NetIQ Access Manager 4.3

REST API Guide

Contents

1. Introduction

This guide describes the RESTful APIs supported by the NetIQ Access Manager components. It includes step by step instructions for using these APIs.

Note: NTS will support the general Access Manager setup and any issues where the Access Manager endpoints do not return valid data. Any other code changes needed to integrate with Access Manager are outside the scope of traditional NTS support and need to go through the namsdk@novell.com channel.

2. API Overview

The APIs supported by the Access Manager can be broadly categorized as:

- **Administration APIs**

  The administration APIs help to automate the common administrative tasks. The Administration Console exposes these APIs. They are available in the Access Manager 4.3 release.

- **OAuth and OpenID Connect APIs**

  These APIs expose all the OAuth functionality as endpoints for registering clients, obtaining access tokens, and so forth. The Identity Server exposes these APIs. They are available from the Access Manager 4.1 release.

- **Component Statistics APIs**

  These monitoring APIs provide the statistics of the Identity Servers and Access Gateways. These are exposed by the individual devices. These APIs are a precursor to the Administration APIs for obtaining the device statistics. These continue to be available, but it is recommended to use the administration statistics API instead, as a single API provides details for all the devices.

The following sections provide in depth details about these APIs.

3 Administration APIs

3.1 Accessing the Administration APIs

These APIs are supported by the Administration Console. You can invoke these by using a browser or by using curl command in scripts to help automate the administrative tasks.

Base URL: https://<Administration Console DNS or IP>:<AC Port>/amsvc/v1/...

The AC port will be 8443 by default. However if the Administration Console is installed along with an Identity Server on the same system, then the port will be 2443.

Authentication: The APIs are protected by using Basic Authentication. You can use the Administration Console credentials for accessing the APIs as well. However, the API access differs in the following ways from the Administration Console:

- The username for accessing the APIs should be specified in the fully qualified format. For example, “cn=admin,o=novell”
- The user should have *full admin rights* to the Administration Console. These APIs do not support delegated admin access in this release.

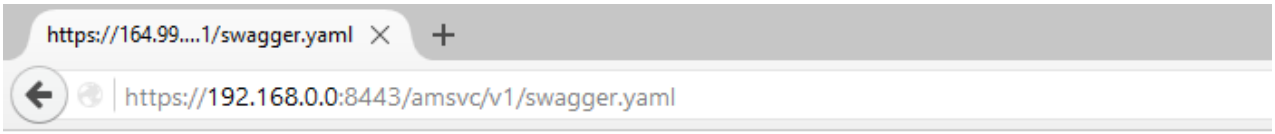
Response Format: It returns the data as XML by default. Set the Accept header to “application/json” to obtain the response in the JSON format.

3.2 Detailed API Documentation

Detailed documentation for all the Administration APIs is available as a YAML file on the Access Manager SDK website. It is also hosted by an installed Administration Console. You can view the YAML file in a user-friendly format using the Swagger UI tool.

Steps to view Swagger documentation:

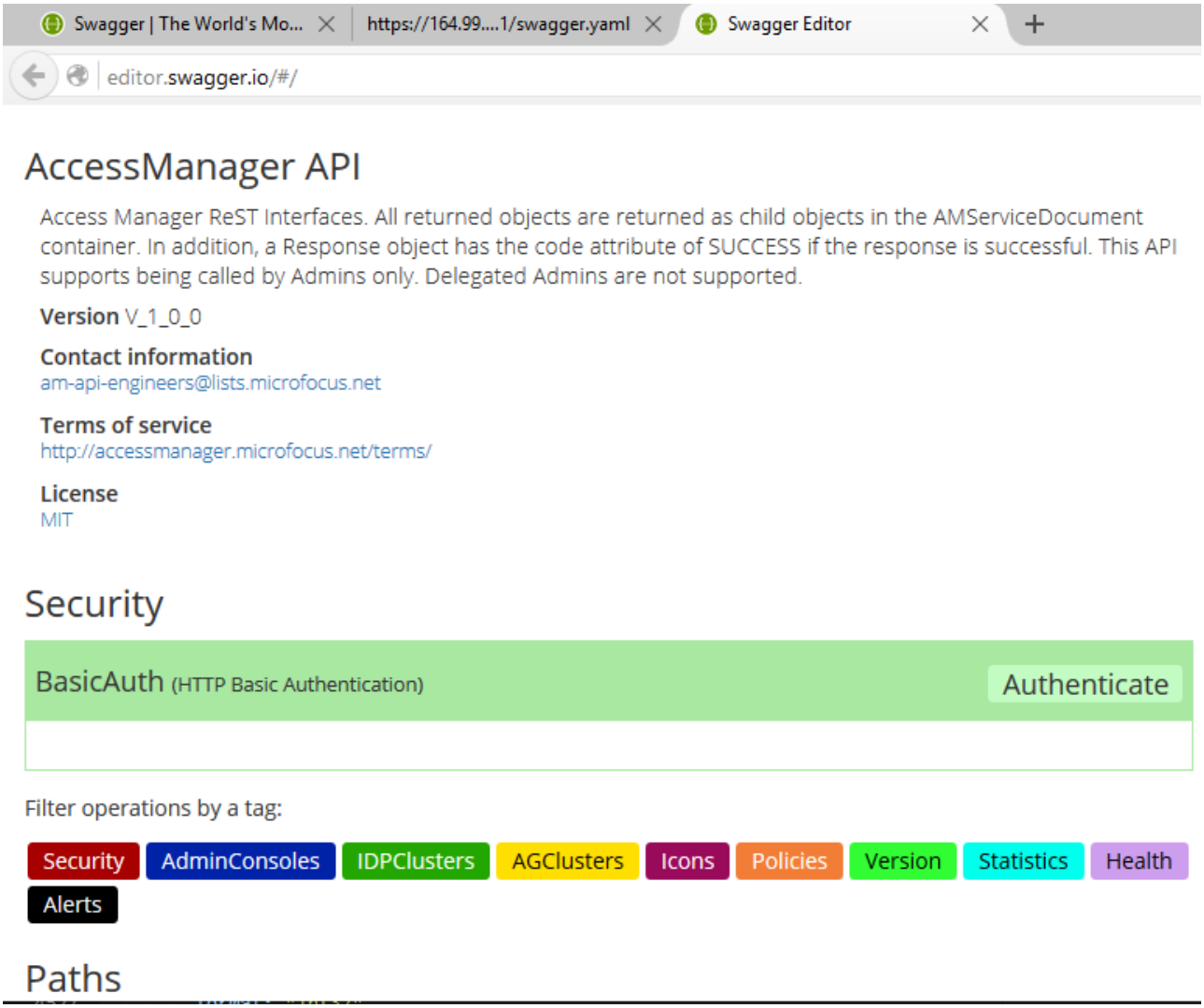
1. Get the YAML file from the SDK site or from an installed Admininstration Console as shown below.




```
---
swagger: "2.0"
info:
  description: "Access Manager ReST Interfaces.  All returned objects are returned\
    \ as child objects in the AMServiceDocument container. In addition, a Response\
    \ object has the code attribute of SUCCESS if the response is successful. This\
    \ API supports being called by Admins only.  Delegated Admins are not supported."
  version: "V_1_0_0"
  title: "AccessManager API"
  termsOfService: "http://accessmanager.microfocus.net/terms/"
  contact:
    email: "am-api-engineers@lists.microfocus.net"
  license:
    name: "MIT"
    url: "http://opensource.org/licenses/MIT"
host: "164.99.184.162:8443"
basePath: "/amsvc/v1"
tags:
- name: "Security"
- name: "AdminConsoles"
- name: "AGClusters"
- name: "Health"
- name: "IDPClusters"
- name: "Alerts"
- name: "Statistics"
- name: "Version"
- name: "Icons"
- name: "Policies"
schemes:
- "https"
paths:
  /adminconsoles:
    get:
```

