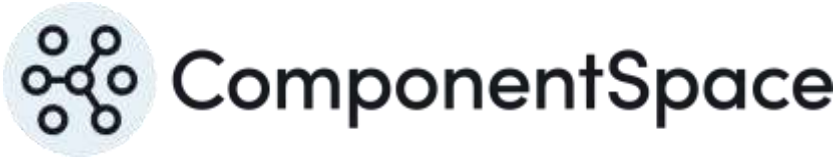


ComponentSpace

OpenID Connect for ASP.NET Core

Configuration Guide

www.componentspace.com

Contents

Introduction

The ComponentSpace OpenID Connect for ASP.NET Core is a .NET class library that provides OpenID provider functionality.

The primary OpenID Connect APIs, defined by the IOpenIDProvider interface, make use of provider, client, and X.509 certificate configuration. For more information about these APIs, refer to the OpenID Connect for ASP.NET Core Developer Guide.

OpenID Connect configuration may be specified either as JSON or programmatically.

Configuration Options

There are several options for specifying configuration.

1. JSON either in the application's appsettings.json or in a separate JSON file
2. Programmatically through the configuration API
3. Programmatically by implementing IConfigurationResolver

Using a JSON configuration file is the simplest approach and requires no additional coding.

If configuration information is stored in a database, it must be set programmatically.

If the configuration changes infrequently, it may be set using the configuration API, typically at application start-up.

If the configuration changes frequently, it's better to implement the IConfigurationResolver interface for the on-demand retrieval of configuration information.

Configuration JSON

Configuration may be specified as JSON either in the application's appsettings.json or in a separate JSON file (eg openid-config.json).

The following example defines an object named OpenIDProvider whose value, for brevity, is the outline of a configuration.

```
{
  "OpenIDProvider": {
    "Configurations": [
    ]
  }
}
```

The configuration delegate is registered in the application's Program class.

```
builder.Services.AddOpenIDProvider(builder.Configuration.GetSection("OpenIDProvider"));
```

If a separate JSON file is used, it must be added to the configuration builder in the application's Program class.

```
builder.Configuration.AddJsonFile("openid-config.json");
```

