Access Manager 5.0
Administration API Guide

March 2021

Legal Notice

Contents

About this guide

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

IMPORTANT: The Technical Support team supports 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 technical support and need to go through the namsdk@microfocus.com channel.

---

1 API Overview

Access Manager APIs are broadly categorized as follows:

- **Administration APIs:** The administration APIs help to automate the common administrative tasks. Administration Console exposes these APIs. See Chapter 2, “Administration API,” on page 9.
- **OAuth and OpenID Connect APIs:** Identity Server exposes these APIs for all OAuth functionalities, such as endpoints for registering clients, and obtaining access tokens. See Chapter 3, “OAuth OpenID Connect API,” on page 19.
- **Component Statistics APIs:** These APIs provide statistics of Identity Servers and Access Gateways. The individual devices expose these APIs. 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. As a single administration API provides details for all devices. See Chapter 4, “Component Statistics API,” on page 55.
- **AWS Auto Scaling API:** These APIs help to automate tasks related to auto scaling Access Manager on AWS. See Chapter 5, “AWS Auto Scaling API,” on page 57.

2 Administration API

2.1 Accessing Administration APIs

Administration Console supports Administration APIs, OAuth and OpenID Connect APIs, and Component Statistics APIs. You can invoke these APIs by using a browser or by using the curl command in scripts to help automate the administrative tasks.

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

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

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

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

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.

2.2 Detailed API Documentation

You can access the detailed documentation for all administration APIs in the REST API Doc on the Access Manager Developer Resources website.

2.3 Administration API Use Cases

2.3.1 Get Device Health

This API returns the health of Access Manager devices (Identity Servers and Access Gateways). The API returns the health for the following levels:

- Entire Access Manager
- Each cluster
- Each device
- Each service and component (remote web servers, data stores, and so forth)

You can use this API for integration with external systems, such as NOC to view the status of Access Manager devices and the remote web servers.

Sample Request

Invoke a URL similar to `https://192.168.0.0:8443/amsvc/v1/health?expand=4`

The `expand` parameter specifies the level of detail that is returned. Accepted values are 1,2,3, and 4. 4 returns the maximum detail for all 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/SCC7c9nsp/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 Administration Console. This data is refreshed every five minutes. Therefore, it is sufficient to invoke this API every 5 minutes to get the latest health.

2.3.2 Get Device Statistics

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

Sample Request

Send a GET request to a URL. For example, `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 Administration Console. It is refreshed every 10 minutes. Therefore, it is sufficient to invoke this API every 10 minutes to get the latest statistics.

2.3.3 Refresh Metadata of SAML 2.0 Trusted Providers

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

Perform the following 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 follows.

5 Invoke the trusted roots API to add the root CA of the signing certificate specified in the metadata. This step is needed if the certificate has changed. For more information, see Section 2.3.4, “Import Trusted Root Certificates,” on page 13.

6 Invoke the Apply changes API to send these changes to Identity Servers in that cluster.

7 Send PUT request to the cluster URL https://192.168.0.0:8443/amsvc/v1/idpclusters/SCC7c9nsp/ with input

```
{ "update" : "all" }
```

Sample Script: You can access a sample script that implements all the steps listed here on the Update MetaData From File (https://community.microfocus.com/t5/Access-Manager-Tips-Information/Update-MetaData-From-File/ta-p/1776007) page.

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: The 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"
}
```

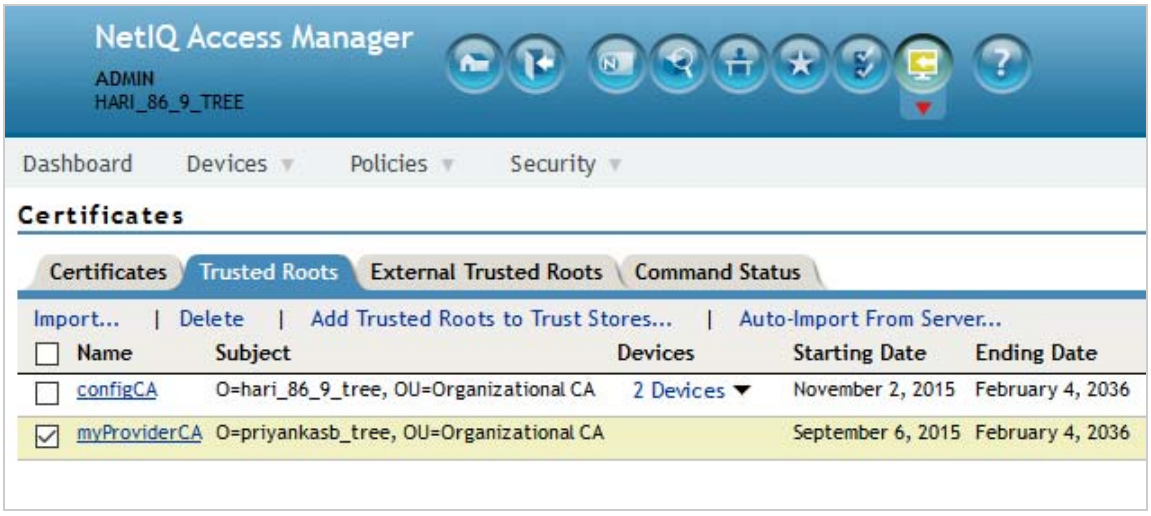
Response: 200 OK

2.3.4 Import Trusted Root Certificates

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



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

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

Response: 200 OK

IMPORTANT:

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

2.3.5 Renew Certificates

You can use this API to renew the certificates that are available through 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----"
}
```

IMPORTANT: 1.An update is required for all devices using that certificate. Updating the connector certificate requires tomcat restart.

2. The 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.

2.3.6 Manage User Sessions

These APIs allow fetching and terminating all active sessions of a given user.

Perform the following steps:

1 Invoke the API to get all 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 so that the sessions of the same user across all clusters are 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

```
{
   "userDN" : "cn=admin, o=novell",

   "sessionDetails": { ["identityServer":"192.168.0.6",
"sessionCount":"1"],

["identityServer":"192.168.0.7", "sessionCount":"2"]

}
}
```

2.3.7 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 Administration Console or purge all content cached on the server.

Perform the following 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.

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.

2.3.8 Scaling the Devices

You can use these APIs to scale up or scale down Access Gateway and Identity Servers. These APIs can only assign or delete a node in an existing cluster. For more information about how to configure these APIs, see the following configurations.

2.3.8.1 Scaling Up Access Gateway

Perform the following steps:

1 Assign a node to an existing Access Gateway cluster. Send a POST request to the following URL with the cluster ID and the device ID.

NOTE: No input required for this request.

POST: `https://<AC_IP:PORT>/amsvc/v1/agclusters/<clusterID>/devices/<deviceID>`

In the above POST request, the following details are used.

Cluster ID: The name of the existing Access Gateway cluster to which the node will add.

Device ID: The device ID of the new Access Gateway node that is to be assigned.

Sample Response

```
200 OK
```

2 When the node is assigned to the cluster, send a PUT request to the following URL to update the cluster

`https:// <AC_IP:PORT>164.99.86.7:8443/amsvc/v1/agclusters/<clusterID>`

With input { "update" : "all" }

2.3.8.2 Scaling Up Identity Server

Perform the following steps:

1 Assign a node to an existing Identity Server cluster. Send a POST request to the following URL with the cluster ID and the device ID.

NOTE: No input required for this request.

POST: `https://<AC_IP:PORT>/amsvc/v1/idpclusters/<clusterID>/devices/<deviceID>`

In the above POST request, the following details are used.

Cluster ID: The name of the existing Identity Server cluster to which the node will add.

