# Welcome to SimpleIdServer

SimpleIdServer is an open source framework enabling the support of OPENID, OAUTH2.0 and SCIM2.0.

It streamlines development, configuration and deployment of custom access control servers.

Thanks to its modularity and extensibility, SimpleIdServer can be customized to the specific needs of your organization for authentication, authorization and more.

Github: https://github.com/simpleidserver/SimpleIdServer

Nuget feed: https://www.myget.org/F/advance-ict/api/v3/index.json

# Getting Started Guide

## Getting started with OAUTH2.0 server

An OAUTH2.0 server can be deployed into an ASP.NET CORE project by installing the Nuget package

“SimpleIdServer.OAuth”.

As a developer you can follow the steps below to deploy a local version:

1. Create an empty ASP.NET CORE project.
2. Install the Nuget package “SimpleIdServer.OAuth”.
3. Open the Startup.cs file and register the dependencies by inserting the instruction “services.AddSIDOAuth()”.
4. Always in the “Startup.cs” file configure the application builder by inserting the instruction “services.UseSID()”.
5. Launch the solution.
6. Check if a JSON structured response is returned by browsing the relative URL “/.well-known/oauth-authorization-server”.

By default OAUTH2.0 clients, scopes and JSON Web keys are stored in memory. Their values can be overridden by using the operations exposed by the “SimpleIdServerOAuthBuilder” instance which is returned by the instruction “services.AddSIDOAuth()”.

For example: default OAUTH2.0 clients can be overridden by calling the instruction : “services.AddSIDOAuth().AddClients();”

OAUTH2.0 settings can also be overridden by manipulating the option “OAuthHostOptions”. For more information about OAuthHostOptions please refer to the reference.

## Getting started with OPENID server

An OPENID server with bootstrap4 theme can be deployed into an ASP.NET CORE project by installing the Nuget package “SimpleIdServer.OpenID.Bootstrap4”. There is one Nuget package per UI theme and at the moment only the Bootstrap4 library is supported.

As a developer you can follow the steps below to deploy a local version :

1. Create an empty ASP.NET CORE project.
2. Install the Nuget package “SimpleIdServer.OpenID.Bootstrap4”.
3. Launch the solution.
4. Check if a JSON structured response is returned by browsing the relative URL “/.well-known/openid-configuration”.

By default there is no authentication method and they should be installed separately by the developers. For example : login password authentication with Bootstrap4 theme can be installed with the Nuget “SimpleIdServer.UI.Authenticate.LoginPassword.Bootstrap4”. The following steps describe the procedure :

1. Install the Nuget package “SimpleIdServer.UI.Authenticate.LoginPassword.Bootstrap4”.
2. Open the “Startup.cs” file and import the namespace “SimpleIdServer.OpenID”.
3. Register the dependencies by calling the method “AddLoginPasswordAuthentication” which comes from the “SimpleIdServerOpenIDBuilder” instance returned by “services.AddSIDOpenID()”.
4. Launch the solution.
5. Browse the relative URL “*authorization?client\_id=f3d35cce-de69-45bf-958c-4a8796f8ed37&redirect\_uri=http://localhost:8080&response\_type=code&scope=openid profile&state=state*”. An authentication screen with bootstrap4 theme must be displayed.
6. Authenticate with the credentials : Login : administrator, Password : password
7. Confirm the consent..
8. User agent will be redirected to the callback url <http://localhost:8080> with the authorization code passed into the query.

Like in OAUTH2.0 server, the clients, scopes, users, JSON Web keys are stored in memory. OPENID introduces the concept of “authentication context class reference” (ACR), the default values can be overridden by calling the operation “AddAcrs” exposed by the “SimpleIdServerOpenIdBuilder” instance.

OPENID settings can also be overridden by manipulating the option “OpenIDHostOptions”. For more information about OpenIDHostOptions please refer to the reference.

## Getting started with SCIM2.0

A system for Cross-domain Identity Management (SCIM2.0) can be deployed into an ASP.NET CORE project by installing the Nuget package “*SimpleIdServer.Scim*”.