Example Configuration

The following is an example of setting the configuration through JSON.

```
"OpenIDProvider": {
  "$schema": "https://www.componentspace.com/schemas/openid-config-schema-v1.0.json",
  "Configurations": [
    {
      "ProviderConfiguration": {
        "ProviderMetadata": {
          "Issuer": "https://ExampleOpenIDProvider",
          "AuthorizationEndpoint": "https://localhost:44311/openid/authorize",
          "TokenEndpoint": "https://localhost:44311/openid/token",
          "UserinfoEndpoint": "https://localhost:44311/openid/userinfo",
          "JwksUri": "https://localhost:44311/openid/keys",
          "EndSessionEndpoint": "https://localhost:44311/openid/logout",
          "ScopesSupported": [ "openid" ],
          "ResponseTypesSupported": [ "code", "id_token", "id_token token", "code id_token", "code
token", "code id_token token" ],
          "ResponseModesSupported": [ "query", "fragment", "form_post" ],
          "GrantTypesSupported": [ "authorization_code", "implicit", "refresh_token",
"client_credentials" ],
          "SubjectTypesSupported": [ "public" ],
          "IdTokenSigningAlgValuesSupported": [ "HS256", "HS384", "HS512", "RS256", "RS384",
"RS512", "ES256", "ES384", "ES512", "PS256", "PS384", "PS512" ],
          "IdTokenEncryptionAlgValuesSupported": [ "A128KW", "A192KW", "A256KW", "dir",
"RSA1_5", "RSA-OAEP" ],
          "IdTokenEncryptionEncValuesSupported": [ "A128CBC-HS256", "A192CBC-HS384",
"A256CBC-HS512" ],
          "UserinfoSigningAlgValuesSupported": [ "HS256", "HS384", "HS512", "RS256", "RS384",
"RS512", "ES256", "ES384", "ES512", "PS256", "PS384", "PS512" ],
          "UserinfoEncryptionAlgValuesSupported": [ "A128KW", "A192KW", "A256KW", "dir",
"RSA1_5", "RSA-OAEP" ],
          "UserinfoEncryptionEncValuesSupported": [ "A128CBC-HS256", "A192CBC-HS384",
"A256CBC-HS512" ],
          "RequestObjectSigningAlgValuesSupported": [ "HS256", "HS384", "HS512", "RS256",
"RS384", "RS512", "ES256", "ES384", "ES512", "PS256", "PS384", "PS512" ],
          "RequestObjectEncryptionAlgValuesSupported": [ "A128KW", "A192KW", "A256KW", "dir",
"RSA1_5", "RSA-OAEP" ],
          "RequestObjectEncryptionEncValuesSupported": [ "A128CBC-HS256", "A192CBC-HS384",
"A256CBC-HS512" ],
          "TokenEndpointAuthMethodsSupported": [ "client_secret_basic", "client_secret_post",
"client_secret_jwt", "private_key_jwt", "none" ],
          "TokenEndpointAuthSigningAlgValuesSupported": [ "HS256", "HS384", "HS512", "RS256",
"RS384", "RS512", "ES256", "ES384", "ES512", "PS256", "PS384", "PS512" ],
          "DisplayValuesSupported": [ "page", "popup", "touch", "wap" ],
          "ClaimSupported": [ "amr", "aud", "email", "exp", "family_name", "given_name", "iat",
"idp", "iss", "jti", "middle_name", "name", "nbf", "nonce", "preferred_username", "sub", "ver" ],
```

```
      "CodeChallengeMethodsSupported": [ "plain", "S256" ],
      "RequestParameterSupported": true,
      "RequestUriParameterSupported": true
     },
     "ProviderCertificates": [
      {
       "FileName": "certificates/op.pfx",
       "Password": "password"
      }
     ]
    },

    "ClientConfigurations": [
     {
      "Description": "Blazor WASM",
      "ClientID": "CFTapaLooboloAasQvjOYFPlf4Hjhmur",
      "ClientSecret": "wZvn0dy7PX8CkkbgqYllDhlfOpqD9xyr",
      "RedirectUris": [
       "https://localhost:44361/authentication/login-callback"
      ],
      "PostLogoutRedirectUris": [
       "https://localhost:44361/authentication/logout-callback"
      ]
     },
     {
      "Description": "Example OpenID Client",
      "ClientID": "wLpJpHADUqEmmAltrZX87yUMz8lgweWs",
      "ClientSecret": "P41HXh7SptRM6rV4xjgdVmUkXssibunr",
      "RedirectUris": [
       "https://localhost:44389/signin-oidc"
      ],
      "PostLogoutRedirectUris": [
       "https://localhost:44389/signout-callback-oidc"
      ],
      "ClientCertificates": [
       {
        "FileName": "certificates/client.cer"
       }
      ]
     }
    ]
   }
  ]
},
```

JSON Schema

A JSON schema file, openid-config-schema-v<version-number>.json, is included in the documentation folder (eg. openid-config-schema-v1.0.json).

The corresponding file is also available under https://www.componentspace.com/schemas (eg. https://www.componentspace.com/schemas/openid-config-schema-v1.0.json).

This may be used to enable Visual Studio Intellisense when editing OpenID configuration JSON or when using a JSON schema validator.