2. Copy the content displayed on to the Swagger UI at http://editor.swagger.io

3. The Swagger tool parses the YAML input and generates an interactive documentation UI, client SDK, and so forth.



3.3 Administration API Use Cases

3.3.1 Get Device Health

This API returns the health of all the devices (Identity Servers and Access Gateways) in Access Manager. The health is returned for various levels - for the entire Access Manager, for every cluster, every device, and every service/component (remote web servers, data stores, and so forth).

This API can be used for integration with external systems like NOC to view the status of Access Manager devices and the remote web servers.

Sample Request:

Invoke a URL like https://192.168.0.0:8443/amsvc/v1**/health?expand=4**

The ‘expand’ parameter specifies the level of detail to be returned. Accepted values are 1,2,3 and 4, where 4 returns the maximum detail for all the devices.

Sample Response:

```
<amService xmlns="urn:novell:schema:am:service">
<health status="noReport" uri="https://192.168.0.0:8443/amsvc/v1/health">
    <idpClusterHealthList status="Green" total="1">
          <clusterHealth status="Green"
          uri="https://192.168.0.0:8443/amsvc/v1/idpclusters/SCC7c9nsp/health
          ">
                <instanceID>SCC7c9nsp</instanceID>
                <displayName>IDPCluster</displayName>
                <deviceHealthList total="1">
                      <deviceHealth status="Green"
                      uri="https://192.168.0.0:8443/amsvc/v1/idpclusters/SCC7
                      c9nsp/devices/idp-CC1B3FFB0BC40AD8/health">
                            <instanceID>idp-CC1B3FFB0BC40AD8</instanceID>
                            <displayName>192.168.0.6</displayName>
                            <serviceHealthList total="5">
                                  <serviceHealth status="Passed">
                                        <serviceName>Config Datastore</serviceName>
                                        <message>Operating properly</message>
                                  </serviceHealth>
                                  ...
```

Note:

This API returns the health information saved in the Administration Console. This data is refreshed every 5 minutes. Therefore it is sufficient to invoke this API every 5 minutes to get the latest health.

3.3.2 Get Device Statistics

This API returns the statistics for all the Identity Servers and Access Gateways in Access Manager.

Sample Request:

Send a GET request to a URL like https://192.168.0.0:8443/amsvc/v1**/statistics**.

Sample Response:

```
<amService xmlns="urn:novell:schema:am:service">
    <response code="SUCCESS"/>
    <statistics uri="https://192.168.0.0:8443/amsvc/v1/statistics">
       <idpClusterStatisticsList total="1">
          <clusterStatistics uri="https://
          192.168.0.0:8443/amsvc/v1/idpclusters/SCC7c9nsp/statistics">
                <instanceID>SCC7c9nsp</instanceID>
                <displayName>IDPCluster</displayName>
```

```
<deviceStatistics uri="https://
192.168.0.0:8443/amsvc/v1/idpclusters/SCC7c9nsp/devices/idp-
CC1B3FFB0BC40AD8/statistics">
   <instanceID>idp-CC1B3FFB0BC40AD8</instanceID>
   <displayName>192.168.0.6</displayName>
   <statisticList total="90">
      <statistic displayName="Cached
      Sessions">100</statistic>
      <statistic displayName="Historical Maximum Logins
      Served">890</statistic> ...
```

Note:

This API returns the statistics information saved in the Administration Console. It is refreshed every 10 minutes. Therefore it is sufficient to invoke this API every 10 minutes to get the latest statistics.

3.3.3 Refresh Metadata of SAML 2.0 Trusted Providers

Trusted providers periodically refresh their metadata. Some metadata repositories like InCommon.org publish an updated metadata every day. Therefore an automated approach for refreshing the metadata of all the service providers and updating the associated trusted root certificates helps relieve the administrator of this frequent chore and also ensures that the system is up to date for security reasons.

Steps:

1. Invoke the API to get the Identity Server clusters. Parse the response to get the cluster URL.
   Sample URL: https://192.168.0.0:8443/amsvc/v1/idpclusters
   Response:
   ...<idpCluster uri="https://192.168.0.0:8443/amsvc/v1/idpclusters/SCC7c9nsp"> ...

2. For each cluster URL, invoke the API to get the list of service providers or identity providers, depending on the provider that needs to be refreshed.
   https://192.168.0.0:8443/amsvc/v1/idpclusters/SCC7c9nsp**/serviceproviders** OR
   https://192.168.0.0:8443/amsvc/v1/idpclusters/SCC7c9nsp**/identityproviders**

3. Parse the response to get the URL of the trusted provider to be updated.
   Response Snippet:
   <serviceProvider
   uri="https://192.168.0.0:8443/amsvc/v1/idpclusters/SCC7c9nsp/serviceproviders/STSPr9spkh">
   <displayName>of365</displayName>
   <protocol>saml2</protocol>
   ...
4. Invoke the metadata refresh API to apply the updated metadata as explained below.
5. Invoke the trusted roots API to add the root CA of the signing certificate specified in the metadata. This step is only needed if the certificate has changed. For more information, see the next section.