As a developer you can follow the steps below to create a local version :

1. Create an empty ASP.NET CORE project.
2. Install the Nuget package “SimpleIdServer.Scim”.
3. Open the Startup.cs file and register the dependencies by inserting the instruction “services.AddSIDScim()”.
4. Configure the authentication process by using the function “AddAuthentication” returned by “AddSIDScim()”. All types of authentication package can be used for example “Microsoft.AspNetCore.Authentication.JwtBearer”, it should be registered with the authentication scheme “SCIMConstants.AuthenticationScheme”. The following statement configures cookie authentication:

*services.AddSIDScim().AddAuthentication(s => s.AddCookie(SCIMConstants.AuthenticationScheme));*

1. Always in the “Startup.cs” file configure the application builder by inserting the instruction “services.UseSID()”.
2. Launch the solution.
3. Check if a JSON structured response is returned by browsing the relative URL “/Schemas”.

By default the SCIM schemas are stored in memory. They can be overridden by using the instruction “AddSchemas” returned by the function “AddSIDScim()” like this :

*Services.AddSIDScim().AddSchemas();*

A helper “SCIMSchemaBuilder ()” is exposed by the package to facilitate the creation of SCIM schema for example :

|  |  |
| --- | --- |
| {{  "id": "urn:dog",  "attributes": [  {  "name": "name",  "type": "string",  "mutability": "readwrite"  },  {  "name": "birthdate",  "type": "datetime",  "mutability": "readwrite",  },  {  "name": "attributes",  "type": "complex",  "multiValued": true,  "mutability": "readwrite",  "subAttributes": [  {  "name": "name",  "type": "string",  "mutability": "readwrite"  },  {  "name": "value",  "type": "string",  "mutability": "readwrite"  }  ]  }  ]  }} | var schema = SCIMSchemaBuilder.Create("urn:dog")  .AddStringAttribute("name")  .AddDateTimeAttribute("birthdate")  .AddComplexAttribute("attributes", callback: c =>  {  c.AddStringAttribute("name");  c.AddStringAttribute("value");  }, multiValued: true)  .Build(); |

SCIM settings can also be overridden by manipulating the properties of the option “SCIMHostOptions”. For more information about “SCIMHostOptions” please refer to the reference.

# Topics

## Build JSON Web Key

The “SimpleIdServer.Jwt” library offers helpers to help the developers to build JSON Web Keys (JWKS). There are two types of JWKS.

The first type is used during the claims transformation phase from clear JSON format into JSON Web Signature (JWS) token. It is generally used by the authorization server and by the clients during the creation of JWS tokens and the verification of their signatures:

* Authorization server uses the JWK to build JWS tokens and return them to the clients.
* The clients check the signature by using the JSON Web Keys exposed by the relative url “jwks”.

This kind of JSON Web Key must be stored by the authorization server and only the public key must be exposed to external clients, the private key must be kept secret.

A JSON Web Key used to create JWS token can be built like this:

using (var rsa = RSA.Create())

{

var sigJsonWebKey = new JsonWebKeyBuilder().**NewSign**("1", new[]

{

KeyOperations.Sign,

KeyOperations.Verify

}).**SetAlg**(rsa, "RS256").Build();

}

The operation “NewSign” accepts two parameters. The first accepts a string and represents a unique identifier, the second accepts an array of key operations in our example it should be set to “Sign” and “Verify”.

The algorithm used during the generation of the signature can be set by calling the operation “SetAlg(rsa, “name”) and it contains two parameters. The first accepts either an RSA or EllipticCurve key, the second is used to set the algorithm name. Supported algorithms are listed below :

|  |  |
| --- | --- |
| Key | Algorithm name |
| ECDsaCng | ES256 |
| ECDsaCng | ES384 |
| ECDsaCng | ES512 |
| HMACSHA256 | HS256 |
| HMACSHA384 | HS384 |
| HMACSHA512 | HS512 |
| RSA | RS256 |
| RSA | RS384 |
| RSA | RS512 |

The second JSON Web Key type is used during the transformation phase from JSON Web Signature (JWS) token into JSON Web Encryption (JWE) token. In most of the case this type is used by the authorization server during the creation of the token and by the client during the decryption phase:

* Authorization server uses the public key exposed by the client to encrypt the data.
* The client use his private key to decrypt the content and extract the JSON Web Signature.