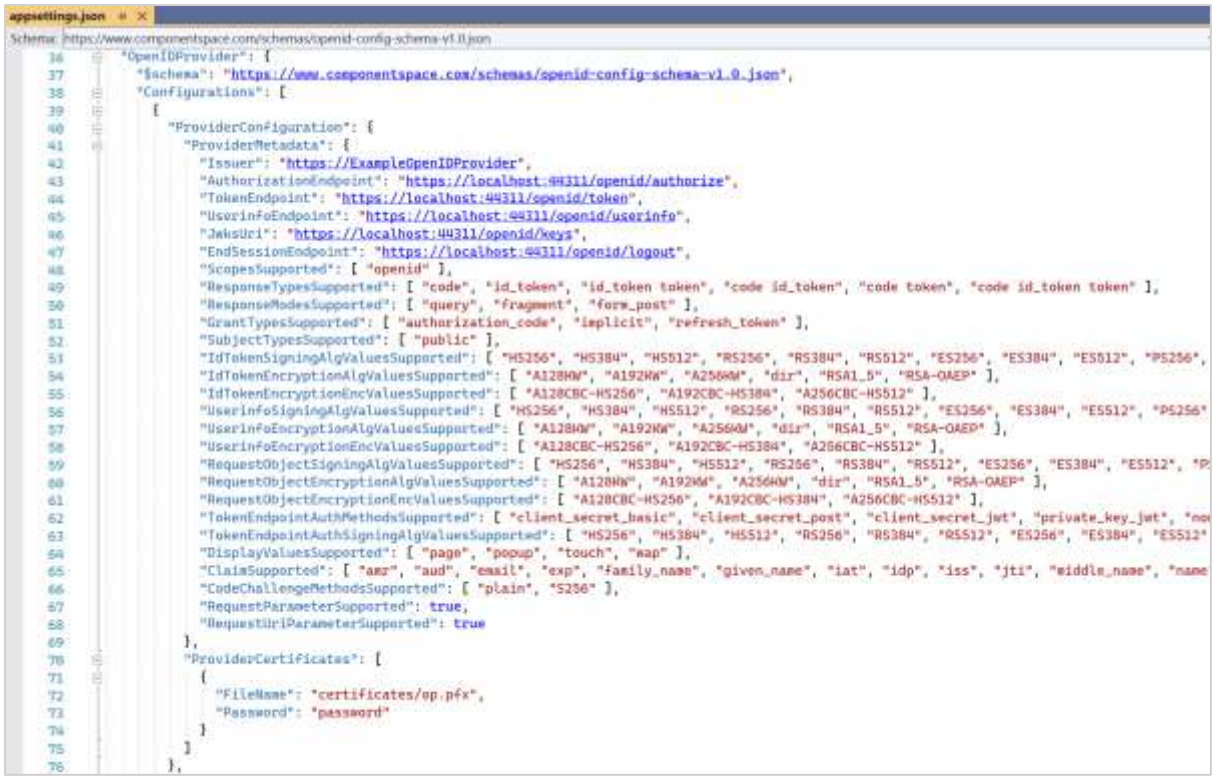
The following example specifies the schema associated with the OpenID configuration.

```
{
  "OpenIDProvider": {
    "$schema": "https://www.componentspace.com/schemas/openid-config-schema-v1.0.json",
    "Configurations": [
    ]
  }
}
```

Enabling Visual Studio Intellisense

By default, when the JSON configuration file is opened in Visual Studio, no schema is selected and therefore Intellisense is not enabled.

To enable Intellisense, copy the $schema value into the Schema combo box and click enter.



Intellisense now should be enabled.

Note that schema validation is not performed at runtime.

Programmatically Specifying Configuration

In many scenarios, storing the configuration as JSON is the simplest and preferred approach.

However, there may be instances where this isn't the case. For example, the configuration may be stored in a custom database. Once retrieved, it would be set using the configuration API.

The configuration delegate is registered in the application's Program class.

```
builder.Services.AddOpenIDProvider(config => ConfigureOpenID(config));
```

ConfigureOpenID is a delegate with the following method signature.

```
private void ConfigureOpenID(OpenIDConfigurations openIDConfigurations)
```

Example Configuration

For an example of setting the OpenID configuration programmatically, refer to the ExampleOpenIDProvider project's ConfigurationExamples class.

Implementing IConfigurationResolver

The IConfigurationResolver interface provides an alternative mechanism for specifying configuration. Rather than calling the configuration API to specify configuration, the IConfigurationResolver interface is implemented to return configuration as requested. This approach is recommended when supporting dynamic configurations.

