Access Control API

2022-08

The goal of this document is to get the API up and running, and provide both general usage guidelines and example requests.

Version	Description
1	2022-08-10 Initial version

Table of contents

Access Control API 1
1 Installation 3
2 Configuration 3
2.1 Create technical user account used for API communication 3
2.2 Adjust Identity Server appsettings files 4
3 REST API 4
3.1 Accessing Swagger UI 4
3.1.1 Authenticate session 4
3.2 Authentication 5
3.3 Paging 5
3.4 Versioning 5
3.5 Example Requests 5
3.5.1 Get version information 5
3.5.2 Query available employees 6
3.5.3 Create an employee 6
3.5.4 Create an access card assigned to the created employee 7
3.5.5 Assign an access authorization to an employee 7
3.5.6 Update an employee 7
3.5.7 Delete an employee 8
3.5.8 Get devices 8
3.5.9 Get device type information 9
3.5.10 Execute a command request 9
3.5.11 Reply to an access request message 10
4 gRPC API 10
4.1 Authentication 11
4.2 Example messages 11

4.2.1 Get version information 11
4.2.2 Get event messages 12
4.2.3 Get state change messages 12

2 13

1 Installation

As a prerequisite for using the Access Control API, you need to have an AMS instance installed and configured. Install AMS from the provided “AMS Server Installer.exe” as outlined in the AMS Installation Guide.

2 Configuration

After the AMS installation has completed and you have restarted the system, some additional manual configuration is required for Access Control API. The following sections provide details on the required steps, which are:

- Create technical user account used for API communication
- Adjust Identity Server appsettings files

2.1 Create technical user account used for API communication

For the API communication you need to create a separate technical user account (“operator” in AMS terms).

In order to create a new operator, you need first to create a new person. To do this, start the AMS Dialog Manager and go to “Personnel data” > “Persons”. Fill in an arbitrary value in “Last name” and select “Employee” as the value for “Person class”. Save the record.

After this go back to the main menu and then to “Configuration” > “Operators and Workstations” > “User rights”. Search for the created employee, if not selected anymore. Enter a value into “User name”, this will be later used as client identifier. Click now on “Change password …” and select a password for the user.

Save the record.

If you have multiple divisions configured, you have to go to the “Divisions” tab in the same dialog and assign all divisions you want the API has access to.

For example, if you don’t assign a division “Test Division” you won’t get any state changes or event messages. Besides, you will not be able to read, create, update or delete entities like employees for the division.

2.2 Adjust Identity Server appsettings files

Create a new file called `appsettings.Extension.AccessControlApi.json` in the folder of the Identity Server (default location is `C:\Program Files (x86)\Bosch Sicherheitssysteme\Access Management System\Identity Server`) with the following content and replace the placeholder with the operator username.

```
{
  "ClientOptions": {
    "Clients": {
      "AmsIntegration": {
        "ClientId": "<OPERATOR-USERNAME>",
        "ClientName": "<NAME-OF-YOUR-APPLICATION>",
        "AllowedGrantTypes": [
          "client_credentials"
        ],
        "AccessTokenLifetime": 1800,
        "AllowedScopes": [
          "ProductApi"
        ],
        "Properties": {
          "IsOperatorBacked": "true"
        }
      }
    }
  }
}
```

When finished, start the “ACE Process Control” application. (By default, the AMS setup will place a shortcut to on your desktop.) Select the row labelled `IDENTITYSERVER` and click the button “Restart process”.

3 REST API

The REST API can be accessed at **https://<YOUR-HOSTNAME>:62999/**.

3.1 Accessing Swagger UI

Access Control API provides comprehensive documentation and facilities for experimentation via Swagger UI. Access Swagger UI by navigating to **https://<YOUR-HOSTNAME>:62999/swagger** in your browser (for instance Chrome).

3.1.1 Authenticate session

In order to use the routes that require authentication, click the green “Authorize” button. In the **client_id** input field enter your operator username and for the **client_secret** use the operator password. Select the checkbox next to **ProductAPI - Access Control API** and click the “Authorize” button.

