Honeywell

OAuth Integration Guide

Leveraging the Honeywell SSO Federation Service Using OAuth and OpenID Connect

Document Version: 2.0
Effective Date: 27-January-2017

Contents

1. Preface

This document provides the steps necessary to integrate with the Honeywell SSO Federation service (herein referred to as simply the SSO Service) using OAuth.

Although there are multiple versions of OAuth, the SSO Service only supports OAuth 2.0 and this version is not backward-compatible with older versions.

2. Overview

OAuth is a standard authorization mechanism for an application (Client) to gain access to another server's resources (Resource Server or RS) on behalf of a Resource Owner (User). It specifies a process for a Resource Owner to authorize third-party access to his resources *without sharing credentials* to the Client. The OAuth processes are managed through an Authorization Server (AS), which is the SSO Service in this case.

3. OAuth Grant Types

The Authorization Server "grants" access to the Client in a few ways. As the AS, the SSO Service provides three standard **grant types**:

Grant Type	Use Case
Authorization Code	This is the most common grant type used for most web and mobile application scenarios that want to call REST web services or leverage OpenID Connect. The user agent (browser) is used to transport an intermediate code (authorization code), which is then exchanged for OAuth tokens.
Implicit	This is used less often. It is most suitable for a so-called "insecure" Client – that is, one that can't keep Client credentials confidential. An example would be a web page that leverages a scripting language such as JavaScript – most often a Single Page Application (SPA). Persistence of the access token is problematic with this grant type. Generally, it should only be persisted as long as the application session exists within the Client.
Client Credentials	This is used when the Client needs to access its own resources from a Resource Server instead of those resources associated with a Resource Owner. A Client may still get resources for a particular Resource Owner, but that requires a separate trust relationship/agreement between the Client and the Resource Server.

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

4. Integration

There are four primary steps to leverage OAuth with the SSO Service:

1. Register Application
2. Configure the SSO Service
3. Client Implementation
4. Resource Server Implementation

4.1. Register Application

An application is considered either a Client or a Resource Server (RS), but not both.

An application is considered the Client if it needs to access resources from a RS. In this case, the application must first register with the SSO Service so that the SSO Service recognizes it as a valid OAuth Client. A *Honeywell employee* can register a Client by following the directions on the SSO Service's self-service site. During the registration process, the requester fills out a requirements XLS sheet containing any information about the application. Much of the information required in the registration process are covered in the following sections of this document.

If an application or site provides resources to Clients, it is considered a RS. *A Resource Server must also register with the SSO Service* so that the SSO Service recognizes it as a valid OAuth Resource Server. As mentioned, an application cannot be both a Client and a RS; thus, both the Client and RS must register separately.

The SSO Service provides two environments; each has its own AS.

- **QA – Quality Assurance**
  This is used for anything that is not considered Production. A Client or RS may have multiple registrations in this environment. For example, a Client may have both a development (DEV) and QA version; in this case, the Client may have two different integrations – with two different identifiers.

- **PROD – Production**
  This is for end-users and live business data. Generally, a Client or RS will be registered only once in this environment.

4.2. Configure the SSO Service

There is no way for an Application Manager to configure the OAuth-specific attributes of the application. Instead, the Application Manager must work with the SSO Service team. While registering the Client or RS, the requirements provided during the registration process are used by the SSO Service team to configure the SSO Service.

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

4.2.1. Identification

Each Client/RS will be uniquely identified with a number – the "application ID" (e.g., 1234). This app ID is used when programming the login page and during other SSO functionality. Each Client will also have a corresponding "Client ID". This is merely a string of "Client_<appid>" (e.g., "Client_1234"). *A Resource Server will also have a Client ID even though it isn't necessarily considered a Client.*

Each Client/RS will also be assigned a Client Secret. Much like a password, this is a string that is used to authenticate the Client to the SSO Service. NOTE: With the Implicit grant type, this secret *should not* be used.

As with all SSO integrations, the Client must implement a login page or upload one within the SSO Service administrative console. Please refer to the New Login Panel integration guide for more details; this is located on the SSO Service's self-service site.

4.2.2. Scopes

The Authorization Server, as its name implies, authorizes a Client to obtain an **access token**, and in many cases this involves the direct authorization from the Resource Owner. As there are many Resource Servers (RS), a Client must be configured as to which RS it can access; this is done by using scopes. A scope can be considered a named set of permissions to a RS.

