NetIQ Advanced Authentication Framework

API Documentation

Version 5.1.0

Table of Contents

Introduction

About This Document

This document describes the HTTP REST API for NetIQ Advanced Authentication Framework Server V5. The document is intended for developers and contains information on how to integrate strong authentication into their applications.

The current API version is 1.0. Since API is based on REST principles, it is easy to write and test applications. The browser can be used to access URLs. Any HTTP client in any programming language can be also used to interact with the API.

API accepts and responds with JSON objects.

Base URL

The Base URL of the REST API is `https://authserver.example.com/api/v1/`. Substitute authserver.example.com with the hostname of your appliance.

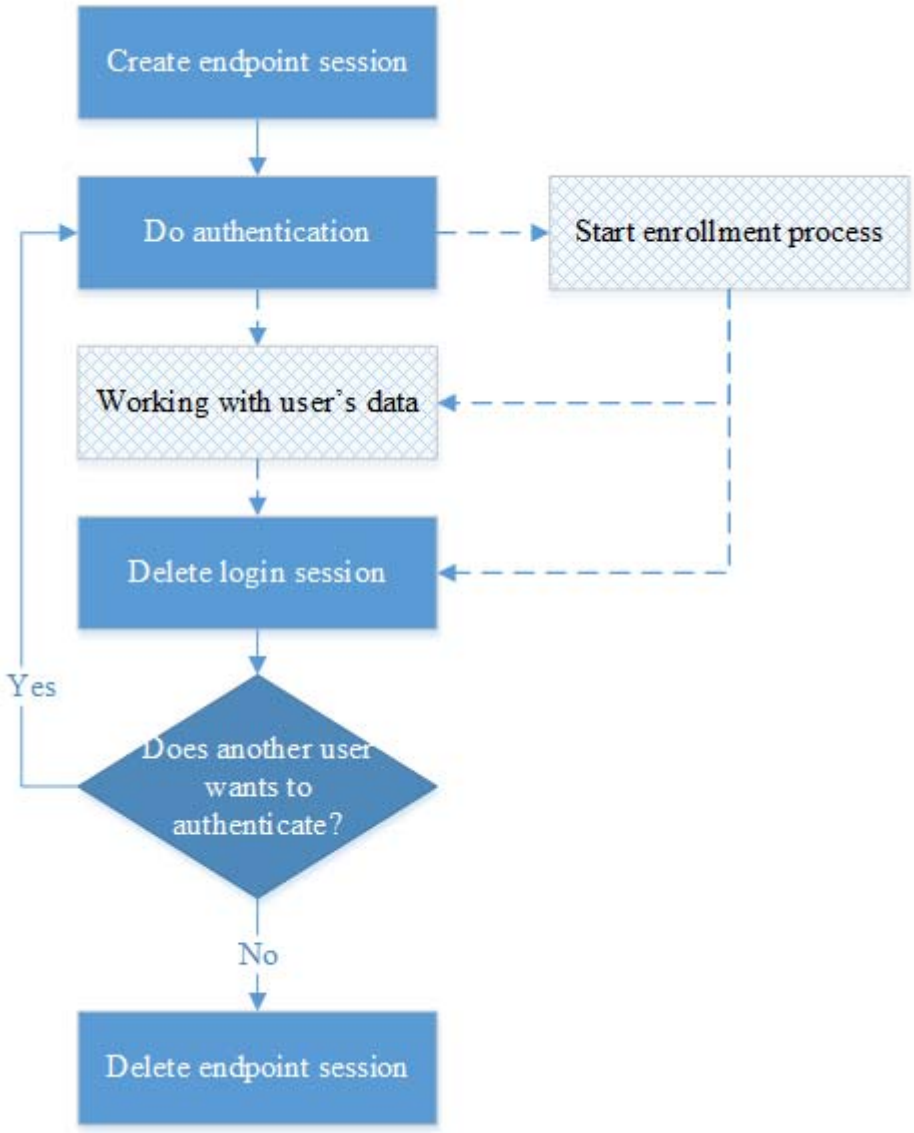
Use the HTTPS protocol for more security with interaction with server's REST API. The authentication server can provide some security information, e.g. Domain password. Note that some network libraries can block self-signed SSL certificate.

Main Process

The Authentication Server has two main processes:

- logon process
- enrollment process

The logon process provides authentication and authorization, the enrollment process stores authentication information for new authentication methods. If the logon process is not successful, the enrollment process will not be started. The following schema describes the logon process.

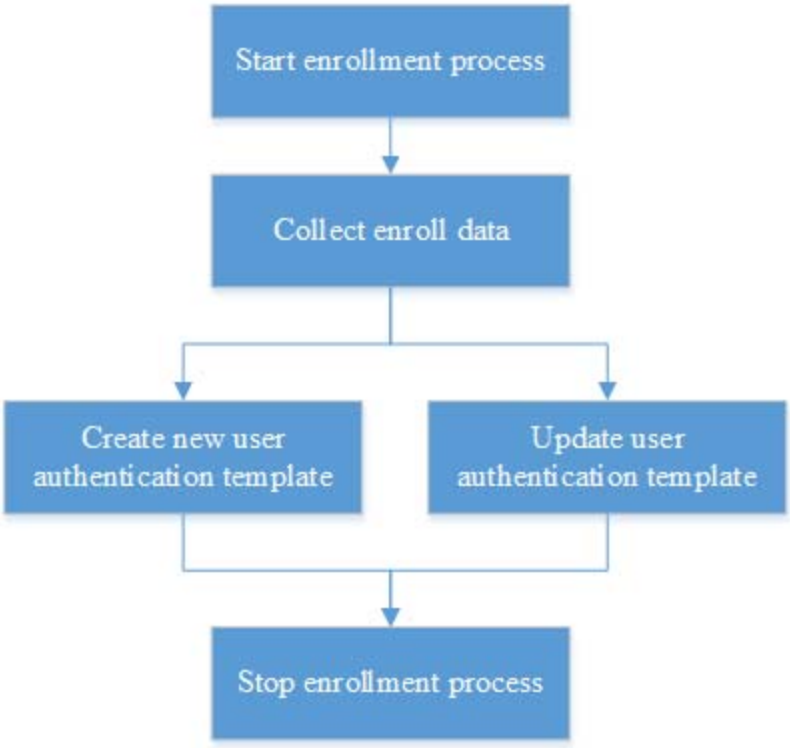


The logon process has several steps:

1. **Create endpoint session** - a sub process. For more information, see the Working with Endpoint Sessions chapter.
2. **Do authentication** - user can authenticate after creating the endpoint session. Authentication is a sub process. For more information, see the Providing Authentication chapter.
3. **Working with user's data** - user can work with user's data after the successful authentication. For more information, see the Working with User's Data chapter. User is also provided with a capability to start enrollment process (if applicable).
4. **Delete login session** - user is provided with a capability to delete login session when all work with the server is done. For more information, see the Deleting Login Session chapter.

5. **Does another user want to authenticate?** - after the login session is deleted, the system will wait for another user. If a new user wants to login, the process will be started from the second step because the endpoint is already created.
6. **Delete endpoint session** - the logon process will delete endpoint session at the end. For more information, see the Deleting Endpoint Session chapter.

The following schema describes the enrollment process.



Enrollment process has several steps:

1. **Start enrollment process** - this step is the primary one in the enrollment process. For more information, see the Starting Enroll Process chapter.
2. **Collect enroll data** - when the enrollment process is started, user should collect all enrollment data. For more information, see the Providing Data Into Enroll Process chapter.
3. User decides what to do with enrollment data:
    - **Create new user authentication template** - user is provided with a capability to create new authentication template. For more information, see the Creating User Templates From Enroll Session chapter.
    - **Update user authentication template** - user is provided with a capability update existing template with new enrollment data after collecting enroll data. For more information, see the Updating User Template chapter.

4. **Stop enrollment process** - after collecting enrollment data, enrollment process should be stopped. For more information, see the Deleting Enroll Process chapter.

The Working with Enrollment chapter contains information of the enrollment process. The Working with User Templates chapter describes users' templates.