Device ID: The device ID of the new Identity Server node that is to be assigned.

Sample Response

```
200 OK
```

2 When the node is assigned to the cluster, send a PUT request to the following URL to update the cluster.

```
https:// <AC_IP:PORT>164.99.86.7:8443/amsvc/v1/idpclusters/<clusterID>
```

With input { "update" : "all" }

2.3.8.3 Scaling Down Access Gateway

Scaling down from a cluster: Perform the following steps:

1 Delete a node from an existing Access Gateway cluster. Send a DELETE request to the following URL with the cluster ID and the device ID.

NOTE: You can not delete the primary Access Gateway nodes. You can delete only the secondary nodes of Access Gateway in a cluster.

POST: `https://<AC_IP:PORT>/amsvc/v1/agclusters/<clusterID>/devices/<deviceID>`

In the above POST request, the following details are used.

Cluster ID: The name of the Access Gateway cluster from which the node will be deleted.

Device ID: The device ID of the Access Gateway node that is to be deleted.

Sample Response

```
200 OK
```

2 When the node is deleted from the cluster, send a PUT request to the following URL to update the cluster.

```
https:// <AC_IP:PORT>164.99.86.7:8443/amsvc/v1/agclusters/<clusterID>
```

With input { "update" : "all" }

Scaling down an individual node: Perform the following to delete a node that is not part of a cluster:

Delete a node from an Administration Console. Send a DELETE request to the following URL with the device ID.

POST: `https://<AC_IP:PORT>/amsvc/v1/agclusters/<clusterID>/devices/<deviceID>`

Device ID: The device ID of the Access Gateway node that is to be deleted.

Sample Response

```
200 OK
```

2.3.8.4 Scaling Down Identity Server

Scaling down from a cluster: Perform the following steps:

1 Delete a node from an existing Identity Server cluster. Send a DELETE request to the following URL with the cluster ID and the device ID.

NOTE: You can not delete the primary Identity Server nodes. You can delete only the secondary nodes of Identity Server in a cluster.

POST: `https://<AC_IP:PORT>/amsvc/v1/idpclusters/<clusterID>/devices/<deviceID>`

In the above POST request, the following details are used.

Cluster ID: The name of the Identity Server cluster from which the node will be deleted.

Device ID: The device ID of the Identity Server node that is to be deleted.

Sample Response

```
200 OK
```

2 When the node is deleted from the cluster, send a PUT request to the following URL to update the cluster.

`https:// <AC_IP:PORT>164.99.86.7:8443/amsvc/v1/idpclusters/<clusterID>`

With input { "update" : "all" }

Scaling down an individual node: Perform the following to delete a node that is not part of a cluster:

Delete a node from an Administration Console. Send a DELETE request to the following URL with the device ID.

POST: `https://<AC_IP:PORT>/amsvc/v1/idpclusters/<clusterID>/devices/<deviceID>`

Device ID: The device ID of the Identity Server node that is to be deleted.

Sample Response

```
200 OK
```

3 OAuth 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 and refresh token from Access Manager, and use it in their applications. By default, all APIs support OpenID Connect.

3.1 Example Scenarios

This section includes information about using API for OAuth and OpenID Connect in the following scenarios:

3.1.1 A Client Application Requires to Access OAuth Protected Resources

A client application can use the Access Manager token to access an Access Manager OAuth protected resource. The following is the workflow of accessing a protected resource when the client uses the Access Manager token:

1. **Registration:** The client must be registered in Access Manager. For information about registering a client, see Registering Client Applications.
2. **Retrieve a token from Access Manager:** The client retrieves the token by selecting any of the following authorization grant flows:
   - Authorization code flow
   - Implicit flow
   - Resource owner credentials flow
   - Client credential flow

- ID token flow
- SAML 2 bearer profile for authorization grant flow

NOTE: For information about grants, see OAuth Authorization Grant in the Access Manager 5.0 Administration Guide. For required endpoints, see the respective endpoint sections in OAuth 2.0 Endpoints.

3. **Send the token to resource server:** The token is sent as an authorization header bearer token.

3.1.2 Resource Server Validating a Token Issued by Access Manager

A resource server can validate a token that is issued by Access Manager through resource server keys or through Access Manager keys. By default, the token encryption is done by using Access Manager keys. The resource server sends a request to Access Manager to validate the token. If you provide the resource server's key and encryption algorithm details in Access Manager, the resource server does not require to send a request to Access Manager. Instead, the resource server can use its key to validate the token.

Only an Access Manager administrator can register a new resource server. To validate a token, the resource server must know how the token is encrypted.

Encrypted by Access Manager: This is an older way of validating the token. You need to send the token to Access Manager's token info endpoint for validation.

Encrypted using configured resource server keys: No need to validate through Access Manager. The resource server cryptokeys can be configured in Access Manager. Access Manager uses this key to encrypt the access token. This enables a resource server to validate the token itself, without sending it to the Access Manager token verification endpoint.

The following is a sample in Java code about how to validate the token:

```
//Step1: decrypt the JWT Token (JWE Standard)
String jwtAccessToken =
"eyJhbGciOiJBMTI4S1ciLCJlbmMiOiJBMTI4R0NNIiwidHlwIjoiSldUIiwia2lkIjoibmFtLTEifQ.Zj
E0jRb5oh3suQZHFmaB-m....";

JsonWebEncryption jwe = new JsonWebEncryption();
jwe.setCompactSerialization(jwtAccessToken);
JsonWebKeySet jsonWebKeySet = new JsonWebKeySet(jwks);
List<JsonWebKey> jsonWebKeys = jsonWebKeySet.getJsonWebKeys();
JsonWebKey jsonWebkey = jsonWebKeys.stream().filter( p ->
p.getKeyId().equalsIgnoreCase(jwe.getKeyIdHeaderValue())).findFirst().orElse(jsonW
ebKeys.get(0));
if(jsonWebkey instanceof RsaJsonWebKey){
RsaJsonWebKey rsa = (RsaJsonWebKey) jsonWebkey;
jwe.setKey(rsa.getPrivateKey());
}else
{
jwe.setKey(jsonWebkey.getKey());
```

```
}
String decryptedToken = jwe.getPlaintextString();

//Step 2: Verify the JWT Signature (JWS Standard)
JsonWebKeySet jsonWebKeySet = new JsonWebKeySet(jwks);

JsonWebKey jsonWebkey = jsonWebKeySet.getJsonWebKeys().get(0);
JsonWebSignature jws = new JsonWebSignature();
jws.setKey(jsonWebkey.getKey());;
jws.setCompactSerialization(decryptedToken);
if(true == jws.verifySignature()){
System.out.println("Signature is valid.");
String payload = jws.getPayload(); //
}
```

For detailed sample code and tool for validating the JWT access token, see JWT Validation tool.

No encryption: Trust and accept the token. As access token is not encrypted, use the sample in Java code mentioned in the previous step to verify the signature and trust the token. For information about configuring access token encryption keys, see Registering a Resource Server.

3.1.3 Access Manager Revoking Refresh Tokens

Access Manager revokes only the refresh token and its corresponding access token. Only the refresh tokens that are generated by Access Manager Version 4.4 or later can be revoked.

You can perform the following tasks by using Access Manager API:

- Revoking refresh token for applications
- Revoking tokens that are issued to a device

For example, a user lost the device and wants to revoke all tokens that are issued to that device.

Using Mobile Access SDK: Use the Access Manager user portal for deregistering a device. When device is deregistered, the refresh token and associated access token are revoked.

Not using Mobile Access SDK: If you do not use Mobile Access SDK to revoke a device, you must provide the device ID in the access token request so that the device can be associated with the token. You can use this device ID later for revoking the tokens issued to the device. For more information, see Revoking Token Issued to a Device.