When a Client requests an **authorization code** (i.e. using an authorization code grant) or access token (i.e., using an implicit grant), it requests one or more scopes. It may only ask for those scopes to which it is authorized as configured within the SSO Service. A Client doesn't have to request for all scopes to which it is authorized. As a best practice, in any given request, it chooses only the scope(s) it requires. No matter which scopes are requested, even if the Client does not ask for any explicit scopes, it is still granted what is known as a "default scope", that is, a scope with no name and that is not associated with any RS. By itself, the "default scope" may not be useful as each RS should verify that the Client has a scope that is associated with the RS.

When registering a Client, it may need access to many different scopes, but it must choose at least one. It may choose any OpenID Connect scope and/or any scope associated with a RS.

The SSO Service will have a list of all published scopes. That list will include various metadata about the scope, including the name, authorization text, scope description, service environment, hosting Authorization Server, Resource Server owner, whether it needs owner approval to use it, and whether or not the owner requires the authorization page.

4.2.2.1. Name

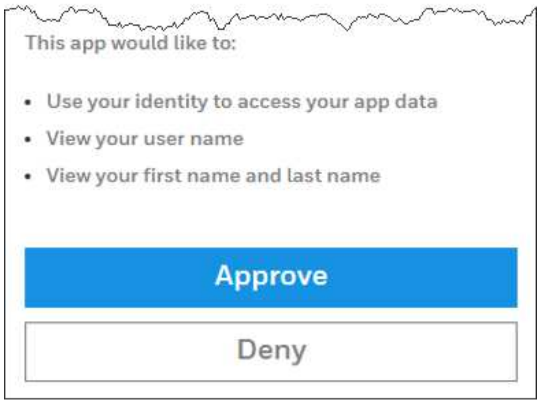
In the SSO Service, each RS will generally have one scope. To keep things unique, the syntax of the scope name will be the URL of the RS. When requesting integration with the SSO Service, the RS owner will collaborate with the SSO team to define the scope name associated with the RS.

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

For example, if an RS has a set of REST web services, the scope name may be the URL path that is common amongst them: “`https://example.honeywell.com/path/service`”

4.2.2.2. Authorization Text

Each scope must have a short (i.e., one line) and meaningful description that is displayed to the Resource Owner (i.e., user) when asking for authorization. The following is part of the authorization page that shows the various scope descriptions:



This app would like to:

- Use your identity to access your app data
- View your user name
- View your first name and last name

The first listed is the description of the “default scope” and will always be shown. The second and third listed are for the “openid” and “profile” OpenID Connect scopes and would show up only if those two scopes were requested.

4.2.2.3. Scope Description

An explanation of the scope’s purpose or any other information the scope owner wants Clients to know. This is not displayed anywhere outside of the published list of scopes.

4.2.2.4. Service Environment

This is the environment associated with the RS. For example, this could be “DEV”, “SIT”, “QA”, “Staging”, or “PROD”. This is not the SSO Service environment within which the RS/scope is registered. For example, both a DEV and QA version of a particular RS may be registered within the QA SSO Service environment.

4.2.2.5. Hosting Authorization Server

This is the FQDN of the AS that hosts the RS associated with the scope. It will either be “PROD – cwa.honwyell.com” or “QA – qcwa.honeywell.com”.

4.2.2.6. Resource Server Owner

Each RS has an owner – identified by an email – and any scope associated with the RS is owned by that same contact. This contact must be a Honeywell resource (i.e., internal user). The owner must approve Client requests, if applicable.

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

4.2.2.7. Approval

The RS owner must determine whether it must approve any Client request to use the scope (only one time during Client configuration). This can have two values:

- **Yes** – Client must get approval to be configured to access this scope
- **No** – Client doesn't need approval to be configured to access this scope

The RS owner must determine whether Clients must use an authorization page. An RS owner may choose one of three values:

- **Open** – The RS owner doesn't mind whether or not Clients use an authorization page. The Client doesn't need approval to skip the authorization page.
- **Approval** – The RS owner normally wants Clients to use an authorization page. Approval from the RS owner is required (only one time during Client configuration) in order to skip the authorization page.
- **Always** – Client ***must always*** use an authorization page.