API Documentation

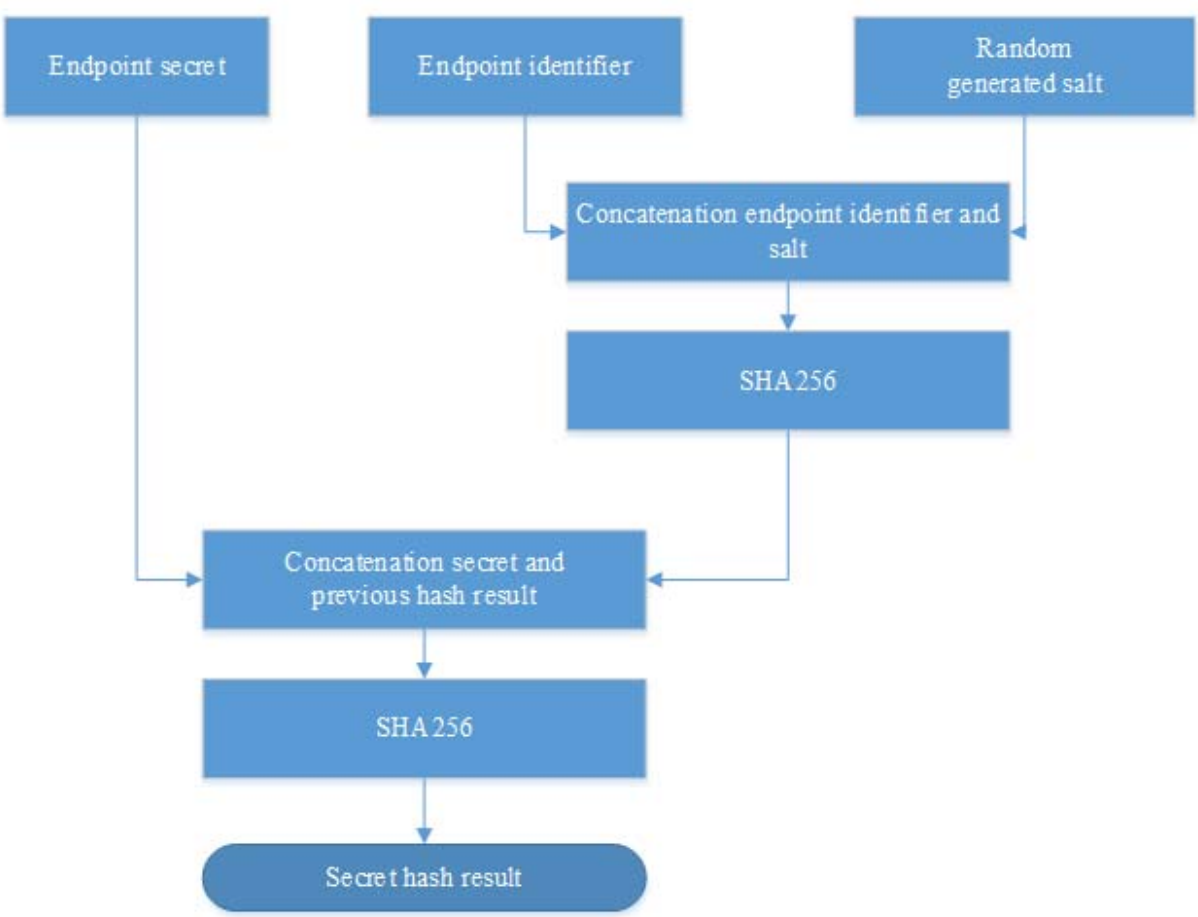
In this chapter:

Working with Endpoint Session

In this chapter:

About Endpoint Session

Endpoint is any logical or physical unit which interacts with the authentication server. E.g., client computer, tablet device, smartphone, any software or system is an endpoint. Endpoint should create endpoint session on the server to start working. Each endpoint has identifier and secret. Secret is a security value that is used for generating endpoint security hash. Security hash is used to start endpoint session. The algorithm for generating the secret hash is represented on the following schema.



After SHA-256 function, you should transform result to HEX-value string. All parameters are string. All concatenation functions are string concatenation functions. Salt should be generated on the client side. Strong algorithm should be used for its generation.

The following example shows how to generate secret hash.

```
Endpoint id =   4242424242424242424242424242424242
Salt = e26eaecba7cbe186c08469f6ddbf6f6c0321651b53f80d8eb2c3b0d4e1c19c4c
Endpoint secret =   12345678
Middle result = e53e75ec3d48171-
abff1e4d2f0a5066e64fdbe11a4d3fd7c45810d7d93c26956 = SHA256(endpoint id + salt)
Result 3b5dac383282df6936f9350a01ad079096f777f5c44eda8e0c2e66bfc443ee26 =
SHA256(endpoint secret + middle result)
```

Endpoint identifier and endpoint secret need to be very secure. This information should be kept in a secure place and parameters should be changed every time the information is compromised.

Endpoint can work with the server and provide user authentication after the endpoint session creation. Endpoint session has a lifetime, after this endpoint session will be deleted and the session will need to be renewed. All users can work with one endpoint session on one endpoint.

Creating Endpoint Session

To create the endpoint session, use the following resource with URI:

```
/endpoints/{endpoint_id}/sessions
```

Resource is provided by HTTP POST. It has the following parameter.

URI parameter name	Parameter description
endpoint_id	Endpoint identifier. Identifier of endpoint that creates endpoint session.

Resource accepts JSON-object with the following parameters.

Parameter name	Parameter description
salt	Client generated salt. This salt is used in secret hash generated algorithm.
endpoint_secret_hash	Generated secret hash by algorithm.
session_data	Any session data. This is a JSON-object with any parameters and structure.

The resource returns JSON-object with the following endpoint session identifier.

Parameter name	Parameter description
endpoint_session_id	Endpoint session identifier. This is an identifier of the created endpoint session. It should be used for other methods.

Example

Endpoint with identifier "42424242424242424242424242424242" creates endpoint session. Endpoint has already generated salt and endpoint secret hash.

```
HTTP POST
https://authserver.example.com/api/v1/endpoints/42424242424242424242424242424242/sessions

Request
        {"salt":"2615c070937935246c6a91df70a8eb672b21d842a225621c9797a83bedf00a7b",
```

```
        "endpoint_secret_hash":"38d55f-
b7a899dcef6cbec053df8f7673cb05068b9ee9d6a23ee759232b25cf4e",
        "session_data":{}}

Response

        {"endpoint_session_id":"IRx7UXwenytMn5B7fgRSH4k1s6PAAs0I"}
```

Reading Information about Endpoint Session

To read information about endpoint session, use the following resource with URI:

```
/endpoints/{endpoint_id}/sessions/{session_id}?salt={salt}&endpoint_secret_
hash={endpoint_secret_hash}
```

Resource is provided by HTTP GET. It has the following parameters.

URI parameter name	Parameter description
endpoint_id	Endpoint identifier. Identifier of endpoint which session will be read.
session_id	Session identifier. Identifier of endpoint session for reading.
salt	Salt. It is used for generated endpoint secret hash.
endpoint_secret_hash	Endpoint secret hash.

Resource returns JSON-object which contains the following information about endpoint session.

Parameter name	Parameter description
sid	Endpoint session identifier, identifier of endpoint session.
endpoint_id	Endpoint identifier.
session_data	Session data, this data can be added on creating session. This is any JSON-object with any structure.

Example

We read the endpoint session information for endpoint with identifier "42424242424242424242424242424242". Session identifier is "T2Fv3FOvWojyBOZBlsik1d9RBDjDr9Rq". Endpoint has already prepared secret hash and salt.

```
HTTP GET
https://authserver.example.com/api/v1/endpoints/4242424242424242424242424242
242/sessions/T2Fv3FOvWojyBOZBlsik1d9RBDjDr9Rq?salt=279bc0da2a9da8afcfda50c2bc
```

```
3f418218354020d73c00a6a088dea704ece3f9&endpoint_secret_hash=bfe7b1e1cad57525a
9695f39eb5cff4d4dd7a42e039122c732adb151d097835c
```

Response

```
{"sid":"T2Fv3FOvWojyBOZBlsik1d9RBDjDr9Rq",
"endpoint_id":"42424242424242424242424242424242",
"session_data":null
}
```

Deleting Endpoint Session

To delete endpoint session, use the following resource with URI:

```
/endpoints/{endpoint_id}/sessions/{session_id}?salt={salt}&endpoint_secret_
hash={endpoint_secret_hash}
```

Resource is provided by HTTP DELETE. It has the following parameters.

URI parameter name	Parameter description
endpoint_id	Endpoint identifier. Identifier of endpoint which session will be deleted.
session_id	Session identifier. Identifier of endpoint session to be deleted.
salt	Salt. It is used for the generated endpoint secret hash.
endpoint_secret_hash	Endpoint secret hash.

Resource does not return any data. If the deletion is successful, the HTTP 200 status will be returned.