3.2 Prerequisites for Establishing an OAuth 2.0 Connection with a Client Application

3.2.1 Registering Client Applications

Registering a client application includes the following activities:

- Section 3.2.1.1, “Getting Client ID And Secret,” on page 22
- Section 3.2.1.2, “Registering Redirect URI,” on page 22
- Section 3.2.1.3, “Registering Authorization Grants,” on page 22
- Section 3.2.1.4, “Registering OpenID Connect Configuration,” on page 22

3.2.1.1 Getting Client ID And Secret

To get an *access token* or an *ID token*, the application needs to send Client Credentials. Client Credentials are unique credentials assigned per client application. Developers need to register their applications to Access Manager with necessary details to use any of the APIs. The details of how to register their applications are specified in Registering a Client Application. After registering the application, Access Manager provides *client id* and *client secret*. Note these values.

3.2.1.2 Registering Redirect URI

A valid redirection URI must be registered with Access Manager along with each client application. Access Manager redirects only to registered URIs for issuing tokens in authorization code grant flow and implicit grant flow. One of the registered URIs must be sent along with requests in these flows.

3.2.1.3 Registering Authorization Grants

A client application must specify the OAuth2.0 authorization grant flows the application will use. Access Manager issues tokens only in the specified flows. Any request with non-registered flows during client registration is not supported. You can modify this setting after a client is registered.

An administrator can disable few 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 required.

3.2.1.4 Registering OpenID Connect Configuration

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*.

Access tokens and ID tokens are JSON Web Tokens (JWT) signed by Identity Server and ID tokens are optionally encrypted by the 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 the ID token, provide a publicly accessible URL of public certificate and algorithm in the JWKS format. You need to configure this during the client application registration process.

3.2.2 Managing Client Applications

You can programmatically register, view, modify, and delete a client application in Access Manager.

Before performing any of these actions, you must define your role as `NAM_OAUTH2_DEVELOPER` or `NAM_OAUTH2_ADMIN` in the IDP Role policy.

You can register an application in any of the following two ways:

- Using username and password.
- Using Access token. To register a client application by using an access token, you must have your role defined as `NAM_OAUTH2_DEVELOPER` in the IDP Role policy.

Use the resource owner flow to get an access token. The endpoint of the resource owner flow is `https://<Identity Server URL: Port Number>/nidp/oauth/nam/token`.

This endpoint requires the followings parameters to provide an access token:

Parameter	Value
grant_type	Password
username	Application developer's user name
password	Application developer's password
scope	urn:netiq.com:nam:scope:oauth:registration:full (This scope allows you to register, view, modify, and delete client applications.) urn:netiq.com:nam:scope:oauth:registration:read (This scope provides read-only access)
token endpoint	Identity Server URL: Port Number>/ nidp/oauth/nam/token

- Section 3.2.2.1, “Registering a Client Application,” on page 23
- Section 3.2.2.2, “Modifying a Client Application,” on page 25
- Section 3.2.2.3, “Viewing a Client Application,” on page 26
- Section 3.2.2.4, “Deleting a Client Application,” on page 26

3.2.2.1 Registering a Client Application

To register a client application, the HTTP method value must be POST. Identity Server uses the following endpoint for registering a client application:

`https://<Identity Server URL: Port Number>/nidp/oauth/nam/clients`

The endpoint requires the following OAuth parameters for client registration:

Parameter	Required/ Optional	Description
client_name	Required	Name of the client application.
application_type	Optional	web or native.

Parameter	Required/ Optional	Description
enableNativeSSO	Optional	Specify true as a value to enable single sign on for a user who uses the client applications on a desktop or a mobile. For example, A user accesses client A using the credentials and is authenticated. Client A receives a refresh token and an access token. The user accesses client B immediately or after a few days. If this option is enabled for client B, the client uses the persistent cookie to retrieve the token and authenticate the user. Hence, client B is authenticated automatically. If this option is not enabled for the client B configuration, then to retrieve refresh token and access tokens, the user is required to provide credentials even though the user has already authenticated for client A.
redirect_uris	Required	Redirection URI values used by the client application.
grant_types	Optional	The following are supported grant types: ◆ authorization_code ◆ implicit ◆ refresh_token ◆ resource_owner_credentials ◆ client_credentials ◆ saml2_assertion If you do not specify a grant type, the default grant type `authorization_code` is used.
response_types	Optional	The following list includes supported response types: ◆ code ◆ code token ◆ code id_token token ◆ id_token ◆ id_token token ◆ access_token ◆ refresh_token
alwaysIssueNewRefreshToken	Optional	Specify true to issue a new refresh token for each refresh token request.
authzCodeTTL	Optional	Specify the duration in minute after that the authorization code becomes invalid.
accessTokenTTL	Optional	Specify the validity duration access token and ID token in minutes.
refreshTokenTTL	Optional	Specify the validity duration for the refresh token in minutes.
corsdomains	Optional	To allow access to requests from only selected domains, specify the domains as a JSON array. For example, `["beem://www.test.com", "fb://app.local.url", "https://namapp.com"]`

Parameter	Required/ Optional	Description
logo_uri	Optional	Specify the URL of the logo that to be included in the consent page. For example, `https://client.example.org/logo.png`
policy_uri	Optional	Specify the URL of the relying party client's privacy policy. For example, `https://client.example.org/privacypolicy`
tos_uri	Optional	Specify the URL of the relying party's terms of service. For example, `https://client.example.org/terms`
contacts	Optional	Specify the email addresses of people related to this client application.
jwks_uri	Optional	Specify the URI of the JSON file containing the json web keys. This key set contains signing keys that the relying party uses to validate signatures from the OpenID provider. For example, `https://client.example.org/my_public_keys.jwks`
id_token_signed_response_alg	Optional	Specify the ID Token Signed Response Algorithm. This algorithm is required for signing the ID token issued to a client.
id_token_encrypted_response_alg	Optional	Specify the algorithm used to encrypt the key.
id_token_encrypted_response_enc	Optional	Specify the algorithm used to encrypt the content.

3.2.2.2 Modifying a Client Application

Perform the following steps:

1. Retrieve client details from the `https://<Identity Server URL: Port Number>/nidp/oauth/nam/clients/<client ID>` endpoint. In the request for retrieving client details, use GET as the HTTP method value.
2. Send the update request. In the request, use POST as the HTTP method value. Identity Server uses the following endpoint for modifying a registered client application:

   `https://<Identity Server URL: Port Number>/nidp/oauth/nam/clients/`

   For the list of parameters this endpoint requires for a client application modification, see the table under Registering a Client Application.

NOTE: For updating a client application, you must send the XML with all parameters in the update request. If you do not include a parameter in the XML, the server does not initialize this parameter. For example, to update the `response_types` parameter, send the updated value for this parameter and existing values for other parameters in the request.

3.2.2.3 Viewing a Client Application

To view a client application, use GET as the HTTP method value.

- **https://<Identity Server URL: Port Number>/nidp/oauth/nam/clients/:** To view all clients applications registered by a developer
- **https://<Identity Server URL: Port Number>/nidp/oauth/nam/clients/<client ID>:** To view a specific client application registered by a developer

3.2.2.4 Deleting a Client Application

To delete a client application, the HTTP method value must be DELETE. Identity Server uses the following endpoint for deleting a registered client application:

```
https://<Identity Server URL: Port Number>/nidp/oauth/nam/clients/<client ID>
```

3.2.3 Registering a Resource Server

By default, access tokens are signed by JSON Web Tokens (JWT) and encrypted by Identity Server. Registering a resource server provides more features such as options to encrypt an access token by using the resource server encrypted keys or Identity Server encrypted keys. It also provides an option to not encrypt an access token. This is not recommended as it might cause security issues.