The configuration resolver is registered in the application's Program class.

```
builder.Services.AddOpenIDProvider();
builder.Services.AddScoped<IConfigurationResolver, ConfigurationResolver>();
```

Example Configuration

For an example of implementing the IConfigurationResolver interface, refer to the ExampleOpenIDProvider project's ConfigurationExamples class.

Multi-Tenancy Support

Multi-tenancy support refers to a single application accommodating multiple customers or tenants each of whom has their own separate configuration.

For most use cases, a single configuration will suffice, and multi-tenancy support is not required.

As with a single configuration, multiple configurations may be specified through JSON, programmatically or via the IConfigurationResolver interface.

The following is an example outline of multiple configurations.

```
"OpenIDProvider": {
 "$schema": "https://www.componentspace.com/schemas/openid-config-schema-v1.0.json",
 "Configurations": [
  {
   "Name": "Tenant1",
   "ProviderConfiguration": {
   },
   "ClientConfigurations": [
   ]
  },
```

```
  {
   "Name": "Tenant2",
   "ProviderConfiguration": {
   },
   "ClientConfigurations": [
   ]
  },
  {
   "Name": "Tenant3",
   "ProviderConfiguration": {
   },
   "ClientConfigurations": [
   ]
  }
 ]
}
```

The Name property uniquely identifies each of the configurations.

Configuration Selection

Prior to processing OpenID requests, a configuration must be selected. This is done by setting the ConfigurationName property on the IOpenIDProvider interface. Refer to the OpenID Connect for ASP.NET Core Developer Guide for more information.

The following example specifies the configuration to use when processing the authentication request.

```
// Identify the tenant (application specific, details not shown).
var tenantName = GetTenantName();

// Specify the OpenID configuration.
_openIDProvider.ConfigurationName = tenantName;

// Receive and process the authentication request.
var authenticationRequest = await _ openIDProvider.ReceiveAuthnRequestAsync();
```

Identifying the Tenant

The application is responsible for identifying the tenant and therefore the name to specify when setting the ConfigurationName property.

Possible methods include:

- Separate subdomain names for each tenant
- Query string parameter
- Special HTTP headers or cookies
- IP address ranges

OpenIDConfigurations

The OpenIDConfigurations class is the top-level class specifying the OpenID configurations.

Configurations [required]

The Configurations is a list of one or more OpenIDConfiguration items. Each OpenIDConfiguration item corresponds to a tenant in a multi-tenancy application. In the more common single tenancy application, a single OpenIDConfiguration is defined.

OpenIDConfiguration

The OpenIDConfiguration class specifies a single OpenID configuration for an OpenID provider.

Name [optional]

Each OpenIDConfiguration is identified by a unique name. This name is internal to the configuration and is not exposed to clients.

A name is only required if there are multiple OpenID configurations.

ProviderConfiguration [required]

The ProviderConfiguration specifies the OpenID provider configuration.

ClientConfigurations [required]

The ClientConfigurations is the list of one or more ClientConfiguration items. Each ClientConfiguration specifies the configuration of an OpenID Connect client.

ProviderConfiguration

The ProviderConfiguration specifies the configuration for the OpenID provider.

Description [optional]

The description of the OpenID provider.

ProviderMetadata [required]

The OpenID provider discovery metadata.

ProviderCertificates [required]

The certificates used by the OpenID provider to secure tokens.

AuthCodeExpiry [optional]

The authorization code expiry.

The default is ten minutes.

AccessTokenExpiry [optional]

The access token expiry.

The default is ten minutes.

IdTokenExpiry [optional]

The ID token expiry.

The default is ten minutes.

ClockSkew [optional]

The permitted clock skew between the OpeID provider and clients.

The default is five minutes.

ProviderMetadata

The ProviderMetadata corresponds to the OpenID provider metadata defined in the OpenID Connect Discovery specification available at:

https://openid.net/specs/openid-connect-discovery-1_0.html.

Property names are in Pascal rather than the Snake case uses in the Discovery specification.

This means the property “Issuer” is included in the metadata returned to the client as “issuer”. Similarly, “AuthorizationEndpoint” is returned as “authorization_endpoint”.

ClientConfiguration

The ClientConfiguration specifies the configuration for a client of the OpenID provider.