4.2.3. Authorization Page

By default, the authorization code and implicit grant types require the Resource Owner to actively authorize/approve the Client to obtain an access token and perform actions according to the scopes being requested. This user interface is presented after the user is authenticated but before the access token is issued. From a security perspective, this makes sense since the Client is acting on behalf of the Resource Owner – technically *performing transactions as that Resource Owner*. In certain situations, however, it may be acceptable to skip this authorization step for a specific Client. The following criteria must be satisfied in order to have a Client configured to skip this step:

- The Client must be created and managed by Honeywell.
- The Client must get approval to skip the authorization page from the RS owner for each scope configured for the Client.
    - An approval for a given scope is automatic if the scope's owner has identified that an authorization page is not required for that scope (i.e., "Open" setting).
    - For example, if a Client is configured with three scopes, and only two of them require approval, then the Client needs to get approval from the owner of the two scope that require it.

If a Client has previously been configured to skip the authorization page and then requests a new scope, it must get similar approval for that new scope. If the request is rejected for any reason, the Client must decide either to start using an authorization page (the SSO team will change the Client configuration when asked) or not use the desired, new scope (i.e., no change necessary).

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

4.3. Client Implementation

As mentioned earlier, the SSO Service provides two environments; the fully-qualified domain name (i.e., base FQDN) for each is as follows:

Environment	Base FQDN
QA	qcwa.honeywell.com
PROD	cwa.honeywell.com

4.3.1. Authorization Code Grant Type

The authorization code grant type will require the following parameters from the Client:

Parameter	Description	Mandatory
client_id	Client ID configured in the SSO Service *(case-sensitive)*	Yes
client_secret	Client Secret configured in the SSO Service *(case-sensitive)*	Yes
grant_type	`authorization_code` *(case-sensitive)*	Yes
response_type	`code` *(case-sensitive)*	Yes
redirect_uri	URI where the authorization code request and access token request will be sent after success. This should be URL-encoded.	No*
scope	The scope of access requested. If multiple scopes are requested, it is space-delimited. *(case-sensitive)*	No
state	A string of characters that the Client may use to maintain state between the request and callback. If included, the AS returns this and the given value when redirecting the user-agent back to the Client.	No

* The SSO service can support multiple redirect URIs for a Client, although each SSO Service transaction will use only one of them. If there are multiple URIs defined, the **redirect_uri** parameter is mandatory. If a Client's redirect URI is configured in the SSO Service using a wildcard (i.e., matching a pattern), it is considered as having multiple redirect URIs. If omitted during the request, and only one redirect URI is configured for the Client, then the AS will use the configured value.

The OAuth endpoints used are:

Type	Endpoint
Authorization	https://<base_FQDN>/as/authorization.oauth2
Token	https://<base_FQDN>/as/token.oauth2

As an example, consider the following Client information:

Parameter	Example Value
client_id	`Client_1234`
client_secret	`appsecret1234`
response_type	`code`
redirect_uri	`sample://oauth2/code/cb`

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

To initiate the process, the Client application will redirect the browser to the **authorization endpoint**. This redirect will contain the applicable attributes in the query string of the URL. Using the above example parameters, the Client will redirect the user to the following URL:

```
https://<base_FQDN>/as/authorization.oauth2?client_id=Client_1234&
response_type=code&redirect_uri=sample%3A%2F%2Foauth2%2Fcode%2Fc
```

This will first initiate an authentication process using the browser (i.e., the user agent). Upon successful authentication, it will ask the user for authorization. Once the user successfully completes authentication and authorization, the browser is redirected with an **authorization code** to the redirect URI as defined in the request.

In the example, a successful authorization request redirects the browser to the following URL:

```
sample://oauth2/code/cb?code=XYZ...123
```

The Client extracts the **code** from the query string included in the URL.

HTTP Status Codes

As with all REST calls, upon success, a 200 HTTP response status code is returned. An error condition from the authentication or authorization process will return a 400 (bad request) or 401 (unauthorized) HTTP response status code. No specific error information is sent by SSO service. The caller (i.e., Client or RS) must check the HTTP response status code and take appropriate action as per its business requirements.