Example

Endpoint with identifier "42424242424242424242424242424242" deletes session with identifier "9Ml1ByqdIsCr3re5C0aivXR6ap20fNW8".

```
HTTP DELETE
```

```
https://authserver.example.com/api/v1/endpoints/42424242424242424242424242424
242/sessions/9Ml1ByqdIsCr3re5C0aivXR6ap20fNW8?salt=4b470b4ee96630ced049a78d64
88ee127b9d419f463530c63f3e1ee27226f721&endpoint_secret_hash=c2bf7b8288aa55de2
2baa9d077150d83b3460d5951a57d6e7ccf2a01f61bc9ec
```

Response

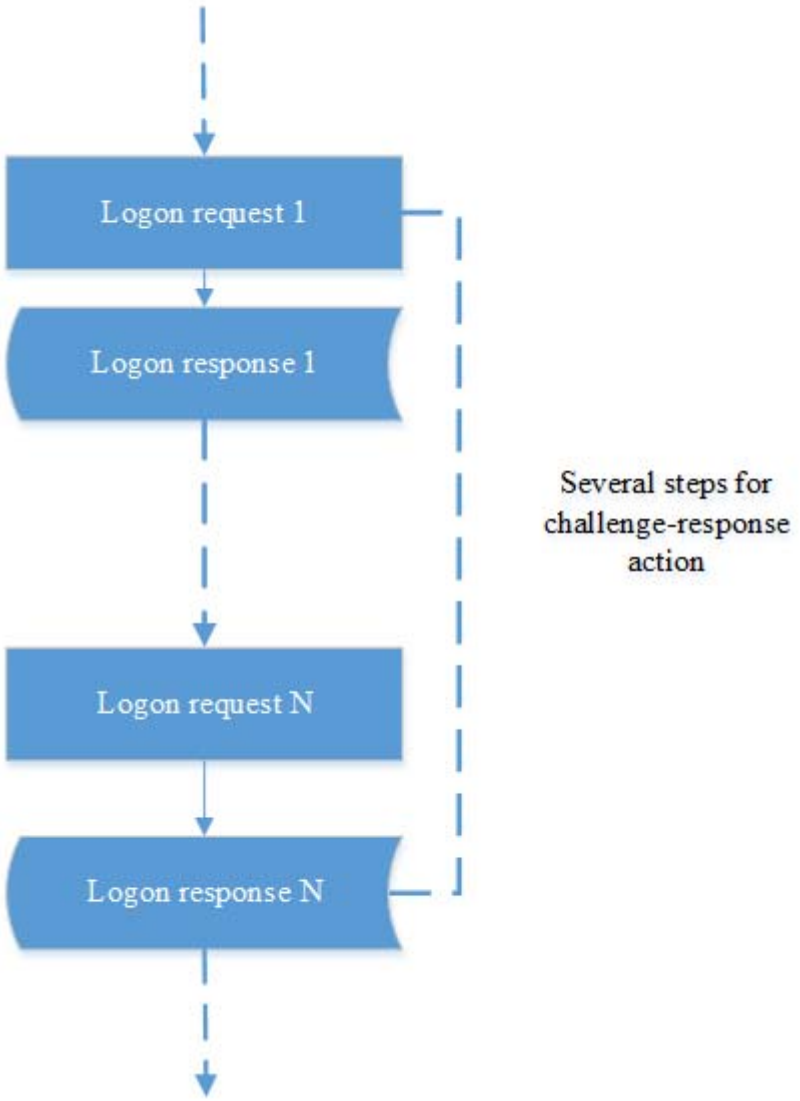
```
HTTP 200
```

Providing Authentication

In this chapter:

About Login Process

Server provides strong user's authentication by using the chain-login concept. Each chain is a challenge-response login. The following schema describes chain logic. To get successful authentication, the complete chain should be completed. A chain can consist of one or several authentication method(s).



User should start the logon process and pass all authentication methods. If all of them are successful, the user will login. If not, he/she will fail and should start the logon process once again.

API has the following standard responses for authentication status description.

Status name	Status value	Status description
OK	OK	This status is used to describe success result of logon process. Server has created the login session.
More data	MORE_DATA	This status is used to indicate that authentication method requests for additional data for authentication. Each authentication method requests different data.
Next	NEXT	This status is used for chained login process. It indicates that

		authentication method is successful and we should pass to the next method.
Failed	FAILED	This status is used for unsuccessful result of logon process description. It indicates that the logon process should be started once again.

Once the chain has been completed, user's data associated with event can be accessed. Event is the logical final destination for login process. Each event is a security point of system and login process provides authentication and authorization to this event. System has the following standard events.

Event name	Event identifier	Event description
Windows logon	windows_logon	Event for logon into Windows OS. It is used for authentication on Windows based operation systems.
Template management	TemplatesManagement	Template management event. It is used for the enroll process, collecting authentication data and updating user's template.
Endpoint management	EndpointsManagement	Endpoint management event. It is used for management endpoints.
NAM	NAM	Event for NetIQ NAM system.
NCA	NCA	Event for NetIQ NCA system.

Organizations can create their own login chains with different authentication method. Currently the following authentication methods are supported.

Authentication method name	Authentication method identifier	Authentication method description
LDAP password	LDAP_ PASSWORD:1	Authentication by LDAP password. System uses different LDAP users repository and provides authentication by LDAP password.
One-time password based on hash algorithm	HOTP:1	Authentication by OTP with hash algorithm.
One-time password based on time algorithm	TOTP:1	Authentication by OTP with time based algorithm.
One-time password sending by e-mail	EMAIL_OTP:1	Authentication by OTP sending by e-mail.

One time password sending by SMS	SMS_OTP:1	Authentication by OTP sending by SMS.
RADIUS password	RADIUS:1	Authentication by RADIUS server.
Security question	SECQUEST:1	Authentication by security question.
Smartphone authentication	SMARTPHONE:1	Authentication by smartphone application.
Virtual password	PASSWORD:1	Authentication by password assign to user.
Voice call	VOICE:1	Authentication by voice call.

Combining different authentication methods into authentication chains provides strong protection for different applications.

Before you start the logon process, you should create an endpoint session. The endpoint is the final destination for the login process. E.g., a client PC is an endpoint. One endpoint session can provide many logon processes. Endpoint session should be created once and used for all logon processes. For more information on endpoint session, see the Creating Endpoint Session chapter.

Providing Simple Authentication Using One Method in a Chain

API has two resources for providing login:

- for starting the login session
- for providing data to the authentication provider

This chapter describes how to provide a simple logon with one authentication method in the chain.

To start the logon process, the request should be sent to the following resource with URI:
`/logon`

Resource is provided by HTTP POST. It accepts JSON-object which describes the new logon process.

Parameter name	Parameter description
endpoint_session_id	Endpoint session identifier. Identifier of endpoint which creates logon session.
method_id	Authentication method identifier. Identifier of method from authentication methods list that is supported by the server. This parameter defines which method will be used for authentication. To view the list of the supported

	methods, see the About Login Process chapter.
user_name	User name for login. User name should be provided in format *repo_name/user_name*, where *repo_name* is user's repository name from server, *user_name* is user name.
application	Event identifier. Event from list of events supported by server.
is_1N	For later usage.
unit_id	For later usage.

Resource returns JSON-object with data of the created login session.

Parameter name	Parameter description
chains	JSON-object which describes available chains for login.
completed_methods	JSON-array which contains the list of completed authentication methods supported by the server.
current_method	Current authentication method which is used for authentication.
msg	Message about process. It contains more information about authentication process or about errors.
logon_process_id	Logon process identifier, identifier of current logon process. It is constant in the authentication process and is used in next steps.
plugins	JSON-array that contains the list of available plugins.
status	Current status of logon process. To view the full list of response statuses, see the About Login Process chapter.

Chains JSON-object has the following parameters.

Parameter name	Parameter description
methods	JSON-array of authentication methods represented into chain. All of these methods should be successful for login.
is_trusted	Boolean flag which determines whether it is a trusted chain or not.
name	Name of the authentication chain.

After starting the logon process, authentication data for the current authentication method in the chain should be sent. To send authentication data, use the following resource with URI:

```
/logon/{logon_process_id}/do_logon
```

Resource is provided by HTTP POST. It has the following parameters.

URI parameter name	Parameter description
logon_process_id	Logon process identifier. Identifier of started logon process.

Resource accepts data as JSON-object with the following parameters.

Parameter name	Parameter description
response	JSON-object with authentication data for method. This object is different for all authentication methods.
endpoint_session_id	Endpoint session identifier. Identifier of endpoint, which works with logon session.