After resource server registration, specify the registered resource server name in the token request for encrypting the access token using the resource server encrypted keys. In this way, no need to contact Identity Server's TokenInfo/UserInfo endpoint for token validation or for claims. Only an Access Manager administrator can register a resource server.

You can register, view, modify, and delete a resource server by using REST API. You must have your role defined as `NAM_OAUTH2_ADMIN` in the IDP Role policy. You can access this endpoint either by login or using an access token. For more information, see Managing Client Applications.

Send an HTTPS POST request with the appropriate URI parameters to resource server endpoint URI.

Resource Server Endpoint: `https://<Identity Server URL: Port Number>/nidp/oauth/nam/resourceservers`

HTTP Method: POST

Request Parameters:

Parameter	Required /Optional	Description
name	Required	The name of the resource server.
disableJWTAccessTokenEncryption	Optional	Specify the value as true for not encrypting the access token. The value as false to encrypt the access token by using Access Manager key or resource server key.

Parameter	Required /Optional	Description
cryptoKeys	Optional	Specify the resource server's JWKS key details to encrypt the access token using this key. The parameter value as a sample JSON format is as follows: `{"jwksUri": "https://www.resourcer.server.com/crypto/jwks", "jwtaccessTokenEncryptionAlgo": { "encryptionAlg": "RSA1_225", "encryptionEnc":"A128CBC-HS2563"}`

Sample Request and Response

A sample request and response for registering resource server crypto keys in Access Manager, with line breaks for better readability and payload in JSON.

```
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
'{"name":"namResourceServer",

"cryptokeys":{"jwksUri": "https://www.resourcer.server.com/crypto/jwks",
"jwtaccessTokenEncryptionAlgo": { "encryptionAlg": "RSA1_225",
"encryptionEnc":"A128CBC-HS2563"}
}
}'
```

A successful Response

```
HTTP/1.1 200 OK
Cache-Control: no-cache, no-store, no-transform
```

3.2.3.1 Deleting a Resource Server

To delete a resource server by using REST API, you must have your role defined as NAM_OAUTH2_ADMIN in the IDP Role policy. To access this endpoint, log in or use an access token. If you use the access token, it should contain the following scope:

Scope:`urn:netiq.com:nam:scope:oauth:registration:full`

HTTP Method: DELETE

Resource Server Endpoint: `https://<Identity Server URL: Port Number>/nidp/oauth/nam/resourceservers/<resourceServerName>`

3.2.3.2 Viewing Registered Resource Servers

To view all registered resource servers by using REST API, you must have your role defined as NAM_OAUTH2_ADMIN or NAM_OAUTH2_DEVELOPER in the IDP Role policy. To access this endpoint, log in or use an access token. If you use the access token, it should contain the following scope:

Scope: `urn:netiq.com:nam:scope:oauth:registration:full`

HTTP method: Get

Resource Server Endpoint: `https://<Identity Server URL: Port Number>/nidp/oauth/nam/resourceservers`

3.2.3.3 Creating Scopes

To create a scope by using REST API, you must have your role defined as `NAM_OAUTH2_ADMIN` in the IDP Role policy. To access this endpoint, you need to log in or use an access token. If you use the access token, it should contain the following scope:

Scope: `urn:netiq.com:nam:scope:oauth:registration:full`

Resource Server Endpoint: `https://<Identity Server URL: Port Number>/nidp/oauth/nam/resourceservers/<resourceServerName>/scopes`

HTTP Method: `Post`

Request URI Parameters:

Parameter	Required/Optional	Description
scope	Required	The name of the scope.
scope_description	Required	Description of the scope. The consent page displays this text while obtaining authorization from the user.
claims	claims or attribute_set is required	The list of claims.
attribute_set	claims or attribute_set is required	Attribute name and attribute dn. Sample: { "name" : "jpeg_photo", "dn" : "cn=jpeg_photo,o=novell"}
userPermissionRequired	Optional	Boolean value. The default value is true.
adminApprovalRequired	Required	Boolean value. The default value is true. Set it to false always.
isGroupOfUserAttributes	Optional	Boolean value. The default value is false.
allowModifyInConsent	Optional	Boolean value. The default value is false.
includeAllClaimsInIDToken	Optional	Boolean value. The default value is false. If the value is true, all claims or attributes in this scope are included in IDToken.
includedClaimsInIDToken	Optional	Attribute or claim name(s). You can use comma(,) as a delimiter to specify names of more than one attribute or claim. The specified claim or attribute is included in IDToken. For example, givenName, mail

Parameter	Required/Optional	Description
includeAllClaimsInJWT	Optional	Boolean value. The default value is false. If the value is true, all the claims or the attributes in this scope will be included the access token.
includedClaimsInJWT	Optional	Attribute or claims names. Use comma (,) as a delimiter to specify names of more than one attribute or claim. The specified claims or attributes will be included in the access token. For example, givenName, mail

3.2.3.4 Modifying a Scope

To modify a scope by using REST API, you must have your role defined as NAM_OAUTH2_ADMIN in the IDP Role policy. To access this endpoint, you need to log in or use an access token. If you use the access token, it should contain the following scope:

Scope: `urn:netiq.com:nam:scope:oauth:registration:full`

The request includes the following details:

Resource Server Endpoint: `https://<Identity Server URL: Port Number>/nidp/oauth/nam/resourceservers/<resourceServerName>/scopes/<scopename>`

HTTP Method: POST

Send only those parameters that you want to modify.

3.2.3.5 Deleting a Scope

To delete a scope by using REST API, you must have your role defined as NAM_OAUTH2_ADMIN in the IDP Role policy. To access this endpoint, login or use an access token. If you use the access token, it should contain the following scope:

Scope: `urn:netiq.com:nam:scope:oauth:registration:full`

The request includes the following details:

Resource Server Endpoint: `https://<Identity Server URL: Port Number>/nidp/oauth/nam/resourceservers/<resourceServerName>/scopes/<scopename>`

HTTP Method: DELETE

3.2.3.6 Viewing Configured Scopes

You can view details of all scopes together or of a specific scope. To view a scope by using REST API, you must have your role defined as NAM_OAUTH2_DEVELOPER or NAM_OAUTH2_ADMIN in the IDP Role policy. To access this endpoint, you need to log in or use an access token. If you use the access token, it should contain the following scope:

Scope: `urn:netiq.com:nam:scope:oauth:registration:full`

Viewing Details of a Specific Scope

The request includes the following details:

Resource Server Endpoint: `https://<Identity Server URL: Port Number>/nidp/oauth/nam/resourceservers/<resourceServerName>/scopes/<scopename>`

HTTP Method: GET

Viewing Details of All Configured Scopes

The request includes the following details:

Resource Server Endpoint: `https://<Identity Server URL: Port Number>/nidp/oauth/nam/resourceservers/<resourceServerName>/scopes`

HTTP Method: GET

3.2.4 OAuth 2.0 Endpoints

To get an access token and identity token, the client invokes requests to corresponding endpoints exposed by Identity Server. Identity Server exposes the following endpoints:

- **Authorization Endpoint:** This is always contacted via a browser. This endpoint requires that the user has existing browser session with Identity Server. If no session exists at the time of request, the Authorization Endpoint redirects the user to login. This endpoint is used when a client uses the authorization code flow or implicit flow.
- T**oken Endpoint:** This is used directly by a client without involving the browser. Therefore, it is possible to get an access token offline when a user is not connected via a browser. This endpoint can issue an access token when the client provides a valid authorization code, SAML2 bearer profile for authorization grant flow, resource owner credentials, or client credentials.
- **TokenInfo Endpoint:** This is used for validating a refresh token and 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:** This is used for getting resource owner's claims. A client can send a request to this endpoint with a valid access token and get the claims that are authorized by the resource owner to share. This endpoint checks whether provided access token has valid scopes to issue the claims.
- **Revocation Endpoint:** This is used for revoking refresh tokens and its corresponding access token.