Mobile

A mobile application may not have a web page which can be used as the redirect. In this case, the **redirect_uri** may be a pre-defined URL provided by the SSO Service. This URL/page will not return any body content, but it will return a 200 HTTP response status code (or 400/401 in the case of an error), which is still required in a mobile scenario.

```
https://<base_FQDN>/admin/OauthResponse.jsp
```

If the user cancels the login box, the browser is redirected to a different, preconfigured URL:

```
https://<base_FQDN>/admin/OauthResponse.jsp?logindenied
```

The mobile application reads the **logindenied** parameter and decides how to proceed.

Swap the authorization code for an access token

The final step for the Client is to swap the **authorization code** received in the previous step for an **access token** that can be used to authorize access to resources.

For this to occur, the Client makes a HTTP POST request to the **token endpoint** on the AS. By limiting the exposure of the access token to a *direct HTTPS connection between the Client and the Authorization endpoint*, the risk of exposing this access token to an unauthorized party is reduced.

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

This request will use the following parameters sent in the body of the request:

Parameter	Example Value
grant_type	`authorization_code`
code	*<the authorization code received in the previous step>*
redirect_uri	*<the redirect_uri of the previous request, if it was used>*

Both **grant_type** and **code** are required. If the **redirect_uri** was provided while obtaining the authorization **code**, the **redirect_uri** is required as well.

This request should authenticate as the Client using HTTP BASIC authentication. In this example, the following request is made to retrieve the **access token**.

```
POST https://<base_FQDN>/as/token.oauth2 HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic Q2xpZW50XzEyMzQ6YXBwc2VjcmV0MTIzNA==
grant_type=authorization_code&code=XYZ...123
```

NOTE: The “Authorization” header requires a base64-encoded version of “client_id:client_secret”. If you base64-encode “Client_1234:appsecret1234” you get “Q2xpZW50XzEyMzQ6YXBwc2VjcmV0MTIzNA==”.

A successful response to this message will result in a 200 HTTP response status code (or 400/401 in the case of an error) and the **access token** is returned in a JSON structure in the body of the response.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{
"access_token":"zzz...yyy",
"token_type":"Bearer",
"expires_in":7200,
"refresh_token":"123...789"
}
```

The application parses the JSON for the **access token**. NOTE: The **refresh token** is sent only if the AS is configured to include it for the Client.

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

4.3.2. Implicit Grant Type

The implicit grant type uses the following parameters:

Parameter	Description	Mandatory
client_id	Client ID configured in the SSO Service *(case-sensitive)*	Yes
response_type	`token` *(case-sensitive)*	Yes
redirect_uri	URI where access token will be sent	Yes*
scope	The scope of access requested. If multiple scopes are requested, it is space-delimited. *(case-sensitive)*	No
state	An opaque value used by the Client to maintain state between the request and callback. If included, the AS returns this parameter and the given value when redirecting the user agent back to the Client.	No

* The SSO service can support multiple redirect URIs for a Client, although each SSO Service transaction will use only one of them. Even though this is *not always* required with an authorization code grant type, it is required for the implicit grant type.

The OAuth endpoint used is:

Type	Endpoint
Authorization	https://<base_FQDN>/as/authorization.oauth2

As an example, consider the following Client information:

Parameter	Example Value
client_id	`Client_5678`
response_type	`token`
redirect_uri	`sample://oauth2/code/cb`

Getting the Tokens

To initiate the process, the Client will redirect the user to the **authorization endpoint**. The redirect should contain the applicable URL-encoded attributes included in the query string of the URL. Using the above example parameters, the Client redirects the user to the following URL:

```
https://<base_FQDN>/as/authorization.oauth2?client_id=Client_5678&
response_type=token&redirect_uri=sample%3A%2F%2Foauth2%2Fimplicit%
2Fcb
```

This initiates an authentication process using the browser (i.e., user agent). Once the user has authenticated and approved the authorization request, the browser is redirected to the **redirect_uri** (or as configured for the Client) with the **access token** included as a *fragment* of the URL. A **refresh token** will NOT be returned to the Client.

```
sample://oauth2/implicit/cb#access_token=zzz...yyy&token_type=bear
er&expires_in=7200
```

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

Mobile

As mentioned earlier, a mobile application may not have a web page which can be used as the redirect. In this case, the **redirect_uri** may be a pre-defined URL provided by the SSO Service. This URL/page will not return any body content, but it will return a 200 HTTP response status code (or 400/401 in the case of an error), which is required in a mobile scenario.