The Swagger UI will not refresh your access token when it expires after 30 minutes. When the access token expires, click the “Authorize” button, press “Logout” and “Authorize” to get a new access token.

3.2 Authentication

Most requests to the Access Control API endpoint must be authenticated. The API relies on the Client Credentials flow in OpenID Connect for authentication.

You can obtain a compliant JWT-format token from the AMS’ Identity Provider based on the OpenID Connect client application you have configured. Pass the acquired access token in an `Authorization: Bearer ...` HTTP header in requests.

For the Swagger UI you can authenticate the session from the UI itself.

3.3 Paging

By default, the API will limit the amount of data returned from a single request to avoid overloading the client application, and return only the first batch of items for each query. When this occurs, the server response will include a non-null **cursor** property, which you can then pass to a subsequent request in order to fetch the next batch of results.

3.4 Versioning

If you do not explicitly specify a version in your request, the API assumes that you want to use the most recent version.

In order to specify a version manually you have two options:

- Query String
- Media Type Header

For the query string, you can extend the route with the version information e.g. **GET https://<YOUR-HOSTNAME>:62999/api/version?api-version=1.0**.

When using the Media Type Header you need to add an `Accept` header to the request with `application/json;v1.0` to specific version `1.0`.

3.5 Example Requests

This is a list of example requests with their payload. In the examples are some further descriptions about the API.

3.5.1 Get version information

The most basic route to check the connection with the API is to fetch version information as you do not need to provide an access token for authentication.

Route: `GET https://<YOUR-HOSTNAME>:62999/api/version`

3.5.2 Query available employees

After the initial connection is successful, you can try to query the available employees from the system. For this, an authenticated request is required (see Authentication).

Route: `GET https://<YOUR-HOSTNAME>:62999/api/employees`

This route supports paging by adding the `cursor` as query parameter:

Route with cursor: `GET https://<YOUR-HOSTNAME>:62999/api/employees?cursor=...`

3.5.3 Create an employee

Route: **POST https://<YOUR-HOSTNAME>:62999/api/employees**

Payload:

```
{
  "firstName": "John",
  "lastName": "Doe",
  "birthName": null,
  "personnelNumber": "12345678",
  "dateOfBirth": "1960-10-23",
  "gender": "Male",
  "department": null,
  "jobTitle": null,
  "eMail": "john.doe@mail.com",
  "phoneHome": "+49 (000) 000 00000",
  "phoneBusiness": null,
  "phoneMobile": null,
  "phoneOther": null,
  "companyId": null,
  "divisionId": "FF0000D300000002"
}
```

Hint: You can omit properties with a **null** value.

In the response you will find a header value called `Location`, for instance: `Location: https://<YOUR-HOSTNAME>:62999/api/employees/00139D53B66AAF67`

The identifier at the end `00139D53B66AAF67` is the identifier of the new created employee. When calling the specified route, you will get the current values for the employee.

3.5.4 Create an access card assigned to the created employee

Creating an access card requires to know the code data of an access card. To get example code data you can assign an access card from the AMS application and read it with the API at **GET https://<YOUR-HOSTNAME>:62999/api/accessCards**.

It depends on the used card encoding but it is usually split into two parts:

1. The higher 8 bits, which is usually fixed and dependent on the card type and technology, like **00000001**00000004 (marked bold digits represents the higher 8 bits)
2. The most lower 8 bits which represent the card number itself in HEX value (usually printed on the card): 00000001**00000004** (marked bold digits represents the lowest 8 bits)

Route to add a new card: `POST https://<YOUR-HOSTNAME>:62999/api/accessCards`

```
{
  "codeData": "0000000000000042",
  "technologyType": "IClass",
  "dataType": "SerialNumber",
  "numberOfBits": 32,
  "personId": "00139D53B66AAF67",
  "divisionId": "FF0000D300000002"
}
```

Same as the employee, you will find the identifier in the response headers.