3.2.5 Other Endpoints

In addition to the basic endpoints mentioned in Section 3.2.4, “OAuth 2.0 Endpoints,” on page 30, Identity Server exposes the following endpoints:

Metadata Endpoint: This exposes basic services and options available at 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:

- OAuth 2.0 Endpoints
- ID token supported algorithms
- JWKS keys that can be used for verifying access token and ID token

Client Registration Endpoint: Developers use this to register OAuth2.0 clients through REST API. This endpoint is protected by OAuth 2.0, therefore 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, a developer must have the `NAM_OAUTH2_DEVELOPER` role defined in Identity Server.

Scope and Resource Server registration Endpoint: This is used to register, modify, and delete a scope. Users who invoke this endpoint must have the `NAM_OAUTH2_ADMIN` role defined in Identity Server to be able to register and modify the scope values.

JSON Web Key Set Endpoint: Provides the information about the signing certificate that is used by Access Manager, which can be used for verifying an access token.

3.3 Authentication

The application can request an authentication service from Access Manager by using one of the supported OAuth 2.0 authorization grant flows. Access Manager implements OpenID Connect 1.0 specification on top of these flows. Therefore, 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 the 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 the user has an active session at Identity Server with requested or default assurance level. The flows requiring active user session involves a browser redirect, therefore 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 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 and password directly, see Resource Owner Credential Grant.

Getting Identity Tokens

You can use any one of the following flows to get an identity token:

- Authorization code grant flow

- Implicit Flow
- Hybrid Flow

3.3.1 Authorization Code Grant Flow

The authorization code grant flow is a two-step process.

1. Get a short lived authorization code from Identity Server.
2. Exchange this authorization code with ID token.

The application can also request for an access token for authorization if the application makes REST API calls to resource servers.

The application can also specify minimum assurance level for the authentication method from Identity Server by using the following request parameter.

Identity Server ensures that the user is authenticated with the requested level of authentication before sending the identity token.

An identity token is a signed JSON Web Token (JWT). Signing is optional, but recommended. The token is signed when the client is configured with `ID Token Signing Algorithm` during the client registration process.

The application verifies the returned identity token as mentioned in Validating Tokens.

3.3.1.1 Getting an Authorization Code

A client can get an authorization code in by redirecting the browser to the authorization endpoint with the required query string values. See “Request Parameters” on page 32.

The response is a short life-time `authorization code` and must used only once. You can use this code to exchange identity tokens from Identity Server through the token endpoint only once by sending the necessary request parameter. See Exchanging the Authorization Code with Identity Tokens.

Multiple attempts of exchanging code for token revokes tokens that are issued earlier. The authorization code is sent 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/ Optional	Value	Description
client_id	Required		Client application ID, which is obtained at the time of the client application registration process.
response_type	Required	code	code

Parameter	Required/ Optional	Value	Description
redirect_uri	Optional		If provided, the value of this must exactly match to one of the registered URIs during the application registration process. If not provided, the browser is redirected to any of the registered redirect URIs registered during application registration.
scope	Required	OpenID	The 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 must be space separated %20 or +.
resourceServer	Optional		Specify the registered resource server name. If this parameter is available, the authorization server uses the respective configured method to encrypt the access token.
state	Recommended		An opaque value used by a client to maintain state between request and callback. The authorization server includes this value when redirecting a user-agent back to a client. This parameter is used to prevent cross- site forgery requests.
prompt	Optional	none login consent	none: No user interface is shown to a user if the user is not already authenticated. If not authenticated, an error message in one of "login_required", "interaction_required", or other is sent to the client application. This is useful if a client wants to detect whether the user has an existing session with Identity Server and has necessary consents. login: Identity Server asks for re-authentication. consent: Identity Server asks for consent even if the consent is already given.
max_age	Optional	300	Maximum authentication age at Identity Server in seconds. If a user does not log in within this time, the user is re-prompted for authentication
acr_values	Optional	/name/password/uri	If a client request contains acr_values, Identity Server maps the value to configured contracts in Identity Server and prompts the user with the contract if the user is not authenticated with the contract. The contract is not sent in ID token.
device_id	Optional		Specify the device ID that token to be associated with device.

Response Values

Identity Server responds with a HTTP 302 redirect message to requested redirect_uri in an authorization request. If the request does not contain the redirect_uri param, Identity Server redirects to one of the registered redirect_uri. The redirect response contains 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=code
&client_id=bb775b12-bbd4-423b-83d9-647aeb98608d
&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
```

3.3.1.2 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 authorization code is allowed for exchange only once. The request should be sent to the token endpoint.

The response is sent 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 validates 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 was authenticated. The client needs to verify the signature of the ID token if available and check the issuer.

Request Parameters

Specify the following parameters in the token request:

Parameter	Required/ Optional	Description
resourceServer	Optional	Registered resource server name. If this parameter is available, the authorization server uses the respective configured way to encrypt the access token.
grant_type	Required	authorization_code
client_id	Required	Client ID of the registered client.
client_secret	Optional	Client secret of the registered client. It is optional for a native application and mandatory for a web application.
code	Required	Code received in the authorization code flow.
redirect_uri	Required	This must be same as the one sent during the authorization code request.
device_id	Optional	Specify the device ID that token to be associated with device.

Response Values

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

Parameter	Required/ Optional	Description
token_type	Required	The type of the token. The authorization server supports only the Bearer type.
access_token	Required	An 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 contains an ID token.
scope	Optional	The list of scopes that a user has authorized. This can contain all scopes the client requested.
state	Optional	if the `state` parameter was available in the client authorization request, the same state value is sent in the response.

NOTE: Ensure that you do not use the `Expect: 100-Continue` header in the request when using a multi-node Identity Server cluster setup. If the request contains this header, you may experience HTTP 400 Bad Request.

If you are using CURL, use "-H 'Expect:'" or do not include IDP cookies.

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"
}
```

3.3.2 Implicit Grant

Get an access token and ID token by sending an HTTPS GET or POST request with the appropriate URI parameters to the authorization endpoint base URI. The ID token issued only if the scope is having OpenID param value. The ID token can be signed and encrypted based on the client's registration. 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 a 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 to provide the user's email address and profile to your application.

Sample implicit request/response:

```
https://example.netiq.com/nidp/oauth/nam/
authz?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 request:

```
https://client.example.com/callback#token_type=bearer&access_token=/
wEBAAUFACAjDfPtn d/zlOWPpN/
kV1Jtt3nxCPtzHyUH~&expires_in=3600&id_token=eyJhbGciOiJSUzI1NiJ9.eyJp
c3MiOiJo&scope=email&state=s1234
```

Token Request URI Parameters

The token request needs to include the following query parameters and the request needs to be sent to the authorization endpoint:

Parameter	Required/ Optional	Description
response_type	Required	The possible values are `token`, `id_token`, and `token id_token`. The id_token is issued only if the scope contains the OpenID value. If the value is sent as `token`, then the authorization server issues only an access token to the client. If the value is `id_token`, then authorization server issues only id_token in the response. If the value is `token id_token`, then authorization server issues both access token and ID token in the response.
resourceServer	Optional	Specify the registered resource server's name. If this parameter is available, the authorization server uses the respective configured way to encrypt the access token.
client_id	Required	Client application ID that is obtained at the time of the client application registration process.
redirect_uri	Optional	If provided, the value of this parameter must exactly match to one of URIs of the registered application.
scope	Optional	Scopes supported by the authorization server. You can get `scopes_supported` at authorization server's OpenID Metadata endpoint. For the ID token, OpenID needs to be available in the scope. Specify multiple scope values separated with space %20.
state	Required	An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when the response is sent to the client. The parameter prevents cross-site forgery requests.
nonce	Required	String value used to associate a client session with an ID token, and to mitigate replay attacks.

Parameter	Required/ Optional	Description
prompt	Optional	A 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 following are defined values: ◆ **none**: The authorization server does not display any authentication or consent user interface pages. An error is returned if an end user is not authenticated or the client does not have preconfigured consent for the requested claims, or does not fulfill other conditions for processing the request. ◆ **Login:** The authorization server prompts the end user for re-authentication. If it cannot re-authenticate the end user, it returns an error to the client. ◆ **Consent:** The authorization server prompts the end user for consent before returning information to the client. If it cannot obtain consent, it returns 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.
device_id	Optional	The device ID of the device with which the token will be associated.

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 is the `redirect_uri` parameter sent in the request. Only the browser can read the fragment values and these are never sent to redirect_uri.

Parameter	Required/Optional	Description
token	Required	The type of the token. The authorization server supports only bearer.
Access_token	Based on the response type value	If the response type value is `token` or `token id_token`, the authorization server sends an access token.
id_token	Based on the response type value	If the response type value is `id_token` or `token id_token`, the authorization server sends an ID token.
scope	Optional	The user given consent scope names.
state	Optional	If the `state` parameter was available in the client authorization request, the same state value is sent in the response.

3.3.3 Hybrid Flow

Get a combination of tokens or code, such as access token, ID token, or authorization code, based on the `response_type` parameter by sending an HTTPS GET or POST request with the appropriate URI parameters to the authorization endpoint base URI. The hybrid flow works and the ID token is issued only if the scope contains the OpenID param value. An ID token can be signed and encrypted based on the client's registration. To access any of the user's attributes or getting permissions to access the user's resource pages, you can include the request in the scope parameter.

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

Response types that are allowed in hybrid flows, in any combination, are as follows:

response_type	Token Generated
code id_token	Authorization code and ID token
code token	Authorization code and access token
code id_token token	Authorization code, ID token, and access token

Sample hybrid flow request/response: `https://example.netiq.com/nidp/oauth/nam/authz?`

```
response_type=code 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, 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 this request:

```
https://client.example.com/
callback#code=ey2hgwshg223jdsw&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/ Optional	Description
response_type	Required	The possible values are `code token`, `code token id_token`, `code id_token`, and other combinations of the three values. It should be space separated.

Parameter	Required/ Optional	Description
resourceServer	Optional	Specify the name of the registered resource server. If this parameter is available, the authorization server uses the respective configured way to encrypt the access token.
client_id	Required	The client application ID that is obtained at the time of the client application registration process.
redirect_uri	Required	The value of this must parameter must exactly match with one of URIs of the registered application.
scope	Required	Scopes supported by the authorization server. Get `scopes_supported` at authorization server's OpenID Metadata Endpoint. It must contain the `openid` scope value. You can add multiple space separated scope values.
state	Optional	An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when the response is sent to the client. The parameter prevents cross-site forgery requests.
nonce	Required	A string value used to associate a client session with an ID Token, and to mitigate replay attacks.
prompt	Optional	A 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 following are defined values: ◆ **none:** The authorization server must not display any authentication or consent user interface pages. An error is returned if an end user is not authenticated or the client does not have preconfigured 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 re-authentication. If it cannot re-authenticate the end user, it must return an error to the client. ◆ **consent:** The authorization server should prompt the end user for consent before returning the information to the client. If it cannot obtain consent, it must return an error, typically consent_required.
max_age	Optional	Maximum authentication age. Specify the allowable elapsed time in seconds since the last time the user was authenticated by the OP.
device_id	Optional	Specify the device ID of the device that token has to be associated with.

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 is the `redirect_uri` parameter sent in the request. The fragment values can only be read by the browser and never sent to redirect_uri.

Parameter	Required/Optional	Description
token_type	Based on response type value	The type of the token. The authorization server supports only bearer.
access_token	Based on response type value	If the response type value is `code token`, `code id_token token`, or its combination, the authorization server sends an access token.
id_token	Based on response type value	If the response type value is `code id_token`, `code id_token token`, or its combination, the authorization server sends an ID token.
scope	Optional	The user given consent scope names.
state	Optional	if the `state` parameter was available in the client authorization request, the same state value is sent in the response.
expires_in	Based on response type value	Expiration time of the access token in seconds since the response was generated.
code	Required	Authorization code

3.4 Authorization

Access to the resources hosted in the resource server can be protected 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 contains necessary 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 or the grants explained in *Other Grants*:

- Authorization Code Grant
- Implicit Grant
- Refresh Token
  - Resource Owner Credentials Grant
  - Client Credentials Grant
  - SAML2 Bearer Profile for Authorization Grant

3.4.1 Authorization Code Grant

This is same as getting the identity token explained in Authorization Code Grant Flow.

3.4.2 Implicit Grant

This is same as getting the identity token explained in Section 3.3.2, “Implicit Grant,” on page 36. To get an access token, the request must contain `response_type value as token`.

3.4.3 Refresh Token

The Authorization Code Grant requires that the user is available on the browser and has an active session with Identity Server. Therefore, it is called as the online flow.

Sometime, the client might need access to resources even if the user is not available online. For example, when a client wants to perform batch processing on resources owned by a user, it might need to have a longer lifetime of access token. Access tokens usually have shorter lifetime. The refresh tokens have longer lifetime. Using the refresh tokens, clients can ask for fresh access tokens. As 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/ Optional	Description
grant_type	Required	Must be refresh_token.
client_id	Required	The client application ID that is obtained at the time of the client application registration process.
client_secret	Optional	The client secret that is obtained at the time of the client application registration process. The client secret is optional for a native application, but mandatory for a web application.
refresh_token	Required	refresh_token that is obtained during authorization grant, resource owner credentials.
scope	Optional	The list of the scope names separated by space.
device_id	Optional	Specify the device ID that token to be associated with device.
resourceServer	Optional	The name of the registered resource server. If this parameter is available, the authorization server uses the respective configured way to encrypt the access token.

Refresh Token Response Values

A successful request to token endpoint with refresh token results in a response containing a JSON object with the following values:

Parameter	Required/Optional	Description
access_token	Required	Access token
refresh_token	Optional	Re-issue a refresh token
token_type	Optional	Token type that is supported by the authorization server
expires_in	Required	The validity time of an access token
token_scope	Optional	The scopes granted to a 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/voByU9Z2awkCbfp
LZTzpUqFspZ4xrJc/TcNAl3hktfRDJgOUEHUkdyO/FoWxmTn3NrHL0K8kNPQo7nm3kyUSyjpxxvjVw
SOPtVmNl94AXOIxqObYpLoRgpqqeO8TUltvQlk9zMNkAmHscPTYFwMrzHE@B98kIrZ1b266eSbuAmL
r4y1guAx0yYs1XhboFd97I6mabGXDqeAjjpx/DTZBTCptA/LlIJgN10jMwik7x9nZZ3wjv16/4hw8G
UHaS09uHXqqtF3S0pJ6/aM/hsWAgkcZeOhliPGXV8T7tjMmc8V1t4mIzuOagzN0LbaclD1OBkndIKC
OcqJiiMMRDZNEHBjwoOXc~",
"token_type":"bearer",
"expires_in":3599, "scope":"profile email"
}
```

3.5 Validating Tokens

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

By default, access tokens are signed and encrypted by using Access Manager encrypted keys. You can choose to encrypt the access token by using resource server encrypted keys or disable encryption.