```
https://<base_FQDN>/admin/OauthResponse.jsp
```

If the user cancels the login box, the browser is redirected to a different, preconfigured URL:

```
https://<base_FQDN>/admin/OauthResponse.jsp?logindenied
```

The mobile application reads the **logindenied** parameter and decides how to proceed.

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

4.3.3. Client Credential Grant Type

The Client Credential grant type will use the following parameters:

Parameter	Description	Mandatory
client_id	Client ID configured in the SSO Service *(case-sensitive)*	Yes
client_secret	Client Secret configured in the SSO Service *(case-sensitive)*	Yes
grant_type	`client_credentials` *(case-sensitive)*	Yes

The OAuth endpoint used is:

Type	Endpoint
Token	https://<base_FQDN>/as/token.oauth2

As an example, consider the following Client information:

Parameter	Example Value
client_id	`Client_9876`
client_secret	`appsecret9876`
grant_type	`client_credentials`

Getting the Token

The Client makes a request (HTTP POST) to the **token endpoint** with the Client credentials presented as HTTP Basic authentication:

```
POST https://<base_FQDN>/as/token.oauth2 HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic Q2xpZW50Xzk4NzY6YXBwc2VjcmV0OTg3Ng==
grant_type=client_credentials
```

If the request is successful, a 200 HTTP response status code is returned (or 400/401 in the case of an error), and the **access token** will be returned in a JSON structure. A **refresh token** will NOT be returned to the Client.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8{
"access_token":"zzz…yyy",
"token_type":"Bearer",
"expires_in":7200,
}
```

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

4.3.4. Refresh Tokens

An **access token** has a relatively short lifespan – two hours by default – that is relative to the time it is issued. The token may be used numerous times during that period, but it is considered invalid after its lifespan has expired. With the authorization code grant type, a Client can optionally be configured to receive a **refresh token** along with the original access token.

The Client can present the **refresh token** to the AS and receive a new access token. The new access token would have the same lifespan as the original access token. The refresh token can only be used once. When the **refresh token** is exchanged for a new **access token**, generally, a new **refresh token** is also returned. This exchange may occur numerous times – using each, new **refresh token** only once – up to a maximum time period of two days by default.

Consider the following example:

- 1:00 – access token 1 and refresh token 1 issued
- 1:15 – access token 1 used
- 1:30 – access token 1 used
- 3:00 – access token invalid (expired). Client exchanges refresh token 1 for a new access token 2 and refresh token 2.
- 3:30 – access token 2 used
- 5:00 – access token 2 invalid (expired). Client exchanges refresh token 2 for a new access token 3 and refresh token 3.
- *<two days pass – from the original time of 1:00 of the first step>* - both access token 3 and refresh token 3 expire – neither can be used.
- Client must start the authorization code process from the beginning.

Exchanging the refresh token will require the following parameters from the Client:

Parameter	Description	Mandatory
client_id	Client ID configured in the SSO Service *(case-sensitive)*	Yes
client_secret	Client Secret configured in the SSO Service *(case-sensitive)*	Yes
grant_type	`refresh_token` *(case-sensitive)*	Yes
refresh_token	The refresh token issued to the client during a previous access-token request.	Yes
scope	The scope of access requested. The requested scope must be equal to or less than the scope originally granted by the Resource Owner. If omitted, the scope is treated as equal to that originally granted by the Resource Owner.	No

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

The OAuth endpoint used is:

Type	Endpoint
Token	https://<base_FQDN>/as/token.oauth2

As an example, consider the following Client information:

Parameter	Example Value
client_id	`Client_9876`
client_secret	`appsecret9876`
grant_type	`refresh_token`
refresh_token	*<the currently-valid refresh token>*

Getting the Token

The Client makes a request (HTTP POST) to the **token endpoint** with the Client credentials presented as HTTP Basic authentication:

```
POST https://<base_FQDN>/as/token.oauth2 HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic Q2xpZW50Xzk4NzY6YXBwc2VjcmV0OTg3Ng==
grant_type=refresh_token&refresh_token=123...789
```