6. Invoke the Apply changes API to send these changes to the Identity Servers in that cluster.
   Send PUT request to the cluster URL
   https://192.168.0.0:8443/amsvc/v1/idpclusters/SCC7c9nsp/
   with input
   { “update” : “all” }

Sample Script

A sample script that implements all the steps listed above is available on the NetIQ Cool Solutions website.

Sample Request:

URL format: <trusted provider URL in step 3>/metadata

Send a PUT request to https://192.168.0.0:8443/amsvc/v1/idpclusters/SCC7c9nsp/serviceproviders/STSPr9spkh/metadata with metadata as input. Metadata can be specified as a text or a URL.

Sample text input: (Note: metadata text must be URL encoded)

```
{
"metadata"     :
"%3C%3Fxml%20version%3D%221.0%22%20encoding%
3D%22UTF-8%22%20%3F%3E%3Cmd%3AEntityDescriptor%20xmlns%3
Amd%3D%22urn%3Aoasis%3Anames%3Atc%3ASAML%3A2.0%3Ametadata%
22%20ID%3D%22idXMuLnBrALGXkMAMUXd9WXvS0aEI%22%20entityID%
3D%22https%3A%2F%2Fpriyankasb.blr.novell.com%2Fnidp%2Fsaml
2%2Fmetadata%22%3E%3Cds%3ASignature%20xmlns%3Ads%3D%22http
%3A%2F%2Fwww.w3.org%2F2000%2F09%2Fxmldsig%23%22%3E%0A%3Cds
..............
%3C%2Fmd%3AEntityDescriptor%3E"
}
```

Sample metadata URL input:

```
{
"metadata"    : "https://164.99.87.129:8443/nidp/saml2/metadata"
}
```

Response:

```
200 OK
```

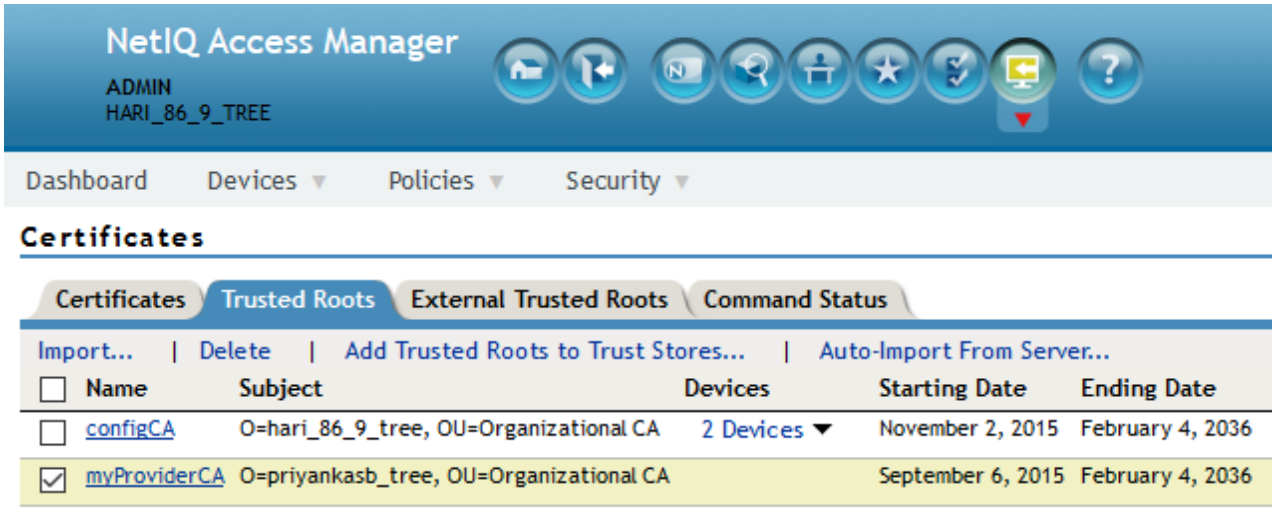
3.3.4 Import Trusted Root Certificates

You can use this API to import a trusted root certificate. This is usually used in conjunction with the above metadata refresh API.

Sample Request:

Send a PUT request to
https://192.168.0.0:8443/amsvc/v1/security/trustedroots/myProviderCA

where “myProviderCA” is the trusted root name displayed on the Administration Console.



The URL encoded public CA certificate must be specified as input.

```
{
"certificate" : "-----BEGIN%20CERTIFICATE-----
                %0AMIIFNDCCBBygAwIBAgIkAhwR%......
                -----END%20CERTIFICATE-----"
}
```

Response:

200 OK

Note:

- The certificate must be URL encoded
- Apply changes to all the devices that might use this certificate.

3.3.5 Renew Certificates

You can use this API to renew the certificates that are available through the Administration Console. Specify the certificate name and the certificate content as input to the API.

Sample Request:

Send a PUT request to https://192.168.0.0:8443/amsvc/v1/security/certificates/test-signing

with the following input, where the intermediate certificates are optional:

```
{
"entityCertificate" :"-----BEGIN%20CERTIFICATE-----%0AMIIFDjCCA%2FagAwIBAg
    IkAhwR%2F6b94LzCZy%2BK8kSqu-----END%20CERTIFICATE-----",
"rootCertificate" :"-----BEGIN%20CERTIFICATE-----%0AMIIFDjCCA%2FagAwIBAg
```

```
    IkAhwR%2F6b94LzCZy%2BK8kSqu-----END%20CERTIFICATE-----",
"intermediateCertificate1" :"-----BEGIN%20CERTIFICATE-----
    %0AMIIFDjCCA%2FagAwIBAgb94LzCZy%2BK8kSqu-----END%20CERTIFICATE-----",
"intermediateCertificate2" :"-----BEGIN%20CERTIFICATE-----
    %0AMIIFDjCCA%2FagAwIBAgzCZy%2BK8kSqu-----END%20CERTIFICATE-----"
}
```

Note:

1. An update is required for all devices using that certificate. Updating the connector certificate requires tomcat restart.
2. Certificate specified must be the PEM formatted public certificate and must be URL encoded.
3. Entire chain must be specified. Entity Cert → Intermediate 1 → Intermediate 2 → Root CA where → indicates that the Entity certificate was signed by Intermediate 1 and so on.