This kind of JSON Web key must be stored by the client and only the public key must be exposed to external authorization server, the private key must be kept secret.

A JSON Web Key used to create JWE token can be built like this:

using (var rsa = RSA.Create())

{

var sigJsonWebKey = new JsonWebKeyBuilder().**NewEnc**("1", new[]

{

KeyOperations.Encrypt,

KeyOperations.Decrypt

}).**SetAlg**(rsa, "RSA-OAEP").Build();

}

The supported algorithms are listed below :

|  |  |
| --- | --- |
| Key | Algorithm name |
| RSA | RSA-OAEP |
| RSA | RSA1\_5 |
| RSA | RSA-OAEP-256 |

# Reference

## OAuthHostOptions

The SimpleIdServer OAUTH2.0 framework takes its configuration from the “OAuthHostOptions” option class. The properties can be changed in the Startup class like this:

**public void ConfigureServices(IServiceCollection services)**

**{**

**services.AddSIDOAuth(o =>**

**{**

**o.ClientSecretExpirationInSeconds = 2;**

**})**

**}**

The OAuthHostOptions class contains the following properties:

**DefaultScopes** : Default scopes assigned to a OAUTH2.0 client during the registration process.

**ClientSecretExpirationInSeconds** : Default expiration time in seconds of a client secret. By default no expiration time is set.

**DefaultTokenProfile** : Default token profile assigned to a OAUTH2.0 client during the registration process. The possible values are “mac” or “bearer”.

**DefaultOAuthScopes** : Default OAUTH2.0 scopes. Refer to the chapter “OAuth2.0 scope’s properties” for more information about the scope’s properties.

**SoftwareStatementTrustedParties** : If a “software\_statement” parameter is passed to the client registration request then the property “SoftwareStatementTrustedParties” is used by the authorization process to check the parameter. The validation process is made of two steps:

* Fetch the “iss” parameter from the JWS header and get the corresponding JWKS URL from “SoftwareStatementTrustedParties”.
* Fetch the JSON Web Key from the URL and check the signature of the “software\_statement” parameter.

**DefaultCulture** : Default culture used by the UI if there is no “ui\_locales” parameter specified in the authorization request.

## OAuthClient

**ClientId** : client identifier.

**Secrets** : one or more client secrets.

**ClientNames** : one or more human readable client name.

**LogoUris** : one or more URL that references a logo for the client.

**ClientUris** : one or more URL of a web page providing information about the client.

**PolicyUris** : one or more URL that points to a human-readable policy document for the client.

**TosUris** : one or more URL that points to a human-readable terms of service document for the client.

**TokenSignedResponseAlg** : Cryptographic algorithm used to secure the JWS access token.

**TokenEncryptedResponseAlg** :Cryptographic algorithm used to encrypt the JWS access token.

**TokenEncryptedResponseEnc** : Content encryption algorithm used perform authenticated encryption on the JWS access token.

**TokenEndpointAuthMethod** : Requested authentication method for the token endpoint. The possible values are :

* “client\_secret\_post”
* “client\_secret\_basic”
* “private\_key\_jwt”
* “client\_secret\_jwt”

**GrantTypes** : Array of OAUTH2.0 grant type strings that the client can use at the token endpoint. The possible values are :

* “authorization\_code”
* “implicit”
* “password”
* “client\_credentials”
* “refresh\_token”

**ResponseTypes** : Array of the OAUTH2.0 response type strings that the client can use at the authorization endpoint. The possible values are :

* “code”
* “token”

**AllowedScopes** : Scope values that the client can use when requesting access tokens.

**RedirectionUrls** : Array of redirection URIS for use in redirect-based flows.

**JwksUri** : URI string referencing the client’s JSON Web Key (JWK) Set document, which contains the client’s public keys.

**JsonWebKeys** : Client’s JSON Web Key Set document value, which contains the client’s public keys.

**TokenExpirationTimeInSeconds** : Token expiration time in seconds.

**RefreshTokenExpirationTimeInSeconds** : Refresh token expiration time in seconds.

**PreferredTokenProfile** : preferred token profile, possible values are : “bearer” or “mac”.

**Contacts** : Array of strings representing was to contact people responsible for the client, typically email addresses.