If the request is successful, a 200 HTTP response status code is returned (or 400/401 in the case of an error), and the **access token** will be returned in a JSON structure. A **refresh token** MAY be returned to the Client, if applicable.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{
"access_token":"zzz...yyy",
"token_type":"Bearer",
"expires_in":7200,
"refresh_token":"123...789"
}
```

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

4.3.5. OpenID Connect

OpenID Connect is a layer on top of OAuth. It gives a Client the ability to get the user's identity and other user-related attributes within an **ID token**. In essence, it is used for *authentication* and can be used instead of SAML or OpenToken. An **ID token** is not for *authorization* as that is the purpose for the **access token**.

There are three actions a Client needs to execute in order to implement OpenID Connect:

- **Get an OpenID Connect ID Token**
  Any grant type can leverage OpenID Connect by adding one or more OpenID-related **scope** values when requesting the access token.
- **Validate the ID Token**
  Validate the **ID token** to ensure that it originated from a trusted issuer (i.e., the AS) and that the contents have not been tampered with during transit.
- **Retrieve profile information**
  Using the **access token**, access the UserInfo endpoint to retrieve profile information about the authenticated user.

The **ID token** is a token used to identify an end-user to the Client application and to provide data around the context of that authentication. It is constructed using the JSON Web Token (JWT) format and is signed according to JSON Web Signing (JWS) specifications.

Refer to the OpenID Connect specifications for more information. The specifications also include guidelines for validating an **ID token** (Core specification section 3.1.3.7).

http://openid.net/specs/openid-connect-core-1_0.html#IDTokenValidation

The **ID token's** signature has to be validated using the **client secret** as a symmetric key and the HMAC with SHA-256 hash algorithm. Although there are other signing methods such as using an RSA (asymmetric) key, the SSO Service only supports the symmetric method above at this time.

4.3.5.1. OpenID Connect Scopes

There are three scopes related to OpenID.

- **openid** – In order to obtain an **ID token**, the Client *must* include this scope.
- **email** – This is optional and used to obtain the user's email address.
- **profile** – This is optional and used to obtain the user's first name and last name.

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

4.3.5.2. Sample User Info Endpoint Request

The OAuth endpoint used is:

Type	Endpoint
User Info	https://<base_FQDN>/idp/userinfo.openid

Once the Client has an access token, it can make a request to the **user info endpoint** to retrieve the requested attributes about a user. The request will include the **access token** presented using a method described in RFC6750 – one that includes the **access token** as a Bearer authorization credential.

An example HTTP client request to the **user info endpoint**:

```
GET https://<base_FQDN>/idp/userinfo.openid HTTP/1.1
Authorization: Bearer AAA...ZZZ
```

A successful response will return a HTTP 200 response status code (or 400/401 in the case of an error) and the user's claims in JSON format:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{
"sub":"E875834",
"family_name":"Pavlich",
"given_name":"Matthew",
"email":"abc@honeywell.com"
}
```

The "**sub**" is the user ID (i.e., Resource Owner identity). The above example includes three other values. If the "**email**" scope was requested, the JSON would include the "**email**" value (as it is in the example above). If the "**profile**" scope was requested (as it is in the example above), the JSON would include both the "given_name" and "family_name" values which correspond to the user's first name and last name respectively.

Before the values returned from the **user info endpoint** can be trusted, the Client must verify that the "**sub**" claim returned from the **user info endpoint** request matches the subject from the **ID token** (i.e., to guard against a token substitution attack).

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

4.4. Resource Server Implementation

4.4.1. Get Access Token

An **access token** can be used as an authorization token to configured resources (e.g., web services). To do this, the **access token** is be passed to the Resource Server. The most common approach is to use the HTTP Authorization header and include the access token as a Bearer authorization credential, however RFC 6750 also defines mechanisms for presenting an access token via query string and in a POST body.

For example, to enact a GET request on a REST web service, given an **access token** "AAA...ZZZ", the Client makes the following HTTP request:

```
GET https://api.company.com/user HTTP/1.1
Authorization: Bearer AAA...ZZZ
```

This will provide the **access token** to the Resource Server, which can validate the token, validate the client, verify the scope, identity the Resource Owner, and perform the appropriate action, if authorized.

4.4.2. Validate Access Token

For an API developer to integrate with OAuth 2.0, the Resource Server must accept and validate the **access token**. Once the **access token** has been received, the Resource Server can validate it with the AS. The response from the access token validation will include attributes that the Resource Server can use for authorization decisions.