You can verify the access token received in earlier flows based on how the token is encrypted.

Use one of the following ways to verify the token:

- When an access token is signed and encrypted by using Access Manager encrypted keys, the token can be validated by sending a request to the TokenInfo endpoint of Identity Server. This scenario requires the token to be sent to Identity Server for verification.
- When access token is encrypted by using resource server encrypted keys, the resource server can validate the token by decrypting the token, verifying the signature, trusting the token that is issued by the trusted Identity Server, or by sending the decrypted token to TokenInfo endpoint of Identity Server. This scenario does not require token to be sent to Identity Server instead the resource server can verify the token by itself. For detailed sample code and tool for validating the JWT access token, see JWT Validation tool. The access token contains a claim called "_pvt" that is an Access Manager encrypted private claim and can be decrypted and used only by Access Manager.
- When access token is not encrypted, the resource server can verify the signature of the token and trust the token that is issued by trusted Identity Server.

To validate access tokens and refresh token that are signed and encrypted by using Access Manager encrypted keys, send a request to TokenInfo endpoint. Refresh tokens are always encrypted by using Access Manager encryption keys. You can refer to the following sample to send the request:

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

The TokenInfo endpoint supports both HTTP GET and POST methods.

The request must contain the token in the authorization header as follows:

Authorization: Bearer access_token

Response Values

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

Parameter	Required	Description
expires_in	Yes	Time in seconds the token is valid from now
user_id	Yes	The user to whom the token was issued
scope	Yes	The list of scope values the token holds

Sample Request and Response

```
Request:
HTTP/1.1 GET /nidp/oauth/nam/tokeninfo
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": []
}
```

3.6 Revoking Tokens

The client application can programmatically revoke its token in the following scenarios:

- When an end user logs out.
- When an application is uninstalled.
- To notify Identity Server that a previously obtained refresh token is no longer needed.

The refresh token received in earlier flows can be revoked by sending a request to the revocation endpoint of Identity Server.

IMPORTANT: Only the refresh tokens that are generated by Access Manager Version 4.4 or later can be revoked.

The URL to revoke is `https://idpbaseurl.com/nidp/oauth/nam/revoke`.

The request should contain the refresh token and client credentials in HTTP request parameters as mentioned in the following table:

Parameter	Required/ Optional	Description
client_id	Required	The client application ID that is obtained at the time of the client application registration process.
client_secret	Optional	The client secret that is obtained at the time of the client application registration process. The client secret is optional for a native application, but mandatory for a web application.
Token	Required	refresh_token that is obtained during authorization grant, resource owner credentials, client credentials flow

Response Values

- Identity Server responds with the HTTP status code 200 OK if the token has been revoked successfully or if the client submitted an invalid token.
- Identity Server returns the error code unsupported_token_type when the provided token is not a refresh token.
- If Identity Server responds with the HTTP status code 503, the client must assume that the token still exists and may retry revoking the refresh token after a reasonable delay.

Revoking Token Issued to a Device

When Mobile Access SDK is not used for on-boarding and off-boarding devices, the token can be manually associated with a device. This can be done by providing additional parameter `device_id` while requesting for an access token. Such manually associated tokens can be revoked by using the revocation endpoint.

The URL to revoke tokens that are issued to a device is:

`https://idpbaseurl.com/nidp/oauth/nam/revoke/<device_id>`

HTTP Post

Content-Type: application/x-www-form-urlencoded (Optional)

Request Parameters

Parameter	Required	Description
userstore_name	Yes	Specify the name of the user store.
user_dn	Yes	Specify the user's dn to whom the token issued.

Response Values

```
HTTP 200 OK
{
"status": "Successfully revoked token(s) issued to this device."
}
```

Sample Request

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

```
HTTP/1.1 POST /nidp/oauth/nam/revoke/andriodtest_1401
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
'userstore_name=namsignboxuserstore
& user_dn=cn%3Dharry%2Co%3Dnovell'
```

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
{
"status": "Successfully revoked token(s) issued to this device."
}
Error Response
```

When an invalid device id specified or device had not been associated with any token, returns HTTP 404 NOT FOUND with error response

```
{
"error": "invalid_request",
"error_description": "Invalid device ID or no tokens to revoke for this
device."
}
```

3.7 Authorization Code Grant Flow with PKCE

The native apps use Authorization Code with PKCE OAuth 2.0 grant to mitigate vulnerability with the authorization code grant flow. To implement PKCE flow client must generate random secret and store. Using random secret, client has to create code verifier and code challenge. (rfc7636 (https://tools.ietf.org/html/rfc7636))

1 A client sends the code challenge as part of the OAuth 2.0 Authorization request with following additional parameters:

Parameter	Required / Optional	Description
code_challenge	Required	Code challenge parameter if PKCE flow has to be initiated.
code_challenge_method	Optional	The default value is `plain`. The value can be `plain` or S256.

2 Returned Authorization Code is associated with `code_challenge` and `code_challenge_method`.

3 The client sends an access token request to the token endpoint with additional parameter.

The following additional request parameters can be used along with Authorization Code grant flow:

Parameter	Required	Description
code_verifier	Yes	Code verifier parameter is required if Authorization Code is requested using PKCE flow.

4 The server verifies `code_verifier` before returning the token.

PKCE flow error messages

```
PKCE verification failed:
{
"error": "invalid_grant",
"error_description": "Either invalid authorization code or invalid code
verifier, PKCE verification failed"
}
{
"error": "invalid_grant",
"error_description": "PKCE verification failed because either code
challenge is null or code challenge method is not supported"

}
```

Example:

PKCE initiate request to Authorization endpoint:

```
[https://<<IDP>>:8443/nidp/oauth/nam/
authz?code_challenge=WsEH2Rr4lWdciBEbCuHVlH_UIBUGFPRbDXcPsb-
Pl74&code_challenge_method=S256&scope=profile&response_type=code&redir
ect_uri=<<Redirec URI>>&client_id=484fd33f-12b0-44c4-bbf5-82bae803b71d
"
PKCE flow Token request parameters to Token Endpoint:
code=<<authorization code received from authorization endpoint>>
&grant_type=authorization_code&redirect_uri=<<Redirect
URI>>&client_id=484fd33f-12b0-44c4-bbf5-
82bae803b71d&code_verifier=0ak1mD3loHOy1ZksmyoO1fQEhRBEuzGYbkQqKFe1Ny0
```

3.8 Other OAuth 2.0 Grants

3.8.1 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. 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/ Optional	Description
resourceServer	No	The name of the registered resource server. If this parameter is available, the authorization server uses the respective configured way to encrypt the access token.
client_id	Required	The client application ID that is obtained at the time of the client application registration process.
client_secret	Optional	This is optional for a native application, but mandatory for a web application.
grant_type	Required	Specify `password` as the value for this parameter.
username	Required	The user login name.
password	Required	The user login password.
scope	Optional	Scopes supported by the authorization server. Get `scopes_supported` at the authorization server's OpenID Metadata Endpoint. For the ID token, OpenID should be available in the scope. You can add multiple scope values with space separated %20 or +.

Parameter	Required/ Optional	Description
acr_values	Optional	If a client request contains the `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 the ID token. Use space as delimiter to specify more than one contract URI for acr_values. In this case, Identity Server executes contracts in the sequence 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	OAuth 2.0 access token.
token_type	The type of token returned. At this time, this is always `Bearer`.
expires_in	The remaining lifetime of an access token.
scope	Scopes requested. The access token allows you access to these scopes.
refresh_token	The refresh token is returned if a client application is registered for it. This token can be used to refresh the access token when it expires.

Sample Request and Response

The following is 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 occur, HTTP Status 400 is returned with the JSON response containing `error` and `error_description`.

The following is a 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"
}
```

3.8.2 Client Credential Grant

The client credentials can be exchanged for an access token. To get an access token, send an HTTPS POST request with the appropriate URI parameters to the token endpoint base URI. 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/ Optional	Description
client_id	Required	The client application ID, which is obtained at the time of the client application registration process.
client_secret	Optional	The client secret is optional for a native application, but it is mandatory for a web application.
grant_type	Required	Specify `client_credentials` as value for this parameter.