Resource returns JSON-object with data of authentication result.

Parameter name	Parameter description
chains	JSON-object which describes available chains for login.
current_method	Current authentication method which is used for authentication.
logon_process_id	Logon process identifier, identifier of current logon process. This identifier is constant in the authentication process and is used in next steps.
completed_methods	JSON-array which contains the list of completed authentication methods supported by server.
status	Current status of logon process. To view the full list of response statuses, see the About Login Process chapter.
repo_id	Repository identifier, server identifier of users' repository.
user_name	User name.
app_id	Event server identifier
user_id	User server identifier.
msg	Message about process which contains more information about authentication process or about errors.
login_session_id	Login session identifier, identifier of success authentication session.
plugins	JSON-array that contains the list of available plugins.

Each authentication method has a different number of challenge-response steps. Check method description for more information.

Example

User "JSmith" from repository "AUTHASAS" tries to login to Template Management form endpoint with session identifier "Yhmvrpd1nclIER1puxkdBuup0KgeZJwE".

```
HTTP POST
https://authserver.example.com/api/v1/logon

Request
```

```
{"method_id":"LDAP_PASSWORD:1",
"user_name":"AUTHASAS\\JSmith",
"application":"TemplatesManagement",
"endpoint_session_id":"Yhmvrpd1nclIER1puxkdBuup0KgeZJwE "}
```


Response


```
{"chains":[{"methods":["TOTP:1"],
"is_trusted":true,
"name":"TOTP"},
{"methods":["SMS_OTP:1"],
"is_trusted":true,
"name":"SMS OTP"},
{"methods":["SMARTPHONE:1"],
"is_trusted":true,
"name":"Smartphone Out-of-Band"},
{"methods":["VOICE:1"],
"is_trusted":  true,
"name":"Voicecall"}],
"completed_methods":[],
"current_method":"LDAP_PASSWORD:1",
"msg":  "Process started",
"logon_process_id":"Hok9BVH6w02VwE8KCIsQXe9fwp0UHx8Z",
"plugins":  [],
"status":  "MORE_DATA"}
```


HTTP POST
https://authserver.example.com/api/v1/logon/Hok9BVH6w02VwE8KCIsQXe9fwp0UHx8Z/do_logon

Request


```
{"response":{  "answer":"P@$sW0Rd"  },
"endpoint_session_id":"Yhmvrpd1nclIER1puxkdBuup0KgeZJwE"}
```



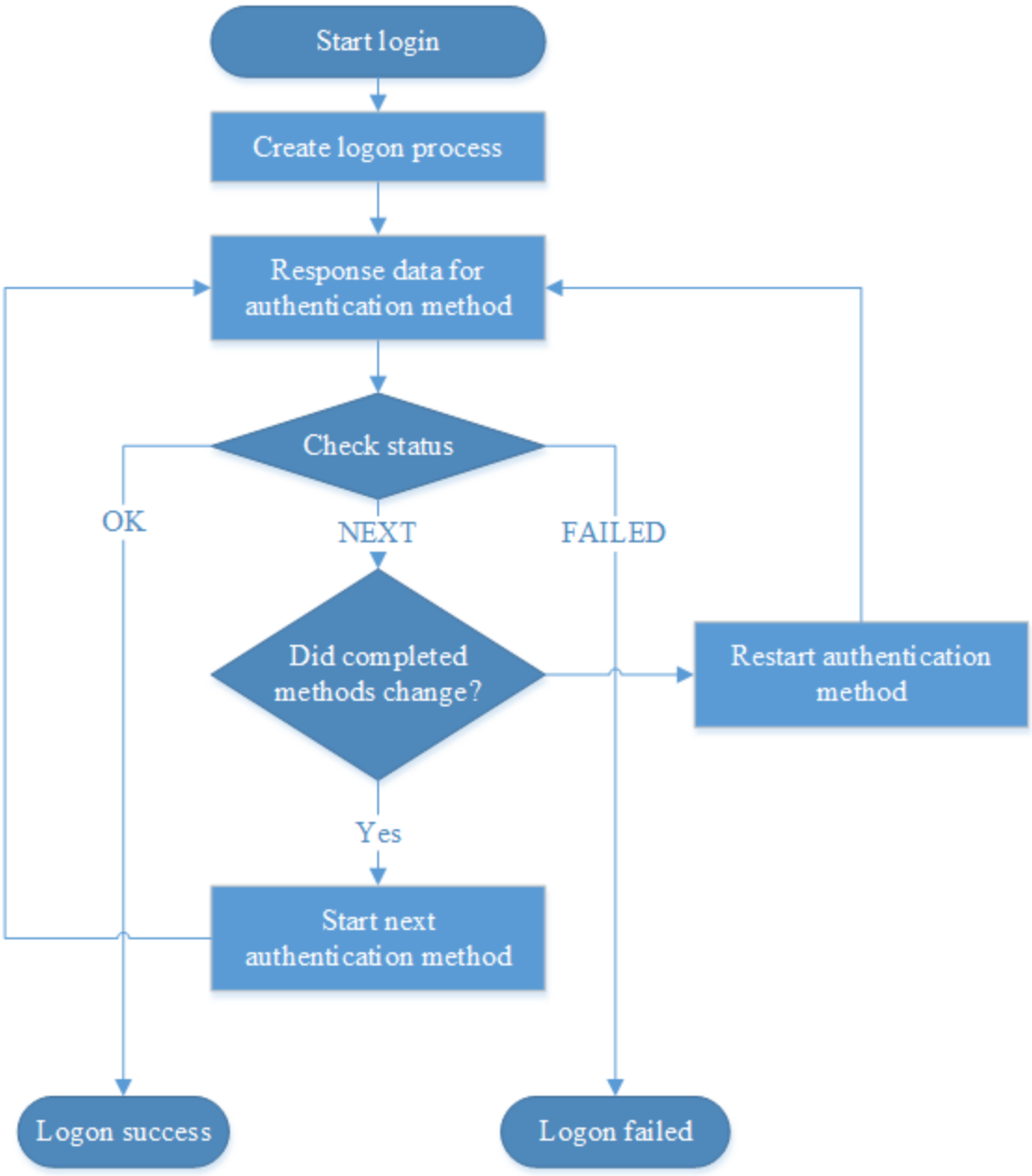
```
{"chains":[{"methods":[  "TOTP:1"  ],
"is_trusted":true,
"name":"TOTP"},
{"methods":[  "SMS_OTP:1"  ],
"is_trusted":true,
"name":"SMS OTP"},
{"methods":[  "SMARTPHONE:1"  ],
"is_trusted":true,
"name":"Smartphone Out-of-Band"},
{"methods":[  "VOICE:1"  ],
"is_trusted":  true,
"name":"Voicecall"}],
"current_method":"LDAP_PASSWORD:1",
```

```
"application_name":"TemplatesManagement ",
"logon_process_id":"Hok9BVH6w02VwE8KCIsQXe9fwp0UHx8Z",
"completed_methods":["LDAP_PASSWORD:1"],
"status":"OK",
"repo_id":"73d7b1f586c911e4b1a300155d62a8b3",
"user_name":  "AUTHASAS\\JSmith",
"app_id":"842c265912a951c1256200145d71c9b9",
"user_id":"750b199486c911e4885200155d62a8b3",
"msg":  "Welcome!",
"login_session_id":"QnBdD5VCqoKksNVadQRJXVrLCn4M1Wfs",
"plugins":["LdapRules"]}
```


Providing Chained Authentication

Chained authentication is like a simple authentication, it uses all methods from simple authentication and has same JSON-object and other parameters. The differences are in step count, chained authentication combine many authentication methods and all of them should be passed successfully.

The logic of the chained authentication is displayed on the next schema. In this login, you should start next authentication method until you get successful or unsuccessful result. When the logon process is started, it will be required to choose authentication method and provide data for this authentication method at first iteration of logon process.



API for chained login has an additional resource for starting next authentication method, this resource has the following URI:

```
/logon/{logon_process_id}/next
```

Resource is provided by HTTP POST. It has the following URI parameter.

URI parameter name	Parameter description
logon_process_id	Logon process identifier, identifier of the started logon process.

Resource accepts JSON-object which contains information about the next authentication method.

Parameter name	Parameter description
endpoint_session_id	Endpoint session identifier. Identifier of endpoint which will create logon session. For more information, see the Creating Endpoint Session chapter.
method_id	Authentication method identifier. Identifier of next method from authentication methods list that is supported by server. This parameter defines which method will be used for authentication. For more information on the supported authentication methods, see the About Login Process chapter.