Validating an **access token** will require the following parameters from the Client:

Parameter	Description	Mandatory
client_id	Client ID configured in the SSO Service *(case-sensitive)*	Yes
client_secret	Client Secret configured in the SSO Service *(case-sensitive)*	Yes
grant_type	`urn:pingidentity.com:oauth2:grant_type:validate_bearer` *(case-sensitive)*	Yes

As an example, consider the following Client information:

Parameter	Example Value
client_id	`Client_5678`
client_secret	`appsecret5678`
grant_type	`urn:pingidentity.com:oauth2:grant_type:validate_bearer`

The OAuth endpoint used is:

Type	Endpoint
Token	https://<base_FQDN>/as/token.oauth2

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

In order to fulfill the request, the RS first extracts the **access token** from the authorization header. It then queries the **token endpoint** to validate the token:

```
POST https://<base_FQDN/as/token.oauth2 HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic Q2xpZW50XzU2Nzg6YXBwc2VjcmV0NTY3OA==

grant_type=urn:pingidentity.com:oauth2:grant_type:validate_bearer&
token=AAA...ZZZ
```

A successful response to this message will result in a 200 HTTP response status code (or 400/401 in the case of an error) and a JSON structure in the body of the response similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{
"access_token":
  {
    "UserName":"jsmith"
  },
"token_type":"Bearer",
"expires_in":14400,
"scope":"https://example.honeywell.com/path/service",
"client_id":"client_5678"
}
```

The Resource Server extracts the Resource Owner identity from the **access token**; it can use this information to make a fine-grained authorization decision to allow or deny the web request.

NOTE: The scope is only included if one (or more) was requested by the Client and assigned when the **access token** was successfully returned to the Client.

4.4.3. Validate Client

The Resource Server (RS) can optionally authorize a request based on the Client that is making the request. It may, for example, allow access to Client X but not to Client Y. This type of authorization is not mandatory. The RS could simply rely on the AS to authorize Clients for the scope(s) specifically related to the RS. As mentioned earlier, the RS owner will provide a one-time approval when a Client is configured and wants to be able to request the scope(s) related to the RS. Assuming this is done, the RS doesn't have to check the Client identity for each request.

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

4.4.4. Verify Scope

An **access token** will contain one or more scopes – depending on what the Client is authorized to request and what was actually requested. No matter what, even if there were no scopes requested, the **access token** is considered as having the "default scope" – that is, a scope with no name and associated with no RS. In almost all cases, the RS must verify that the **access token** has the appropriate scope that is associated with the RS service being accessed. For example, if the RS is associated with a scope named "`https://example.honeywell.com/path/service`", then it should ONLY allow access to its services if the **access token** contains this scope. If the RS doesn't enforce this, then a security issue exists because any Client, with an **access token** with any scope (or none!) would be able to access the RS services.

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.

5. Appendix: Glossary

Acronym / Term	Description
Access Token	Allows a Client to claim authorization for accessing particular resources from a Resource Server.
Authorization Server (AS)	Manages the interactions within the OAuth framework. The Authorization Server provides access tokens to the Client and validates these when asked by the Resource Server.
Client	The application that is requesting access to resources.
FQDN	Fully-qualified domain name. For example, *widget.honeywell.com*.
Refresh Token	Allows a Client to obtain a new access token without re-obtaining authorization from the Resource Owner.
Resource Owner	The application user.
Resource Server (RS)	The application or API that provides resources.
Representational State Transfer (REST)	A stateless architecture that runs over HTTP and generally applies to web services.
Security Assertion Markup Language (SAML)	An XML-based framework for communicating user authentication, entitlement, and attribute information.
Scope	A named permission to access services of a given Resource Server.
Single Page Application (SPA)	A web application that consists of only a single web page. As the user interacts with the application, specific contents on the page are changed without the user being directing to a new web page.
User Agent	The user’s browser.

THIS DOCUMENT IS PUBLISHED AS PART OF AN ELECTRONIC DOCUMENT REPOSITORY.
THE USER IS RESPONSIBLE FOR REFERENCING THE MOST RECENTLY PUBLISHED ELECTRONIC VERSION.