3.5.5 Assign an access authorization to an employee

Prerequisites: Already configured entrances and access authorizations

Get the current access authorizations with the route `GET https://<YOUR-HOSTNAME>:62999/api/accessAuthorizations`.

From this, you can get the identifiers required for assigning an access authorization to an employee.

Route: `POST https://<YOUR-HOSTNAME>:62999/api/persons/00139D53B66AAF67/accessAuthorizations/00139D53EDFF57BB`

The first identifier is from the created employee, the second one is an identifier of an access authorization.

3.5.6 Update an employee

You can also update details of an existing employee.

Route: `PUT https://<YOUR-HOSTNAME>:62999/api/employees/00139D53B66AAF67`

Payload (providing only a subset of properties here, all other properties will be reset - see below for details):

```
{
  "firstName": "John",
  "lastName": "Doe",
  "dateOfBirth": "1960-10-23",
  "gender": "Male",
  "eMail": "john.doe@mail.com",
  "phoneHome": "+49 (123) 000 00000"
}
```

All **PUT** requests have in common that omitting a property is the same as setting the property to the default value (like **null**).

In the example payload, if a `personnelNumber` was set previously, after the request it will be set to `null`.

The default response code for `PUT` requests is `204 No Content` which indicates that the request was successfully accepted and the entity is changed accordingly.

3.5.7 Delete an employee

Route: **DELETE https://<YOUR-HOSTNAME>:62999/api/employees/00139D53B66AAF67**

The `DELETE` request, like the `PUT` request, answers with a `204 No Content` when the employee is successfully deleted.

3.5.8 Get devices

Route: **GET https://<YOUR-HOSTNAME>:62999/api/devices**

Returns a list of devices like this:

```
{
  "devices": [
    {
      "id": "FF00003300000003",
      "name": "MAC",
      "description": "MAC",
      "parentId": "FF00003300000001",
      "deviceTypeId": "MAC",
      "capabilities": []
    },
    {
      "id": "FF00003300000001",
      "name": "DMS",
```

```
      "description": "DMS-Server",
      "parentId": null,
      "deviceTypeId": "DMS",
      "capabilities": []
    },
  ],
  "nextCursor": null
}
```


3.5.9 Get device type information

When you fetched the devices, you see, that every device has a **deviceTypeId**. With this, you can get additional information like what kind of command types are allowed.

Route: `GET https://<YOUR-HOSTNAME>:62999/api/deviceTypes/MAC`

Example request:


```
{
  "id": "MAC",
  "name": "MAC",
  "description": "Main Access Controller",
  "stateTypeIds": [
    "CONNECTION",
    "ASM"
  ],
  "commandTypeIds": [
    "MacEnableAccessSequenceMonitoring",
    "MacDisableAccessSequenceMonitoring"
  ]
}
```


3.5.10 Execute a command request

If you know a device identifier and a command type identifier, you can use both to execute a command request.

Route: `POST https://<YOUR-HOSTNAME>:62999/api/devices/execute`

Example payload:


```
{
  "deviceId": "FF00003300000003",
  "commandTypeId": "MacEnableAccessSequenceMonitoring"
}
```


Example response:


```
{
  "success": true,
  "error": null,
  "payload": {}
}
```

Important: The property **success** does not indicate if the command is executed successfully, it describes that the command request was successfully issued.

If the MAC of the requested device is offline, you will always get an error message, but e.g., if a reader is offline, you will get a success message as the request was successfully sent to the MAC of the device.

3.5.11 Reply to an access request message

In this final example for the REST API, we will show how to respond to an access request message as it may occur in a video verification use case, in which a human operator will manually verify the requestor's identity and either grant or deny entry.

Prerequisite: You need to configure your system to send access requests and received an access request message.