Resource returns JSON-object equal to JSON-object returned form resource for starting logon process. For more information, see the Main Process chapter. Other requests to resources that are used for chained login are described in the Providing Simple Authentication Using One Method in a Chain chapter.

Example

The user with username "JSmith" from users' repository "AUTHASAS" tries to log in from endpoint with session identifier "mPlTOjsNyfdbmKSDkVEWORpc61snlLwo". The user uses chain with LDAP password and RADIUS password. He/she authenticates to the NAM application.

```
HTTP POST
https://authserver.example.com/api/v1/logon

Request
        {"method_id":"LDAP_PASSWORD:1",
        "user_name":"AUTHASAS\\JSmith",
        "application":"NAM",
        "endpoint_session_id":"mPlTOjsNyfdbmKSDkVEWORpc61snlLwo"}

Response

        {"chains":[{"methods":["LDAP_PASSWORD:1",
        "RADIUS:1"],
        "is_trusted":true,
        "name":"RADIUS and LDAP"}],
        "completed_methods":[],
        "current_method":"LDAP_PASSWORD:1",
        "msg":"Process started",
        "logon_process_id":"fBRfAwKhSqw3CPAdmMAM4WmcRup8JlsF",
        "plugins":[],
        "status":"MORE_DATA"}
```

HTTP POST
https://authserver.example.com/api/v1/logon/fBRfAwKhSqw3CPAdmMAM4WmcRup8JlsF/do_logon

Request


```
{"response":{"answer":"P@S$w0rD"},
"endpoint_session_id":"mPlTOjsNyfdbmKSDkVEWORpc61snlLwo"}
```


Response


```
{"user_name":"AUTHASAS\\JSmith",
"chains":[{"methods":["LDAP_PASSWORD:1",
"RADIUS:1"],
"is_trusted":true,
"name":"RADIUS and LDAP"}],
"current_method":"LDAP_PASSWORD:1",
"msg":"Continue with next login method",
"logon_process_id":"fBRfAwKhSqw3CPAdmMAM4WmcRup8JlsF",
"completed_methods":["LDAP_PASSWORD:1"],
"user_id":"750b199486c911e4885200155d62a8b3",
"plugins":["LdapRules"],
"status":"NEXT",
"repo_id":"73d7b1f586c911e4b1a300155d62a8b3"}
```


HTTP POST
https://authserver.example.com/api/v1/logon/fBRfAwKhSqw3CPAdmMAM4WmcRup8JlsF/next

Request


```
{"method_id":"RADIUS:1",
"endpoint_session_id":"mPlTOjsNyfdbmKSDkVEWORpc61snlLwo"}{"chains":[{"methods":["LDAP_PASSWORD:1",
"RADIUS:1"],
"is_trusted":true,
"name":"RADIUS and LDAP"}],
"completed_methods":["LDAP_PASSWORD:1"],
"current_method":"RADIUS:1",
"msg":  "Process started",
"logon_process_id":"fBRfAwKhSqw3CPAdmMAM4WmcRup8JlsF",
"plugins":[],
"status":  "MORE_DATA"}
```


HTTP POST
https://authserver.example.com/api/v1/logon/fBRfAwKhSqw3CPAdmMAM4WmcRup8JlsF/do_logon

Request

```
{"response":{"answer":"R@D!u$PaS$W0rd"},
"endpoint_session_id":"mPlTOjsNyfdbmKSDkVEWORpc61snlLwo"}
```

Response

```
{"chains":[{"methods":["LDAP_PASSWORD:1",
"RADIUS:1"],
"is_trusted":true,
"name":"RADIUS and LDAP"}],
"current_method":"RADIUS:1",
"application_name":"NAM",
"logon_process_id":"fBRfAwKhSqw3CPAdmMAM4WmcRup8JlsF",
"completed_methods":["LDAP_PASSWORD:1", "RADIUS:1"],
"status":  "OK",
"repo_id":"73d7b1f586c911e4b1a300155d62a8b3",
"user_name":  "AUTHASAS\\Administrator",
"app_id":  "WindowsLogon",
"user_id":"750b199486c911e4885200155d62a8b3",
"msg":"Welcome!",
"login_session_id":"tfT87rgyRLnvaY5oQXCWbHzpnvpJQZs8",
"plugins":  ["LdapRules"]}
```

Reading Available Chains

Before you start the logon process, you should get available authentication chains for this end-point and user. API has the following resource with URI:

```
/logon/chains?user_name={user_name}&application={application}&is_trusted={is_trusted}&endpoint_session_id={endpoint_session_id}
```

Resource is provided by HTTP GET. It has the following parameters.

URI parameter name	Parameter description
user_name	User name (optional parameter). In case of using this parameter the server will return chain only for this user. If the parameter is not used, the server will return chains for all users.
application	Event identifier from server supported events list. The evens list is described in the About Login Process chapter.
is_trusted	Boolean flag (optional parameter). In case of using this parameter the server will

	return only trusted or untrusted chains. If the parameter is not used, all available chains will be returned by the server.
end-point_session_id	Endpoint session identifier. Identifier of endpoint session which wants to get the chains. For more information, see the Creating Endpoint Session chapter.

Resource returns JSON-object which contains the list of available chains.

Parameter name	Parameter description
chains	Chain. This parameter is used for wrapping chains JSON-object array.

Example

Endpoint with session "2NC69uuu0r63fYpqamMSJfYEVjIkSsbv" tries to get all trusted chains available for NAM event.

```
HTTP GET
https://authserver.example.com/api/v1/logon/chains?application=NAM&endpoint_
session_id=2NC69uuu0r63fYpqamMSJfYEVjIkSsbv&is_trusted=true

Response

        {"chains":[{"methods":["LDAP_PASSWORD:1",
        "RADIUS:1"],
        "is_trusted":true,
        "name":"RADIUS and LDAP"},
        {"methods":["LDAP_PASSWORD:1",
        "TOTP:1"],
        "is_trusted":true,
        "name":"TOTP and LDAP"},
        {"methods":["LDAP_PASSWORD:1",
        "HOTP:1"],
        "is_trusted":true,
        "name":" HOTP and LDAP"}]}
```

Deleting Logon Process

To delete the logon process, use the following resource with URI:

```
/logon/{logon_process_id}?endpoint_session_id={endpoint_session_id}
```

Resource is provided by HTTP DELETE. It has the following URI parameters.

URI parameter name	Parameter description
logon_process_id	Logon process identifier. Identifier of process for deleting.
endpoint_session_id	Endpoint session identifier. Identifier of endpoint session which wants to delete the login process. For more information, see the Creating Endpoint Session chapter.

The resource does not return any data. If deletion is successful, the method will return the HTTP 200 status.

Example

Endpoint with session identifier "Sm7hACXcyvfl4ApoQ17g5QntE0q1ZW1K" deletes logon process with identifier "wLlxiBUCDDiTRJtL0xJPOeZpWfNcWfoH".

```
HTTP DELETE
https://authserver.example.com/api/v1/logon/wLlxiBUCDDiTRJtL0xJPOeZpWfNcWfoH?endpoint_session_id=Sm7hACXcyvfl4ApoQ17g5QntE0q1ZW1K

Response

HTTP 200
```

Working with User's Data

In this chapter:

About User's Data

Users' data is data that is stored for a specific event for a user. After logon, user can get access to the user's data. Users' data is a JSON-object with a custom structure and parameter names. Each event has different users' data. Check умуты description in the About Login Process chapter.

Reading User's Data

To read user's data, use the following resource with URI:

```
/users/{user_id}/appdata/{app_id}/{data_parameter}?login_session_id={login_session_id}
```

Resource provided by HTTP GET. It has the following parameters.

URI parameter name	Parameter description
user_id	User identifier. Identifier of user which wants to get assigned application data.
app_id	Event identifier. Identifier of event which data will be read.
data _parameter	User's data parameter (optional parameter). If this parameter is used, resource will return only this parameter from user's data.
login_session_id	Login session identifier. Identifier of user login session. For more information, see the Providing Authentication chapter

Resource returns user's data as JSON-object. Each event has different data format. For more information on events, see the About Login Process chapter.

Parameter name	Parameter description
appdata	Container for application data JSON-object. If method uses application data parameter, this container will contain only application data parameter.