Description [optional]

The description of the client.

ClientID [required]

The client ID uniquely identifies the client.

ClientSecret [optional]

The client secret.

RedirectUris [required]

The list of valid redirect URIs for authorization responses.

PostLogoutRedirectUris [optional]

The list of valid redirect URIs for logout responses.

ClientCertificates [required]

The certificates used by the OpenID provider to validate tokens sent by the client.

SignIDTokens [optional]

The flag specifying whether ID tokens should be signed.

The default is true.

EncryptIDTokens [optional]

The flag specifying whether ID tokens should be encrypted.

The default is false.

SignUserInfo [optional]

The flag specifying whether user info should be signed.

The default is true.

EncryptUserInfo [optional]

The flag specifying whether user info should be encrypted.

The default is false.

SignatureAlgorithm [optional]

The signature algorithm. For more information, refer to RFC 7518.

The default is RS256.

KeyManagementAlgorithm [optional]

The key management algorithm. For more information, refer to RFC 7518.

The default is RSA-OAEP.

EncryptionAlgorithm [optional]

The encryption algorithm. For more information, refer to RFC 7518.

The default is A128CBC-HS256.

RequireCodeChallenge [optional]

The flag specifying whether a Proof Key for Code Exchange (PKCE) code challenge is required.

The default is false.

Certificate

The certificate specifies the location and purpose of an X.509 certificate.

Certificates may be base-64 encoded strings, stored on the file system, within a Windows certificate store, or an Azure key vault.

For certificate strings, the base-64 encoded string and optional password must be specified.

For certificate files, the file name and optional password must be specified.

For certificates in a Windows certificate store, the store name and location may be specified along with one of the following: the certificate's serial number; thumb print; or subject name.

For certificates in an Azure key vault, the configuration key must be specified.

For more information, refer to the OpenID Connect for ASP.NET Core Certificate Guide.

Status [optional]

The certificate status may be:

- Retired

- Active
- Future

This assists with key rollover.

Only active certificates are used for signature generation or encryption.

Retired certificates are previously active certificates that are no longer in use.

Future certificates will become active at some future point in time.

All certificates are included when returning the provider's keys as part of discovery.

The default is Active.

Use [optional]

The certificate use may be:

- Encryption
- Signature
- Any

A certificate whose use is encryption may be used for encryption or decryption only.

A certificate whose use is signature may be used for signature generation or verification only.

A certificate whose use is any may be used for any purpose with no restrictions.

The default is encryption and signature.

String [optional]

The string is the certificate base-64 encoded string. This may contain the public key only or the public and private keys.

FileName [optional]

The file name is the relative or absolute path to the X.509 certificate file. This may be a CER file containing a public key only or a PFX file also containing a private key.

Password [optional]

The password protects the private key.

StoreName [optional]

For certificates in a Windows certificate store, the store name specifies the store.

The store name may be one of the standard stores:

- AddressBook
- AuthRoot
- CertificateAuthority
- Disallowed
- My
- Root
- TrustedPeople

- TrustedPublisher

Alternatively, it may be any other store including:

- WebHosting

The default is the My (i.e. personal) store.

StoreLocation [optional]

For certificates in a Windows certificate store, the store location specifies the location.

The store location may be:

- CurrentUser
- LocalMachine

The default is the local machine.

SerialNumber [optional]

For certificates in a Windows certificate store, the certificate is specified by its serial number.

Thumbprint [optional]

For certificates in a Windows certificate store, the certificate is specified by its thumb print.

SubjectName [optional]

For certificates in a Windows certificate store, the certificate is specified by its subject name.

Key [optional]

For certificates in an Azure key vault, the certificate is specified by its configuration key.

URL

OpenID Provider metadata URLs may be absolute or relative.

URLs are relative to the host name and port number of the current HTTP request.

For example, an authorization endpoint URL may be specified absolutely.

https://localhost:44311/openid/authorize

Alternatively, it may be specified as a path.

/openid/authorize

This is converted to an absolute URL using the base URL of the current HTTP request.