3.3.6 Manage User Sessions

These APIs allow to fetch and terminate all the active sessions of a given user.

Steps:

1. Invoke the API to get all the Identity Server clusters.

   Sample URL: https://192.168.0.0:8443/amsvc/v1/idpclusters

2. Parse the response to get the URL for each Identity Server cluster.

   …<idpCluster uri="https://192.168.0.0:8443/amsvc/v1/idpclusters/SCC7c9nsp">…

3. Invoke the URL of a cluster to get the sessions for a user.

   URL Format: <IDP cluster URL>/sessions?userid=<user name>

4. Repeat step 3 for other clusters as well, so that the sessions of the same user across all clusters is handled.

Sample Request:

*https://192.168.0.0:8443/amsvc/v1/idpclusters/SCC7c9nsp/*sessions?userid=admin

Use HTTP GET to retrieve all active sessions for the user ‘admin’.

Use HTTP DELETE method to terminate all sessions for the user ‘admin’.

Sample Response for GET request:

```
{
   "userDN" : "cn=admin, o=novell",
   "sessionDetails": { ["identityServer":"192.168.0.6", "sessionCount":"1"],
                       ["identityServer":"192.168.0.7", "sessionCount":"2"]
```

```
                    }
}
```

3.3.7 Purge Access Gateway Cache

You can use this API to purge the access gateway server cache. Periodic purging of the cache frees up storage. You can select to purge the content of the purge list that has already been configured on the Administration Console or purge all content cached on the server.

Steps:

1. Get the list of Access Gateway clusters

   Sample URL: https://164.99.86.7:8443/amsvc/v1/agclusters

2. Parse the response to get the URL of the cluster you want to purge

   Sample Response:

   <agCluster uri="https://164.99.86.7:8443/amsvc/v1/agclusters/ce035b033e6c7f29">

3. Invoke the URL to get the devices in that cluster

   URL format: <cluster uri from above>/devices

   Sample URL: https://164.99.86.7:8443/amsvc/v1/agclusters/ce035b033e6c7f29**/devices**

4. Parse the response to get the URL of the device you want to purge

   Sample Response:

   <agDevice uri="https://164.99.86.7:8443/amsvc/v1/agclusters/ce035b033e6c7f29/devices/ag-6459CF981F6FD178">

5. Send a PUT request to the device URL with parameters to purge cache. See sample below.

Sample Request:

PUT request to https://164.99.86.7:8443/amsvc/v1/agclusters/ce035b033e6c7f29/devices/ag-6459CF981F6FD178

With input { "purge" : "list" }

Specify “list” to purge the content configured in the Purge List on the UI.

Use “all” to purge the entire cache.

Response:

200 OK

Note:

Clearing the cache decreases the responsiveness of a device, as every page will need to be retrieved. Therefore it is recommended to execute this command for one device at a time.

4 OAuth and OpenID Connect API

This section describes OAuth 2.0 and OpenID Connect implementation for authentication and authorization with NetIQ Access Manager. An application developer or administrator can get an access token from Access Manager and use it in their applications. By default, OpenID Connect is supported in all APIs.

4.1 Step by Step Guide

The following are the steps involved in enabling OAuth2.0 and OpenID Connect for your application:

Register your application in Access Manager.

Get a unique Client Credentials (Client ID and Client Secret) for your application.

Implement and invoke appropriate authentication or authorization requests with necessary parameters by using one of OAuth2.0 Authorization Grant flows.

Validate and trust tokens issued by Access Manager.

Invoke REST APIs by using the tokens issued.

Trust the tokens issued by Access Manager in REST APIs by resource servers protecting resources.

4.2 Registering Client Applications

Registering a client application includes the following activities:
1. Get Client ID and Secret
2. Register redirect URI
3.Register Authorization Grants
4. Register OpenID Connect configuration

Getting Client ID and Secret

To get an "Access Token" or "ID Token", the application needs to send Client Credentials. Client Credentials are unique credentials assigned per client application. The developer have to register their application with necessary details into the Access Manager to use any of the following API. The details of how to register their applications are specified in the NetIQ Access Manager Administration Guide. After registering the application, Access Manager will provide "client id" and "client secret". Note these values.

Registering Redirect URI

A valid redirection URI must be registered with Access Manager along with each client application. Access Manager will redirect only to these registered URIs for issuing tokens in the Authorization Code Grant flow and Implicit Grant flow. One of the registered URIs should be passed along with requests in these flows.

Registering Authorization Grants to be used

The client application has to specify which OAuth2.0 Authorization Grant flows the application will use. Access Manager will issue tokens only in the specified flows. Any requests with flows those are not registered during client registration is not supported. You can also modify this information after client is registered.

An administrator of your organization can also disable some of the Oauth 2.0 authorization grant flows to minimize the security risk. For example, an administration can disable the use of "Resource Owner Credential" grant if none of the Oauth 2.0 applications in the organization uses this flow. It is not recommended unless it is absolutely required.

4.2.1 Registering OpenID Connect Configurations

Access Manager supports both Oauth 2.0 and OpenID Connect specifications by default. Typically, OAuth2.0 is used for authorization of applications and OpenID Connect is used for authentication. Oauth 2.0 flow issues a security token called "Access Token" and OpenID Connect issues "ID Token" and optionally "Access Token".

ID Tokens are JSON Web Tokens (JWT) signed by Identity Server and optionally encrypted by client application's public certificate. The relying party can verify the signature of the ID Token and trust that token is issued by trusted Identity Server.

You can register signing algorithm to be used for a JWT token. If your application needs confidentiality of ID Token, provide a publicly accessible URL of public certificate and algorithm in the JWKS format. You need to configure this during client application registration.

4.2.2 OAuth 2.0 Endpoints

To get an Access Token and Identity Token, the client invokes requests to corresponding endpoints exposed by the Identity Server. The Identity Server exposes the following four endpoints:

Authorization Endpoint

Token Endpoint

TokenInfo Endpoint

UserInfo Endpoint

Authorization Endpoint is always contacted via a browser. This endpoint requires that user has

existing browser session with the Identity Server. If no session exists at the time of request, the Authorization Endpoint redirects the user to login. This endpoint is used when the client uses the Authorization Code flow or Implicit flow.