Example

User with identifier "6f4db9228c2711e4bb4100155d62a8b3" with login session identifier "cHIIvvAS4KteAG6MnVzJXx1I7TjVtnxP" gets application data for application "WindowsLogon" and gets only "test1" parameter from application data.

```
HTTP GET

https://authserver.example.com/api/v1/users/6f4db9228c2711e4bb4100155d62a8b3/
appdata/WindowsLogon?login_session_id=cHIIvvAS4KteAG6MnVzJXx1I7TjVtnxP


Response

        {"appdata":{"test1":"value 1",
        "test3":  "value 3",
        "test2":  "value 2"}}

HTTP GET

https://authserver.example.com/api/v1/users/6f4db9228c2711e4bb4100155d62a8b3/
appdata/WindowsLogon/test1?login_session_id=cHIIvvAS4KteAG6MnVzJXx1I7TjVtnxP


Response

        {"appdata":{"test1":"value 1"}}
```

Modifying User's Data

To modify user's data, use the following resource with URI.

```
/users/{user_id}/appdata/{app_id}
```

Resource is provided by HTTP PATCH and has the following parameters.

URI parameter name	Parameter description
user_id	User identifier. The method will add application data to this user.
app_id	Event identifier. Resource will add data for this event. User should be logged on to this event.

Resource accepts JSON-object with user's data.

Parameter name	Parameter description
appdata	User's data container. This parameter contains all user's data.
login_session_id	Login session identifier. Identifier of user login session.

Resource does not return any data. If the modification is successful, the HTTP 200 status will be returned.

Resource has flexible behavior. Resource can update or modify data. E.g., if user has no data, resource will add data for user for this event. If user has some data, resource update parameters will be represented in request. It will be required to add new parameters, if these parameters are not represented into current user's data. Parameters which set to null into request should be deleted.

Example

User with identifier "6f4db9228c2711e4bb4100155d62a8b3" and login session with identifier "BFldNh3rLx39gjmun65gwJpLETjGF5fO" adds user's data to event "WindowsLogon".

```
HTTP PATCH

https://authserver.example.com/api/v1/users/6f4db9228c2711e4bb4100155d62a8b3/
appdata/WindowsLogon

Request

        {"appdata":{"test1":"value 1",
        "test2":"value 2",
        "test3":"value 3"},
        "login_session_id":"BFldNh3rLx39gjmun65gwJpLETjGF5fO"}

Response

HTTP 200
```

Deleting User's Data

To delete user's data, use the following resource with URI:

```
/users/{user_id}/appdata/{app_id}/{data_parameter}?login_session_id={login_ses-
sion_id}
```

Resource is provided by HTTP DELETE. It has the following parameters.

URI parameter name	Parameter description
user_id	Event identifier. Identifier of event which data will be deleted.
app_id	Event identifier. Identifier of event which data will be deleted.
data_parameter	User's data parameter (optional parameter). If this parameter is used, resource will delete only this parameter from user's data. If this parameter is not used, resource will delete all application data.
login_session_id	Login session identifier. Identifier of user login session.

Resource does not return data. If deletion is successful, the HTTP 200 status will be returned.

Example

User with identifier "6f4db9228c2711e4bb4100155d62a8b3" with login session identifier "H0u1NPOpPs6foZonfAxoRRDxgTsSYuVM" deletes application data parameter "test1" for application "WindowsLogon" and deletes all application data for "WindowsLogon".

```
HTTP DELETE

https://authserver.example.com/api/v1/users/6f4db9228c2711e4bb4100155d62a8b3/
appdata/WindowsLogon/test1?login_session_id=H0u1NPOpPs6foZonfAxoRRDxgTsSYuVM


Response

HTTP 200

HTTP DELETE

https://authserver.example.com/api/v1/users/6f4db9228c2711e4bb4100155d62a8b3/
appdata/WindowsLogon?login_session_id=H0u1NPOpPs6foZonfAxoRRDxgTsSYuVM


Response

HTTP 200
```

Working with Login Session

In this chapter:

About Login Session

After user logon, system creates login session. This session should be used to access protected information from the server. E.g., reading event data requires login session identifier. API allows reading the login session information and deleting the session.

Reading Information about Login Session

To read information about an applicable login session, use the following resource with URI:

```
/logon/sessions/{logon_session_id}?endpoint_session_id={endpoint_session_id}
```

Resource is provided by HTTP GET. It has the following parameters.

URI parameter name	Parameter description
logon_session_id	Logon session identifier. Identifier of session for reading.
endpoint_session_id	Endpoint session identifier. Identifier of endpoint session.

Resource returns JSON-object which contains information about logon session.

Parameter name	Parameter description
application_ name	Event name. Name of the event that is linked to an applicable logon session.
repo_id	Users' repository identifier. Identifier of users' repository that is used for user logon.
user_id	User identifier. Identifier of user that has created this logon session.
repo_obj_id	For later usage.
sid	Logon session identifier.

user_name	User name. Name of the user that created logon session.
app_id	Event identifier. Identifier of event which creates an applicable session.

Example

We try to get information about the session with identifier "9hGUd3xuKE6VX0I8bVMWNeX1zNm0QfNd" from endpoint with session identifier "Wpeg2ek8IvsNF1hFZTXqYjPYvhCdUsZd".

```
HTTP GET

https://authserver.example.com/api/v1/logon/sessions/9hGUd3xuKE6VX0I8bVMWNeX1
zNm0QfNd?endpoint_session_id=Wpeg2ek8IvsNF1hFZTXqYjPYvhCdUsZd

Response

        {"application_name":"NAM",
        "repo_id":"6e0b696e8c2711e4bd9600155d62a8b3",
        "user_id":"6f4db9228c2711e4bb4100155d62a8b3",
        "repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
        "sid":"9hGUd3xuKE6VX0I8bVMWNeX1zNm0QfNd",
        "user_name":"AUTHASAS\\JSmith",
        "app_id":"WindowsLogon"}
```

Deleting Login Session

To delete login session, use the following resource with URI:

```
/logon/sessions/{logon_session_id}?endpoint_session_id={endpoint_session_id}
```

Resource is provided by HTTP DELETE. It has the following parameters.

Table 28. URI parameters for deleting login session

URI parameter name	Parameter description
logon_session_id	Logon session identifier. Identifier of session to be deleted.
endpoint_session_id	Endpoint session identifier. Identifier of endpoint session.

Method does not return any data. If deletion is successful, the HTTP 200 status will be returned.

Example

We are going to delete the logon session with identifier "4Keuv7THWdSMR1H7mPc34O4mGoxTP0TP" with endpoint session identifier "Ex3hqCoF5A2ARWGv221AmPtEkWQDUi0U".