Response Values

Parameter	Description
access_token	OAuth 2.0 access token.
token_type	The type of token returned. At this time, this is always 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 occur, HTTP Status 400 is returned with the JSON response containing `error` and `error_description`.

3.8.3 SAML 2.0 Bearer Profile for Authorization Grant

The SAML 2.0 assertions can be exchanged for access token. The Consent page will not be shown to users for authorizing scopes. The access token allows you to access only those scopes that are previously approved by the user. To get an access token, send an HTTPS POST request with the appropriate URI parameters to the token endpoint base URI.

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/ Optional	Description
client_id	Required	The client application ID that is obtained at the time of the client application registration process.
grant_type	Required	Use `urn:ietf:params:oauth:grant-type:saml2-bearer` as the value for this parameter.
Assertion	Required	Use a single base64url encoded SAML2.0 Assertion as the value for this parameter.
client_secret	Optional	The client secret value.
scope	Optional	Scopes supported by the Authorization server. Get `scopes_supported` at authorization server's OpenID Metadata Endpoint. Specify multiple scope values with space separated %20 or +.

Response Values

Parameter	Description
access_token	OAuth 2.0 access token.
token_type	The type of token returned. At this time, this is always `Bearer`.
expires_in	The remaining lifetime of the access token.
scope	Requested scopes that are pre-approved by the user.

Sample Request and Response

The following is a sample request with whitespace for readability:

```
HTTP/1.1 POST /nidp/oauth/nam/token?
&grant_type= urn:ietf:params:oauth:grant-type:saml2-bearer
&client_id=bb775b12-bbd4-423b-83d9-647aeb98608d
&assertion=MPHnbWxv01….SY2
&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: /
arul
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 occur, HTTP Status 400 is returned with the JSON response containing `error` and `error_description`.

Response Values

The following is a sample error response with whitespace for readability:

```
HTTP/1.1 400 Bad Request Content-Type: application/json
{
"error":"invalid_grant",
"error_description":"Audience validation failed"
}
```

3.9 Attribute Service

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

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

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

Parameter	Description
sub	Unique ID identifying the subject. This is GUID of the user.

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

For example, if the client has requested the `email` scope, the UserInfo endpoint returns 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"
}
```

4 Component Statistics API

Identity Servers and Access Gateway systems provide APIs to retrieve the statistics of that system. These are precursors to the Administration API and provide statistics. Both APIs provide the same data, but the difference is in the invocation point. The Component statistics API can be called for each device IP and the Administration API can be called for Administration Console to retrieve statistics of all Identity Servers and Access Gateways in the system. Therefore, it is recommended to use the Administration APIs.

For more information, see Component Statistics Through REST APIs in the NetIQ Access Manager 5.0 Administration Guide.

5 AWS Auto Scaling API

As part of the auto scaling Access Manager process, you require to automate some of the management tasks. Access Manager provides APIS for automating the following activities during auto scaling on AWS:

- Importing Devices to Administration Console
- Adding Devices to the Cluster
- Removing Devices from Administration Console
- Updating Servers in the Cluster
- Additional APIs

NOTE: For more information about how to use these APIs, refer to the supporting scripts included in the sample auto scaling solution provided along with Sample Auto Scaling Deployment of Access Manager on AWS.

5.1 Importing Devices to Administration Console

5.1.1 Importing Identity Server

Use the `reimport_nidp.sh` script available in the `/opt/novell/devman/jcc/conf` location to import Identity Server.

You might require to regenerate jccid of the device before importing it to avoid the jccid conflicts.

5.1.2 Importing Access Gateway

Use the `reimport_ags.sh` script available in the `/opt/novell/devman/jcc/conf` location to import Access Gateway.

You might require to regenerate jcc_id of the device before importing it to avoid the jcc_id conflicts.

5.2 Adding Devices to the Cluster

5.2.1 Adding Identity Servers to the Cluster

Request: `curl -X POST "$rest_url" -u $ADMIN_FQDN:$ADMIN_PASSWORD`

Where rest_url=https://$ADMIN_CONSOLE_IP:8443/amsvc/v1/idpclusters/$CLUSTER_ID/devices/idp-$jcc_id

5.2.2 Adding Access Gateway server to the cluster

Request: `curl -X POST "$rest_url" -u $ADMIN_FQDN:$ADMIN_PASSWORD`

Where rest_url="https://$ADMIN_CONSOLE_IP:8443/amsvc/v1/agclusters/$MAG_CLUSTER_ID/devices/ag-$jcc_id"

5.3 Removing Devices from Administration Console

5.3.1 Removing Identity Server from Administration Console

Request: `curl -k -X DELETE "$rest_url" -u cn=$ADMIN_NAME,o=novell:$ADMIN_PASS`

Where rest_url=https://$AC_IP:8443/amsvc/v1/idpclusters/$IDP_CLUSTER_ID/devices/$IDP_DEVICE_ID"

5.3.2 Removing Access Gateway from Administration Console

Request: `curl -k -X DELETE "$rest_url" -u cn=$ADMIN_NAME,o=novell:$ADMIN_PASS)`

Where rest_url=https://$AC_IP:8443/amsvc/v1/agclusters/$AG_CLUSTER_ID/devices/$AG_DEVICE_ID"

5.4 Updating Servers in the Cluster

5.4.1 Updating the Identity Server Cluster

Request: `curl -k --ciphers ALL -u $ADMIN_FQDN:$ADMIN_PASSWORD -X PUT -H "Content-Type: application/json" -d "{\"update\": \"all\"}" "$rest_url "`

Where rest_url=rest_url="https://$ADMIN_CONSOLE_IP:8443/amsvc/v1/idpclusters/$CLUSTER_ID"

5.4.2 Updating the Access Gateway Cluster

Request: `curl -k --ciphers ALL -u $ADMIN_FQDN:$ADMIN_PASSWORD -X PUT -H "Content-Type: application/json" -d "{\"update\": \"all\"}" "$rest_url "`

Where rest_url="https://$ADMIN_CONSOLE_IP:8443/amsvc/v1/agclusters/$MAG_CLUSTER_ID"

5.5 Additional APIs

5.5.1 Getting the ID of a Specific Cluster

Getting the ID of an Identity Server Cluster:

Request: `curl -k -X GET $rest_url  -u cn=$ADMIN_USERNAME,o=novell:$ADMIN_PASSWORD)`

Where rest_url="https://$ADMIN_CONSOLE_IP:8443/amsvc/v1/idpclusters"

Getting the ID of an Access Gateway:

Request: `curl -k -X GET $rest_url -u cn=$ADMIN_USERNAME,o=novell:$ADMIN_PASSWORD)`

Where rest_url="https://$ADMIN_CONSOLE_IP:8443/amsvc/v1/agclusters"

5.5.2 Getting the Number of Active Sessions in Identity Server

Request: `curl -k -X GET $rest_url -u cn=$ADMIN_NAME,o=novell:$ADMIN_PASS`

Where rest_url="https://$AC_IP:8443/roma/rest/$IDP_CLUSTER_ID/sessions/devices/$IDP_DEVICE_ID"