Token Endpoint is used directly by the client without involving the browser. Hence, it is possible to get an Access Token offline when the user is not connected via a browser. This endpoint can issue an Access Token when the client provides either a valid authorization code, resource owner credentials, or client credentials.

TokenInfo Endpoint is used for validating Access Tokens issued in OAuth 2.0 Authorization flows. Clients can send the Access Token via Authorization Header. This endpoint returns a JSON response stating whether the token is valid.

UserInfo Endpoint is used for getting Resource Owner's claims. A client can send a request to UserInfo endpoint with a valid Access Token and get the claims that are authorized by Resource Owner to share. This endpoint checks whether provided Access Token has valid scopes to issue the claims.

4.2.3 Other Endpoints

In addition to the above basic endpoints, the Identity Server exposes the following

endpoints: Metadata Endpoint

Client Registration Endpoint

Scope and Resource Server registration Endpoint

Metadata Endpoint exposes the basic services and options available at the Identity Server for OAuth 2.0 and OpenID Connect. This also contains URLs for basic endpoints. This endpoint is typically in this format: https://www.idp.com:8443/nidp/oauth/nam/.well-known/OpenID-configuration. Invoking this URL responds with a JSON document containing the following information:

OAuth2.0 Endpoints

ID Token supported algorithms

JWKS Keys which can be used for verifying ID token

Supported Response Modes, Response Types, and scopes

Client Registration Endpoint is used by developers to register the OAuth2.0 clients through REST API. This endpoint itself is protected by OAuth2.0 and hence the clients invoking this endpoint to register clients should obtain Access Tokens from the Authorization Endpoint by providing the developer's username and password. For registering new clients, the developer must have a "NAM_OAUTH2_DEVELOPER" role defined in the Identity Server.

The Scope and Resource Server Registration Endpoint is used to register , modify, or delete a scope. The users who invokes this endpoint must have "NAM_OAUTH2_ADMIN" role defined in Identity Server to be able to register manipulate the scope values.

4.3 Authentication

The application can request an authentication service from Access Manager by using one of the supported OAuth2.0 Authorization Grant flows. Access Manager implements OpenID Connect
1.0 specification on top of these flows, and hence the application can get more information about authentication and it can verify the issued identity tokens.

The result of authentication exchange is an identity token called as "ID Token". This token is in JSON Web Token (JWT) format. This token is signed by public signing certificate of Identity Server. The client application needs to verify the signature of the token and token is issued by the trusted Identity Server.

The authentication service assures the application that user has an active session at Identity Server with requested or default assurance level. The flows requiring active user session involves a browser redirect and hence the authorization grant flows in this section talks about "Authorization Code Grant" flow and "Implicit Grant" flow. The authentication service can use advanced authentication methods configured in the Identity Server. This does not require the application to know the password of the user. If your application does not depend on a browser interface or handles username/password directly, please refer to "Resource Owner Credential Grant" in section "Other OAuth2.0 Grants" below.

4.3.1 Getting Identity Tokens

You can use any one of the following flows to get an Identity Token:

- Authorization Code Grant Flow
- Implicit Flow

Authorization Code Grant Flow

The Authorization Code Grant is a two step process. In the first step, get a short lived "authorization code" from the Identity Server. In the second step, exchange this "authorization code" with "ID Token". The application can also request for "Access Token" for authorization if the application will make RESTful API calls to resource servers.

The application can also specify minimum assurance level for authentication method from Identity Server by using the below mentioned request parameter. Identity Server ensures that the user is authenticated with the requested level of authentication before sending the Identity Token.

Identity Token is a signed JSON Web Token (JWT). Signing is optional, but recommended. The token will be signed when the client is configured with "ID Token Signing Algorithm" during the client registration.

The application verifies the returned Identity Token as mentioned in the "Validating ID Token" section.

Getting an Authorization Code

The client can get an authorization code in first step by redirecting the browser to the Authorization Endpoint with the required query string values. See the “Request Parameters” section.

The response is a short life-time code called "authorization code". This code can be used to exchange identity tokens from Identity Server from the Token Endpoint by passing necessary request parameter explained in the subsequent section "Exchanging Authorization Code for Token". The authorization code is sent back through a browser redirect to a registered redirect_uri. The client should handle this request to the redirect_uri. This involves exchanging the code with an Access Token.

Request Parameters

To get an authorization code, the client should invoke a request to Identity Server's Authorization
Endpoint with the following request query string parameters:

Parameter	Required	Value	Description
client_id	Yes		Client application ID, which is obtained at the time of client application registration
response_type	Yes	code	it should be "code".
redirect_uri	Optional		If provided, the value of this must exactly match one of the registered URIs during application registration. If not provided, the browser will be redirected to any of the registered redirect URIs registered during application registration.
scope	Yes	OpenID	List of scopes the application requires. It should contain OpenID". You can get all “scopes_supported” at the authorization server’s OpenID Metadata Endpoint. Scope values should be space separated %20 or +.
state	Recommended		An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client. The parameter should be used to prevent cross- site forgery requests.
prompt	Optional	none/login/ consent	The values can be either "none", "login", or "consent". none: No user interface will be shown to user if user is not already authenticated. If not authenticated, an error message in one of "login_required", "interaction_required" or other will be sent back to client application. This is useful if client want to detect whether the user has an existing session with Identity Server or not and has necessary consents.

max_age	Optional	300	Maximum authentication age at Identity Server in seconds. If the user has not logged in within this elapsed time, the user will be re-prompted for authentication
acr_values	Optional	/name/password/uri	If client request contains acr_values parameter, Identity Server maps the value to configured contracts in Identity Server and prompts the user with the contract if the user is not already authenticated with the contract. The contract is not sent in ID token in this release.

Response Values

The Identity Server responds a HTTP 302 redirect message to the requested redirect_uri in the Authorization request . If the request does not contain the redirect_uri param, the Identity Server will redirect to one of the registered redirect_uri.