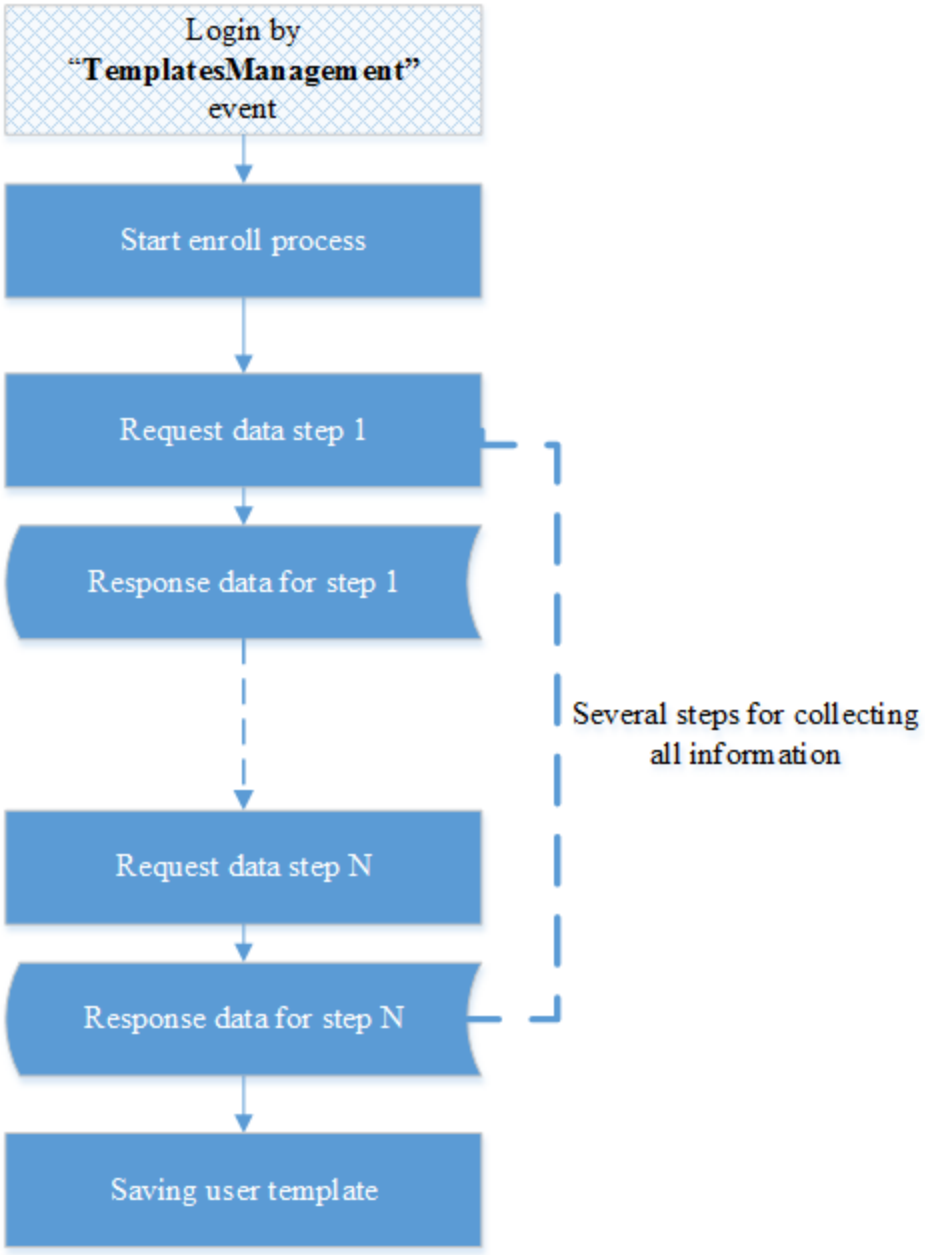
```
HTTP DELETE

https://authserver.example.com/api/v1/logon/sessions/4Keuv7THWdSMR1H7mPc34O4m
GoxTP0TP?endpoint_session_id=Ex3hqCoF5A2ARWGv221AmPtEkWQDUi0U

Response

HTTP 200
```

Working with Enrollment

In this chapter:

About Enroll Process

Enroll process collects information for user templates creation. User templates can be created by several steps with enroll process. Enroll process is a wizard to user templates. Each user can create user templates into enroll process, and administrator can assign enroll process results to another users. The user should be logged by "TemplatesManagement" event to start the enroll process. Enroll process can be described by simple schema; it has several steps for collecting information from user and saving the user template.



Starting Enroll Process

To start the enroll process, make request to create process to the following resource with URI:

`/enroll`

Resource is provided by HTTP POST and accepted JSON object with this parameters.

Parameter name	Parameter description
login_session_id	Login session identifier. User should be logged to “TemplatesManagement” to create enroll process.
method_id	Authentication method identifier. Identifier of method form authentication methods list that is supported by server.

Resource returns enroll process identifier. It is represented as JSON object with this parameter.

Parameter name	Parameter description
enroll_process_id	Enroll process identifier. It is used for next enroll process step.

Example

User with login session identifier “Iz4awDMiRYcZh55SYt8awBz3Fcl1vikJ” starts the enroll process for the security question authentication method.

```
HTTP POST

https://authserver.example.com/api/v1/enroll

Request
        {"method_id":"SECQUEST:1",
        "login_session_id":"Iz4awDMiRYcZh55SYt8awBz3Fcl1vikJ"}

Response

        {"enroll_process_id":"WqQd4TwxPCz7q1tAKrWGzCtajg7Iav14"}
```

Providing Data Into Enroll Process

Data should be provided into enroll process after this process is started. Resource for providing data has URI:

```
enroll/{enroll_process_id}/do_enroll
```

Resource is provided by HTTP POST. It has the following parameter.

URI parameter	Parameter description
enroll_process_id	Enroll process identifier. Identifier of current enroll process. Starts enroll process to get this identifier.

Resource accepts JSON object.

Parameter name	Parameter description
login_session_id	Login session identifier. User should be logged to "TemplatesManagement" to provide enroll data.
response	Object with specific data for method. Data varies for each authentication method.

Resource returns the result for accepting the enroll data. Result can be different for each authentication method. All responses contain parameters which are the same for all authentication methods.

Parameter name	Parameter description
method_id	Authentication method identifier. Identifier of method from authentication methods list that is supported by the server. This parameter shows which authentication method currently works with enroll data.
status	Current status of enroll process.
msg	Message from enroll process. This parameter is used for providing more information about enroll process.

Enroll process can have one of the following statuses.

Status name	Status description
OK	This status indicates that the enroll process has collected all necessary data and you can create or update the user template.
MORE_DATA	This status indicates that enroll process is waiting for more data from user. Specific data for enroll process should be provided.
FAILED	This status indicates that the enroll process has failed, for more information check message parameter from response. The enroll process should be started once again.

Each enroll process is one or more steps for collecting all necessary data for the authentication method. Each authentication method has a different number of steps and data, for more information on authentication method description, see the About Login Process chapter.

Example

We provide data for Security Question authentication method, this method has two steps: first step is getting security question list from server, second step is providing answers for each security question. We have enroll process with identifier

"WqQd4TwxPCz7q1tAKrWGzCtajg7Iav14", user logged to "TemplatesManagement" application with session identifier "7Ge4BCGDLKPyG5b6Mp7PBcKUsQouhpdX". On the first step we will get security question list.

```
HTTP POST

https://authserver.example.com/api/v1/enroll/WqQd4TwxPCz7q1tAKrWGzCtajg7Iav14
/do_enroll

Request

        {"response":{},
        "login_session_id":"7Ge4BCGDLKPyG5b6Mp7PBcKUsQouhpdX"}

Response

        {"questions":{"1":"What is your dog name?",
        "0":   "What is your favorite song?"},
        "method_id":"SECQUEST:1",
        "msg":"Waiting for answers...",
        "status":"MORE_DATA"}
```

We got question and servers change status to "MORE_DATA", this means that the server is waiting for the answers to the questions. Now we will send answers, it will be the second step.

```
HTTP POST

https://authserver.example.com/api/v1/enroll/WqQd4TwxPCz7q1tAKrWGzCtajg7Iav14
/do_enroll

Request

        {"response":{"answers":{"1":"Spotty",
        "0":"Yesterday"}},
        "login_session_id":"7Ge4BCGDLKPyG5b6Mp7PBcKUsQouhpdX"}

Response

        {"method_id":"SECQUEST:1",
        "status":"OK",
        "msg":""}
```

After the second step the server accepted our data and returned status "OK". It means that enrollment was successful.

Deleting Enroll Process

To delete enroll process, use the following resource with URI:

```
/enroll/{enroll_process_id}?login_session_id={login_session_id}
```

Resource is provided by HTTP DELETE. It has the following URI parameters.

URI parameter name	Parameter description
enroll_process_id	Enroll process identifier. Identifier of enroll process which will be stopped.
login_session_id	Login session identifier. User should be logged to "TemplatesManagement" to delete enroll process.

This resource does not return any data. If the enroll process is successfully deleted, the HTTP 200 status will be returned.

Example

A user with login session identifier "4T1WEpCKumaIonjDBhoXbWkHhGg5Tk7Z" deletes enroll process with identifier "AfAXRqxrI1K2jgeJzrarD0NxllhpY9UW".

```
HTTP DELETE

https://authserver.example.com/api/v1/enroll/AfAXRqxrI1K2jgeJzrarD0NxllhpY9UW
?login_session_id=4T1WEpCKumaIonjDBhoXbWkHhGg5Tk7Z

Response

HTTP 200
```

Working with User Templates

In this chapter:

About User Templates

User templates contain authentication information associated with users. Each template is linked to a user and to an authentication method. When users try to logon using a specific authentication method, the server finds the associated user template and provides authentication. Users cannot use authentication methods without associated user templates. The enrollment process creates user templates. For more information, see the Working with Enrollment chapter. Each authentication method stores different information in the templates. The authentication method can update a user's template in the authentication process. Users or administrator can also change templates manually, user can edit only his own templates.

API provides resources for working with user's templates: creating, updating, reading and deleting.

Getting User Templates

To read user templates, use the following resource with URI:

```
/users/{user_id}/templates?login_session_id={login_session_id}
```

Resource is provided by HTTP GET.

URI parameter name	Parameter description
user_id	User identifier. This identifier will be used for getting associated user templates. Administrator can use any user identifier, user should use only his own identifier.
login_session_id	Login session identifier. User's login session identifier used for check-

	ing user access. User should be logged to “TemplatesManagement” to get access to user template.

