	Number of CoAP parsing failures
DuplicatedMessages	Number of duplicated CoAP messages
CoapRoundTripTime	Average roundtrip-time for the CoAP messages in the last 10 seconds
EdtlsHandshakes	TPS of successful eDTLS handshakes
EdtlsRecordsRepeated	Number of eDTLS records repeated in the last 60 seconds
EdtlsAlertsSent	eDTLS failures: number of alerts sent in the last 60 seconds
EdtlsAlertsReceived	eDTLS failures: number of alerts received in the last 60 seconds

Eventing

RegistrationTps	Speed of incoming registration requests
NotificationTps	Speed of incoming notification requests
QueuedSubscriptionTps	Speed of outgoing queued subscription requests
QueuedSubscribedTps	Speed of successful queued subscription requests
QueuedSubscriptionFailures	Number of failed queued subscription requests

M2M-Http

M2MHttpIncomingTps	Incoming TPS received on M2M-HTTP
M2MHttpOutgoingTps	Outgoing TPS sent on M2M-HTTP
M2MHttpRoundTripTime	Average roundtrip-time for the M2M-HTTP messages in the last 10 seconds
M2MHttpParsingFailures	Number of failures in the incoming messages

REST API

ProxyRequestsAccepted	Number of accepted proxy requests in the last 60 seconds
ProxyRequestsRejected	Number of rejected proxy requests in the last 60 seconds
RestApiRequestsAccepted	Number of accepted requests in the last 60 seconds
RestApiRequestsRejected	Number of rejected requests in the last 60 seconds

ThreadPool

AdminUIQueueSize	Queue size in AdminUI thread-pool
AdminUIActiveThreads	Number of active threads in AdminUI thread-pool
AdminUIRejectedRequests	Number of rejected AdminUI messages
CoapQueueSize	Queue size in CoAP thread-pool
CoapActiveThreads	Number of active threads in CoAP thread-pool
CoapRejectedMessages	Number of rejected CoAP messages
RestQueueSize	Queue size in Rest thread-pool
RestActiveThreads	Number of active threads in Rest thread-pool
RestRejectedRequests	Number of rejected Rest messages
M2MHttpQueueSize	Queue size in M2MHttp thread-pool
M2MHttpActiveThreads	Number of active threads in M2MHttp thread-pool
M2MHttpRejectedRequests	Number of rejected M2MHttp messages
QueueConnectorQueueSize?	Queue size in QueueConnector? thread-pool

Storage

Writes	Storage Writes Per Seconds
Searches	Storage Searches Per Seconds
Reads	Storage Reads Per Seconds
Size-resourceCache	Current size of the storage
Size-client-notification-handler	Current size of the storage
Size-groups	Current size of the storage
Size-client-pre-subscriptions	Current size of the storage
Size-client-credentials	Current size of the storage
Size-domains	Current size of the storage
Size-endpoints	Current size of the storage
Size-blacklisted-endpoints	Current size of the storage
Size-edtls-shde-client	Current size of the storage
Size-edtls-shde-server	Current size of the storage
Size-edtls-sessions	Current size of the storage
Size-edtls-credentials	Current size of the storage
Size-edtls-retransmission	Current size of the storage
Size-edtls-certificates	Current size of the storage
Size-edtls-nonce	Current size of the storage
Size-update-triggers	Current size of the storage

notification-channel

ChannelsQueueSize	Notification channel's queue size