The redirect response will have the following parameters:

Parameter	Description
code	An opaque binary token. variable length field. Application should not assume the size of the code and allocate sufficient space for reading the code.
state	Contains the state parameter sent in the authentication request above

Sample Request and Response

A sample request with whitespace for readability:

```
HTTP/1.1 GET /nidp/oauth/nam/authz?
      &response_type=token
       &client_id=bb775b12-bbd4-423b-83d9-647aeb98608d
       &client_secret=bBbE-4mNO_kWWAnEeOL1CLTyuPhNLhHkTThA-
rEckyrdLmRLn3GhnxjsKI2mEijCSlPjftxHod_05dp-uGs6wA
       &redirect_uri=https://www.oauthapp.com/oauth.php
       &scope=email
       &nonce=ab8932b6
       &state=AB32623HS
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/41.0.2228.0 Safari/537.36
Host: www.idp.com:8443
Accept: */*
Cookie: JSESSIONID=188BAEB80B063C852D2CC82FDBD15A43
```

Response

```
HTTP/1.1 302 Found
Cache-Control: no-cache, no-store, no-transform
Location: https://www.oauthapp.com/oauth.php?
             code=/wEBAAY.........Ws1gQ~~
             &scope=email
Content-Length: 0
Date: Tue, 03 Mar 2015 18:12:55 GMT
```

Exchanging the Authorization Code with Identity Tokens

The client, listening for the authorization code in the registered redirect_uri, can use the requests explained in next section to exchange the authorization code with an Access Token. The request should be sent to the Token Endpoint.

The response is sent back in the JSON format and contains both access_token and id_token. Access Tokens are used for authorization and typically sent to an API server. The client when invoking any other API service has to include this access token. The API server will validate this access token and authorize the incoming API requests based on the scopes embedded in the access token.

The ID token contains the authentication information such as the issuer of the token, its validity, and when the user got authenticated. The client has to verify the signature of the ID token if present and check the issuer.

Request Parameters
The token request should have the following parameters:

Parameter	**Required**	**Description**
grant_type	Yes	authorization_ code
client_id	Yes	Client ID of the registered client
client_secret		Client Secret of the registered client
code	Yes	Code received in the Authorization code flow
redirect_uri	Yes	This should be same as the one sent during the authorization code request

Response Values

A successful request contains a JSON object with the following values:

Parameter	Required	Description
token_type	Yes	The type of the token. Authorization server currently supports only Bearer type
access_token	Yes	Access token that can be used to invoke resource server APIs
id_token	Optional if scope contains "OpenID"	When invoking authorization code request, if the client has sent OpenID, this response object will contain an ID Token.

scope	Optional	The list of scopes that user has authorized. This can contain all the scopes the client requested or lesser
state	Optional	if the "state" parameter was present in the client authorization request, the same state value sends in response

Sample Request and Response

```
 POST /nidp/oauth/nam/token HTTP/1.1
 User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/41.0.2228.0 Safari/537.36
 Host: www.idp.com:8443
 Accept: */*
 Content-Length: 695
 Content-Type: application/x-www-form-urlencoded

 grant_type=authorization_code
   &client_id=b017c96c-b16a-4d80-a5fa-68f5050abc58
   &client_secret=ZDDwbuuWPdV_e5quAf7f0Jkg_iJJ7g
   &redirect_uri=https://www.client.com/oauth_callback.php
   &code=/wEBAAk.................................

 HTTP/1.1 200 OK
 Server: Apache-Coyote/1.1
 Content-Type: application/json
 Content-Length: 916
 Date: Thu, 19 Mar 2015 14:14:57 GMT
 Connection: close

 {
   "access_token":"/wEBAAQEACA......",
   "token_type":"bearer", "expires_in":179,
   "refresh_token":"/wEBAAQEACA9lN8bgv..........",
   "scope":"email"
 }
```

Implicit Grant

Get an Access Token and ID Token by sending an HTTPS GET or POST request with the appropriate URI parameters to Authorization endpoint base URI The id token issued only if the scope is having OpenID param value. Idtoken can be signed and encrypted based on the client's registration request parameters. To access any of the user's attribute or getting permissions to access user's resource pages, you can include the request in the scope parameter. For example, if

you want user's email address and profile, set the scope parameter accordingly. If any of this scope requires approval from the user, the consent screen includes a request for provide the user's email address and profile to your application.

Sample implicit request/response:

https://example.netiq.com/nidp/oauth/nam/aut
hz?
response_type=token+id_token&client_id=4e4ae330-1215-4fc8-
9aa7-
79df8325451c&redirect_uri=https://client.example.com/callback&scope=email+OpenID&state
=
s1234&nonce=n123

In this case, the authorization server validates all request parameters and checks whether the user is authenticated. If the user is not authenticated, then it sends the login page to user.After the user is authenticated, it takes the consent from the user for the requested scopes and sends the response to the client.

The authorization server sends the following response for the above request:

https://client.example.com/callback#token_type=bearer&access_token=/wEBAAUFACAjDfPtnd/zlOWPpN/kV1Jtt3nxCPtzHyUH~&expires_in=3600&id_token=eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJo&scope=email&state=s1234

Token Request URI Parameters

The token request should have the following query parameters and the request should be sent to the Authorization Endpoint:

Parameter	Required	Description
response_type	Yes	The possible values are token, id_token and token id_token. If the value is passed as token then authorization server issues only access token to client. If the value is id_token then authorization server issues only id_token in the response. If the value if token id_token then authorization server issues both access token and Id token in the response. The id_token issued only if the scope is having OpenID value
client_id	Yes	Client application ID which is obtained at the time of client application registration.
redirect_uri	Optional	If provided, the value of this must exactly match one of URIs of the registered application.
scope	Optional	Scopes supported by the Authorization server, you should get “scopes_supported” at authorization server’s OpenID Metadata Endpoint. For ID Token, OpenID should be available in the scope. Multiple scope values should be space separated
state	Yes	An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when the response sends to the client. The parameter should be used to prevent cross-site forgery
nonce	Yes	String value used to associate a Client session with an ID Token, and to mitigate replay attacks

Parameter	Required	Description
prompt	Optional	Space delimited, case sensitive list of ASCII string values that specifies whether the Authorization Server prompts the End-User for re-authentication and consent. The defined values are: none The Authorization Server MUST NOT display any authentication or consent user interface pages. An error is returned if an End-User is not already authenticated or the Client does not have pre-configured consent for the requested Claims or does not fulfill other conditions for processing the request. login The Authorization Server SHOULD prompt the End-User for reauthentication. If it cannot reauthenticate the End-User, it MUST return an error to client consent The Authorization Server SHOULD prompt the End-User for consent before returning information to the Client. If it cannot obtain consent, it MUST return an error, typically consent_required.
max_age	Optional	Maximum authentication age. Specifies the allowable elapsed time in seconds since the last time the user was authenticated by the OP.