Resource returns the list of user templates. The list is presented as array of JSON object.

Parameter name	Parameter description
templates	This parameter is used for wrapping array of JSON object with user templates.
id	Identifier of user template.
method_id	Identifier of authentication method.
is_enrolled	Boolean flag. If the flag is true, the template will be enrolled and can be used for authentication. If the flag is false, the template will not be enrolled and cannot be used for authentication.
method_title	Authentication method title.
comment	Comment for user template.

Example

A user with identifier “958ca11a7fa511e4b6ab00155d62a8b3” and session identifier “TBsifFiE4UJCyMnIyTkmY21kFctSxdwe” gets his/her own list of two user templates.

```
HTTP GET

https://authserver.example.com/api/v1/users/958ca11a7fa511e4b6ab00155d62a8b3/
templates?login_session_id=TBsifFiE4UJCyMnIyTkmY21kFctSxdwe

Response

        {"templates":[{"id":"958ca11b7fa511e4a32200155d62a8b3",
        "method_id":"LDAP_PASSWORD:1",
        "is_enrolled":true,
        "method_title":"LDAP password",
        "comment":   "LDAP password template"},
        {"id":"959165a47fa511e4bd1500155d62a8b3",
        "method_id":"HOTP:1",
        "is_enrolled":true,
        "method_title":"HOTP",
        "comment":   "OATH HTOP template"}]}
```

Creating User Templates From Enroll Session

After a successful enroll process, you can assign enrolled data to the user template. To do it, use the following resource with URI:

```
/users/{user_id}/templates
```

Resource is provided by HTTP POST. It has the following parameter.

URI parameter name	Parameter description
user_id	User identifier. Identifier of the user who will be assigned to new template.

Administrator can assign enrolled data and create user templates for any user.

Resource accepts JSON-object with the following parameters.

Parameter name	Parameter description
enroll_process_id	Enroll process identifier. Identifier of success enroll process. The method will use data from an applicable enroll process.
login_session_id	Login session identifier. User should be logged to "TemplatesManagement" to create user template.
comment	Comment for the created user template.

Resource returns JSON-object with identifier of new user template.

Parameter name	Parameter description
auth_t_id	Authentication template identifier. Identifier of the created user template.

Example

User with identifier "e08c6b48810611e4b79300155d62a8b3" and login session identifier "33ebFAQBW1e1kTkKVzIRz5rf5dhv4OoE" creates a user template with enroll process identifier "nrXwfyy0l4QcXxYZNebzjs33rqS9UkG5".

```
HTTP POST

https://authserver.example.com/api/v1/users/e08c6b48810611e4b79300155d62a8b3/templates

Request
```

```
{"enroll_process_id":"nrXwfyy0l4QcXxYZNebzjs33rqS9UkG5",
"comment":"Authentication template comment",
"login_session_id":"33ebFAQBW1e1kTkKVzIRz5rf5dhv4OoE"}
```

Response

```
{"auth_t_id":"9df84602842f11e4817e00155d62a8b3"
```

Assigning User Templates to Another User

You can assign the created user templates to another user. This way a user can impersonate another user using his own authentication method. An example would be a user having 2 accounts, an admin account and a normal user account. He/she will only enroll authentication methods for his normal user account but will link his methods to his admin account. Now he can logon using the username of the admin account and the method of the normal account and this will be logged.

Another use case would be group accounts where multiple users are linked too.

To link an template to another user, use the following resource with URI:

`/users/{user_id}/templates`

Resource is provided by HTTP POST. It has the following parameter.

URI parameter name	Parameter description
user_id	User identifier. Identifier of the user who will be assigned to template.

Resource accepts JSON-object with the following parameters.

Parameter name	Parameter description
auth_t_id	Authentication user template identifier. Identifier of the user template which will be assigned to another user.
login_session_id	Login session identifier. User should be logged to "TemplatesManagement" to create user template.

Resource returns JSON-object with identifier of user's template.

Parameter name	Parameter description
auth_t_id	Authentication template identifier. Identifier of the user template.

This resource just links the user template to another user. New user cannot modify template. User can only read the assigned template.

Example

User with login session "sC7PfOjvHt7OMgIccEHjK7b9XGlGNSoj" assigns template with identifier "ba3cf01e845311e4841900155d62a8b3" to user with identifier "76cc2a62845411e4bd6c00155d62a8b3".

```
HTTP POST

https://authserver.example.com/api/v1/users/76cc2a62845411e4bd6c00155d62a8b3/templates

Request

        {"auth_t_id":"ba3cf01e845311e4841900155d62a8b3",
        "login_session_id":"sC7PfOjvHt7OMgIccEHjK7b9XGlGNSoj"}

Response

        {    "auth_t_id":"ba3cf01e845311e4841900155d62a8b3"    }
```

Updating User Templates

To update user templates, use the following resource with URI:

```
/users/{user_id}/templates/{auth_template_id}
```

Resource is provided by HTTP PUT. It has the following parameter.

URI parameter name	Parameter description
user_id	User identifier. Identifier of the user whose template will be updated.
auth_template_id	Authentication template identifier. Identifier of the template which will be updated.

Administrator can update user template for all users, other users can update his/her own templates.

Resource accepts JSON-object with the following parameters.

Parameter name	Parameter description
enroll_process_id	Enroll process identifier. Identifier of the successful enroll process. The method will use data from this enroll process to update data in current user template.
login_session_id	Login session identifier. User should be logged to "TemplatesManagement" to change user template.
comment	New comment for user template.

You can use one or both parameters for method, if you use enroll process identifier for method you will change user's template authentication data. When an template is updated, all linked users will need to use the updated template

Resource does not return data. If update is successful, the HTTP 200 status will be returned.

Example

User with identifier "e08c6b48810611e4b79300155d62a8b3" updates user template with identifier "ba3cf01e845311e4841900155d62a8b3" with new enrolled data form enroll process with identifier "l4rh0HvzJxNFBaO9d7PC4Uyq9YRc8EMY".

```
HTTP PUT

https://authserver.example.com/api/v1/users/e08c6b48810611e4b79300155d62a8b3/
templates/ba3cf01e845311e4841900155d62a8b3


Request

        {"enroll_process_id":"l4rh0HvzJxNFBaO9d7PC4Uyq9YRc8EMY",
        "comment":"Updated comment",
        "login_session_id":"86oe9ebDvJ08UJvlG0O14ZARm3wIYNTB"}

Response

HTTP 200
```

Deleting User Templates

To delete user templates, use the following resource with URI:

```
/users/{user_id}/templates/{template_id}?login_session_id={login_session_id}
```

Resource is provided by HTTP DELETE. It has the following parameters.

URI parameter name	Parameter description
user_id	User identifier. Identifier of the user whose template will be deleted.
template_id	Template identifier. Identifier of the template to be deleted.
login_session_id	Login session identifier. User's login session identifier that is used to check user access. User should be logged to "TemplatesManagement" to delete user template.

Administrator can delete any user templates, other users can delete only their own templates.

This method doesn't return any data. If deletion is successful, the HTTP 200 status will be returned. If an error occurs, the error description will be returned.

Example

User with user identifier "e08c6b48810611e4b79300155d62a8b3" and login session identifier "GsnGHDvqOiEdPR3KUhKqy2kZq7l4RCcC" tries to delete his own template with identifier "9df84602842f11e4817e00155d62a8b3".

```
HTTP DELETE

https://authserver.example.com/api/v1/users/e08c6b48810611e4b79300155d62a8b3/
templates/9df84602842f11e4817e00155d62a8b3?login_session_id=GsnGHDvqOiEdPR3KU
hKqy2kZq7l4RCcC

Response

HTTP 200
```

Errors

JSON-object with error information is returned on the error server.

Parameter name	Parameter description
errors	Array of errors JSON-objects. Each object contains information about error.
status	Current status.

Error object is JSON-object with the following parameters.

Parameter name	Parameter description
name	Error name.
location	Location where error occurs.
description	Error full description.

Example

The simple server error response looks in the following way.

```
{"status":"error",
"errors":[{"description":"You are logged to empty application.app_id. It is
not for appdata",
"location":"server",
"name":"AuError"}]}
```

Troubleshooting

HTTP status	Solution
400	Error in API method. Take error object from response to get more information.
404	API method not found on server.
434	Login session not found or expired. Update logon session
434	Endpoint session not found or expired. Update endpoint session.

Index

T

U

W