**SoftwareId** : A unique identifier assigned by the client developer or software publisher used by registration endpoints to identify the client software to be dynamically registered.

**SoftwareVersion** : A version identifier string for the client software identified by “software\_id”.

## OAuthScope

**Name** : name of the OAUTH2.0 scope.

**IsExposedInConfigurationEdp** : If the value is true then the scope is returned by the configuration endpoint “*.well-known/oauth-authorization-server*”.

## OAuthUser

**Id** :identifier of the user.

**Claims** : list of information that identify an end user.

**Consents** : list of user’s consents.

**Credentials** : list of user’s credentials.

## OpenIDHostOptions

OPENID options inherits the properties from OAUTH2.0 options and specify new ones :

**AuthenticationScheme** : Default authentication scheme. The default value is set to “.AspNetCore.MultiAccount”.

**CookieName** : Default cookie name. The default value is set to “MultiAccount”.

**DefaultAcrValue** : Default Authentication Context Class Reference (ACR) used by the authorization server if cannot be deduced. The default value is set to “sid-load-01”.

**DefaultMaxAge** : Default max age assigned to an OPENID client during the registration process. No default value.

**DefaultSubjectType** : Default subject type assigned to an OPENID client during the registration process. The possible values are : “pairwise” or “public” and the default value is set to “public”.

## OpenIDClient

OPENID client inherits the properties from OAUTH2.0 client and defines new ones :

**ApplicationType** : Kind of the application. The default, if omitted, is web. The defined values are “native” or “web”.

**IdTokenSignedResponseAlg** : Cryptographic algorithm used to secure the JWS identity token.

**IdTokenEncryptedResponseAlg** : Cryptographic algorithm used to encrypt the JWS identity token.

**IdTokenEncryptedResponseEnc** : Content encryption algorithm used perform authenticated encryption on the JWS identity token.

**UserInfoSignedResponseAlg** : Required for signing UserInfo responses.

**UserInfoEncryptedResponseAlg** : Required for encrypting the identity token issued to this client.

**UserInfoEncryptedResponseEnc** : Required for encrypting the identity token issued to this client.

**RequestObjectSigningAlg** : Must be used for signing Request Objects sent to the OpenID provider.

**RequestObjectEncryptionAlg** : JWE alg algorithm the relying party is declaring that it may use for encrypting Request Objects sent to the OpenID provider.

**RequestObjectEncryptionEnc** : JWE enc algorithm the relying party is declaring that it may use for encrypting request objects sent to the OpenID provider.

**SubjectType** : subject\_type requested for responses to this client. Possible values are “pairwise” or “public”.

**DefaultMaxAge** :Default Maximum Authentication Age.

**DefaultAcrValues** : Default requested Authentication Context Class Reference values.

**RequireAuthTime** : Boolean value specifying whether the auth\_time claim in the identity token is required.

**SectorIdentifierUri** : URI using the HTTPS scheme to be used in calculating Pseudonymous Identifiers by the OpenID provider.

**PairWiseIdentifierSalt** : SALT used to calculate the pairwise.

## OpenIDScope

OPENID scope inherits the properties from OAUTH2.0 scope and defines new ones :

**Claims** : Array of strings that specifies the claims.

## AuthenticationContextClassReference

**Name** : Name of the Authentication Context Reference.

**DisplayName** : Human-readable Authentication Context Reference.

**AuthenticationMethodReferences** : Array of strings that specifies the authentication methods.

## SCIMHostOptions

**AuthenticationScheme** : Default authentication scheme.

**GroupSchemaIds**: List of schema identifiers used to represent a Group.

**UserSchemaIds** : List of schema identifiers used to represent a User.

**MaxOperations** : An integer value specifying the maximum number of operations.

**MaxPayloadSize** : An integer value specifying the maximum payload size in bytes.

**MaxResults** :An integer value specifying the maximum number of resources returned in a response.

TODOLIST :

* Ajouter l’authentification de type “tls\_client\_auth” (<https://tools.ietf.org/html/draft-ietf-oauth-mtls-08#section-2.1>)
* Ajouter PKCE (<https://tools.ietf.org/html/rfc7636>).
* Ajouter session management (<https://openid.net/specs/openid-connect-session-1_0.html>).