Response Values

The Authorization Endpoint sends an HTTP 302 Redirect response with the following http Fragment values in the "Location" header. The browser redirects the request to the returned Location header without exposing the Fragment values. The Location header will be the "redirect_uri" parameter passed in the request. The fragment values can only be read by the browser and never sent to the redirect_uri.

Parameter	Required	Description
token_type	Yes	The type of the token. Authorization server supports only bearer
Access_token	Based on response type value	If the response type value is either token or token id_token then authorization server sends access token
id_token	Based on response type value	If the response type value is either id_token or token id_token then authorization server sends id_token

scope	Optional	The user given consent scope names.
state	Optional	if the "state" parameter was present in the client authorization request the same state value sends in response

4.4 Authorization

OAuth 2.0 is an authorization protocol. The resources hosted in the resource server can protect the access to resources by verifying the Access Token available in the API request. A client application offering a service to the user (Resource Owner), that needs to act on the resource owned by the user has to get an Access Token from Identity Server. The resource server verifies that this token is issued by a trusted issuer and containsnecessary scopes to access the resource.

The client can get the Access token from Identity Server by invoking one of the supported OAuth 2.0 authorization flows by using the client's credentials. The client usually invokes one of the following flows mentioned here or the grants explained in "Other Grants".

Getting an Access Token

You can get an Access Token by using any one of the following flows:

- Authorization Code Grant
- Implicit Grant
- Refresh Token

Authorization Code Grant

This is same as getting the Identity Token explained in section "Authentication

> Getting Identity Token > Authorization Code Grant".

Implicit Grant

This is same as getting the identity token explained in section "Authentication > Getting Identity Token > Implicit Grant". To get an access token, the request must contain "response_type value as token".

Refresh Token

The Authorization Code Grant and Implicit Grant flows require that the user is available on the browser and has an active session with the Identity Server. Hence, these flows are called as online flows.

Sometime, the client might need access to resources even if the user is not available online. For example, when a client wants to do batch processing on resources owned by a user, it might need to have a longer life time of access token. Access Tokens usually have shorter life time. The refresh tokens have longer life time. Using the refresh tokens, clients can ask for fresh access tokens. Since, the access tokens are issued offline when the user is not active, this flow is called as an offline flow.

A client can use this option if the access token is expired or going to expire.

Refresh Token Request URI Parameters

The refresh token request should be sent to the Token Endpoint. The request should

have following parameters in query string of the request:

Parameter	Required	Description
grant_type	yes	Must be refresh_token
client_id	yes	Client application ID that is obtained at the time of client application registration.
client_secret	yes	Client secret that is obtained at the time of client application registration.
refresh_token	yes	refresh_token that is obtained during authorization grant, resource owner credentials, client credentials flow

Parameter	Required	Description
scope	optional	List of the scope names separated by space.

Refresh Token Response Values

A successful request to token endpoint with refresh token will result in following response. The response will be in JSON object with the following values:

Parameter	Present	Description
access_token	yes	Access token
refresh_token	yes	Reissue a refresh_token
token_type	yes	Token type supported by authorization server is bearer
expires_in	yes	The validity time of the access
token scope	optional	Granted scopes to the client

Sample Request

A sample request and response, with line breaks for better readability.

```
HTTP/1.1 POST /nidp/oauth/nam/token
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/41.0.2228.0 Safari/537.36
Host: www.idp.com:8443

'grant_type=refresh_token
   &client_id=4e4ae330-1215-4fc8-9aa7-79df8325451c
   &client_secret=Rxl5pvgL80DBzbIcLPVnH17FehZA8LLT-
7oZ9POFrEguEyB2JMzB6kBj3JH4BxpZTrnFSjmFgrCClQuCKt3MUg
   &refresh_token=/wEBAAcHACAup9Kv@JZbLuBBaWeaYfkP/NT'
```

A successful response

```
HTTP/1.1 200 OK
Cache-Control: no-cache, no-store, no-transform
Content-Length: 0
Date: Tue, 03 Mar 2015 18:12:55 GMT

{

"access_token":"/wEBAAYGACBgyZapAgMYk7oJYXFO9/LIblf9FAnqp@Y1/Y/voByU9Z2awkC

bfp
LZTzpUqFspZ4xrJc/TcNAl3hktfRDJgOUEHUkdyO/FoWxmTn3NrHL0K8kNPQo7nm3kyUSyjpxxv
jVw
SOPtVmNl94AXOIxqObYpLoRgpqqeO8TUltvQlk9zMNkAmHscPTYFwMrzHE@B98kIrZ1b266eSbu
AmL
r4y1guAx0yYs1XhboFd97I6mabGXDqeAjjpx/DTZBTCptA/LlIJgN10jMwik7x9nZZ3wjv16/4h
w8G
UHaS09uHXqqtF3S0pJ6/aM/hsWAgkcZeOhliPGXV8T7tjMmc8V1t4mIzuOagzN0LbaclD1OBknd
IKC OcqJiiMMRDZNEHBjwoOXc~",
"token_type":"bearer",
"expires_in":3599,
"scope":"profile
email"
}
```

4.5 Validating Tokens

Access Manager issues tokens of variable length. The application should not assume the size of the tokens.

The access token received in earlier flows can be validated by sending a request to the TokenInfo endpoint of Identity Server.

The URL to invoke is ht://idpbaseurl.com/nidp/oauth/nam/tokeninfo.

The request should contain the token in the Authorization header as follows:

```
Authorization: Bearer access_token
```

Response Values

The response to the TokenInfo endpoint will contain the following values in JSON format:

Parameter	Present	Description
expires_in	Yes	number of seconds the token is valid from now
user_id	Yes	user to whom the token was issued to
scope	Yes	list of scope values the token holds

Sample Request and Response

Request:

```
Host: www.idp.com:8443
Accept: */*
Authorization: Bearer /wEBAAMDACAYtKt.............@kBEzw~~
```

Response:

```
 HTTP/1.1 200 OK
 Server: Apache-Coyote/1.1
 Content-Type: application/json
 Content-Length: 48
 Date: Thu, 19 Mar 2015 15:47:25 GMT

{
  "expires_in":
  145, "user_id":
  "alice",
  "scope": []
}
```

4.6 Other OAuth 2.0 Grants

4.6.1 Resource Owner Credential Grant

The resource owner credential grant flow requires a client to know the user credentials. To exchange the username and password for an access token, send an HTTPS POST request with the appropriate URI parameters to token endpoint base URI. Note the http connections are not
accepted, use HTTPS. You should retrieve the token endpoint base URI at authorization server's
OpenID Metadata Endpoint.