Route: `POST https://<YOUR-HOSTNAME>:62999/api/devices/execute/accessResponse`

Example request body for granting request:

```
{
  "deviceId": "00139D6A1A14011D",
  "commandTypeId": "ReaderReplyAccessGranted",
  "credentialId": "0013A6EA217AD163"
}
```

Example request body for denying request:

```
{
  "deviceId": "00139D6A1A14011D",
  "commandTypeId": "ReaderReplyAccessDenied",
  "credentialId": "0013A6EA217AD163"
}
```

The information for **deviceId** and **credentialId** can be found at the event message.

The response is the same as for a command request. As before, a "successful" response indicates that the request was successfully accepted, but not that the request was successful executed.

4 gRPC API

With the gRPC API you are able to stream messages from the API to your application. You can access the gRPC API at **https://<YOUR-HOSTNAME>:62998/**.

In the folder of the Access Control API (default location is `C:\Program Files (x86)\Bosch Sicherheitssysteme\Access Management`

`System\ProductApi`) is a folder called `docs`. This folder contains the corresponding Protocol Buffers schema as a `.proto` file. You can generate a client in your preferred programming language from this schema.

4.1 Authentication

As already mentioned in the Authentication chapter for the REST API most requests must be authenticated.

For a successful authentication you need to access the API with a SSL connection and an access token as part of the metadata.

Example for C#:

```
var callCredentials = CallCredentials.FromInterceptor(async (_,
metadata) =>
{
    // Call custom function to get an access token
    var accessToken = await GetAccessTokenAsync();
    metadata.Add("Authorization", $"Bearer {accessToken}");
});

var channelOptions = new GrpcChannelOptions
{
    Credentials =
ChannelCredentials.Create(ChannelCredentials.SecureSsl,
callCredentials),
};

var channel = GrpcChannel.ForAddress("https://<YOUR-HOSTNAME>:62998",
channelOptions);
```

The gRPC website provides further information about authentication.

4.2 Example messages

This is a list of example messages with their payload. Please refer to the Protocol Buffers schema definition for full technical details.

4.2.1 Get version information

The most basic action to check the connection with the gRPC API is to fetch version information as you do not need to provide an access token for authentication.

Example response:

```
{
  "version": "1.0",
  "assemblyVersion": "1.x.x.x",
  "productVersion": "...",
  "supportedVersions": [
    "1.0"
  ],
```

```
  "deprecatedVersions": []
}
```


4.2.2 Get event messages

With the **StreamEvents** method, you can stream all kind of events like access granted, access denied, device online / offline, door opened / closed and many more.

Example message when an AMC goes offline:

```
{
  "id": "3320d0d8-50c9-4eff-b18d-9be4632de048",
  "eventTypeId": "16777222",
  "device": {
    "id": "00139D6A177FF48A",
    "name": "AMC-1"
  },
  "creationTime": "2022-01-01T12:00:00.0000000+02:00",
  "loggedTime": "2022-01-01T12:00:00.0000000+02:00",
  "expirationTime": ""
}
```

Example message for a granted access:

```
{
  "id": "51b9a160-e44f-429f-88ef-31218a0376c8",
  "eventTypeId": "16777986",
  "device": {
    "id": "00139D6A1A14011D",
    "name": "Entry Reader 1-1"
  },
  "person": {
    "id": "00139D53B66AAF67",
    "firstName": "John",
    "lastName": "Doe"
  },
  "credential": {
    "id": "0013A6EA217AD163"
  },
  "area": {
    "id": "FF00000900000002",
    "name": "Outside"
  },
  "creationTime": "2022-01-01T12:00:00.0000000+02:00",
  "loggedTime": "2022-01-01T12:00:00.0000000+02:00",
  "expirationTime": ""
}
```

When there is an access request, the property **expirationTime** is filled with a date (like **creationTime**). Then you can send a reply to this access request.

4.2.3 Get state change messages

The **StreamStates** allows you to fetch state changes for example when a door was opened.

13 13

Example message:

```
{
  "sourceId": "00139D6A1A13F163",
  "typeId": "DOOR",
  "value": "OPEN",
  "lastUpdated": "2022-01-01T12:00:00.0000000+02:00"
}
```

--End of document--