Request Parameters

Parameter	Required	Description
client_id	Yes	Client application ID which is obtained at the time of client application registration.
client_secret	Yes	The client secret value.
grant_type	Yes	Use 'password' as value for this parameter.
username	Yes	The user login name
password	Yes	The user login password
scope	Yes	Scopes supported by the Authorization server, you should get "scopes_supported" at authorization server's OpenID Metadata Endpoint.For IDToken, OpenID should be present in the scope. Multiple scope values should be space separated %20 or +.
acr_values	Optional	If client request contains acr_values parameter, Identity Server maps the value to configured contracts in Identity Server and executes the contract. For example, use parameter value as /name/password/uri The contract is not sent in ID token in this release. Use space as delimiter to specify more than one contract URI for acr_values. In this case, the Identity Server execute contracts in the order as specified. Any one of the contract execution success is considered as authentication success. If none of the contract succeeds then authentication fails.

Response Parameters

Parameter	Description
access_token	OAuth2 access token.
token_type	The type of token returned. At this time, this is always having the value Bearer.
expires_in	The remaining lifetime of the access token.
scope	Scopes requested. The access token allows you to access to these scope(s).
refresh_token	The refresh token will be returned if client application is registered for it. This token can be used to refresh the access token once it expires.

Sample Request and Response

A sample request with whitespace for readability

```
HTTP/1.1 POST /nidp/oauth/nam/token?
       &grant_type=password
       &client_id=bb775b12-bbd4-423b-83d9-647aeb98608d
       &client_secret=bBbE-4mNO_kWWAnEeOL1CLTyuPhNLhHkTThA-
rEckyrdLmRLn3GhnxjsKI2mEijCSlPjftxHod_05dp-uGs6wA
       &username=user1
       &password=pass@123
       &scope=email%20profile
> User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/41.0.2228.0 Safari/537.36
> Host: www.idp.com:8443
> Accept: */*
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 630
{
       "access_token":  "/wEBAAEBACAgHkphv9NdD5khH7CLty7PpURg9RKOQ5pm6...",
       "token_type"   : "bearer",
       "expires_in"     :  3599,
       "scope"          : "profile email"
}
```

Note: If validation errors are occurred, HTTP Status 400 returned with the JSON response contains “error” and “error_description”.

Sample error response with whitespace for readability

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: 143
{
 "error"                 :  "invalid_request",
 "error_description" :   "OAuth Client Authentication Failure because password
parameter is missing in the request"
}
```

4.6.2 Client Credential Grant

The client credentials can be exchanged for access token. To get access token send an HTTPS POST request with the appropriate URI parameters to token endpoint base URI. Note the http connections are refused, use HTTPS. You should retrieve the token endpoint base URI at authorization server’s OpenID Metadata Endpoint.

Request Parameters

Parameter	Required	Description
client_id	Yes	Client application ID which is obtained at the time of client application registration.
client_secret	Yes	The client secret value.
grant_type	Yes	Use ‘client_credentials’ as value for this parameter.

Response Parameters Parameter

Description	access_token	OAuth2
access token.		
token_type	The type of token returned. At this time, this is always having the value Bearer.	
expires_in	The remaining lifetime of the access token.	

Sample Request and Response

A sample request with whitespace for readability

```
HTTP/1.1 POST /nidp/oauth/nam/token?
       &grant_type=client_credentials
       &client_id=bb775b12-bbd4-423b-83d9-647aeb98608d
       &client_secret=bBbE-4mNO_kWWAnEeOL1CLTyuPhNLhHkTThA-
rEckyrdLmRLn3GhnxjsKI2mEijCSlPjftxHod_05dp-uGs6wA
       &redirect_uri=https://www.oauthapp.com/oauth.php
       &scope=email%20profile
> User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/41.0.2228.0 Safari/537.36
> Host: www.idp.com:8443
> Accept: */*
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 630
{
    "access_token": "/wEBAAAAACBy4Ku4ApcxEV7er19P6nqH5HZg5J6GcY...",
    "token_type"   : "bearer",     "expires_in"    :  3599
}
```

Note: If validation errors are occurred, HTTP Status 400 returned with the JSON response contains “error” and “error_description”.

Attribute Service

Identity Server exposes an endpoint to which the clients and resource servers can query for user's claims that is associated with an access token. This service is implemented in UserInfo Endpoint.

The clients or resource servers can invoke the request to UserInfo Endpoint by including the access token in the authorization header as given below:

Authorization: Bearer access_token

The UserInfo endpoint returns the claims associated with the access token in a JSON object as given in the response values.

Response Values

ParameterDescription	
sub	Unique ID identifying the subject. This will be GUID of the user.

The other claims are included as values in JSON object if the access token contains the necessary scope and user has authorized the client to access the claim.

For example, if the client has requested "email" scope, the UserInfo endpoint will return a value "email" : "alice@c.com" along with the "sub" field.

Sample Request and Response

Request:

```
 GET /nidp/oauth/nam/userinfo HTTP/1.1
 User-Agent: curl/7.41.0
 Host: www.idp.com:8443
 Accept: */*
 Authorization: Bearer /wEBAA.............DSDG
```

Response:

```
 HTTP/1.1 200 OK
 Server: Apache-Coyote/1.1
 Content-Type: application/json
 Content-Length: 73
 Date: Thu, 19 Mar 2015 16:14:52 GMT

{
  "sub": "6adb7ca411d5a14c94946adb7ca411d5",
  "email": "alice@a.com"
}
```

5 Component Statistics API

The Identity Servers and Access Gateway systems provide APIs to retrieve the statistics of that system. These are a precursor to the Administration API that also provides statistics. Both APIs provide the same data, but the difference is in the invocation point. The Component statistics API must be called for every device IP, while the Administration API can be invoked against the Administration console to retrieve statistics of all the Identity Servers and Access Gateways in the system. Hence it is recommended to use the Administration APIs.

For more information about these APIs, see the Component Statistics Through REST APIs section in the Access Manager Administration Guide.