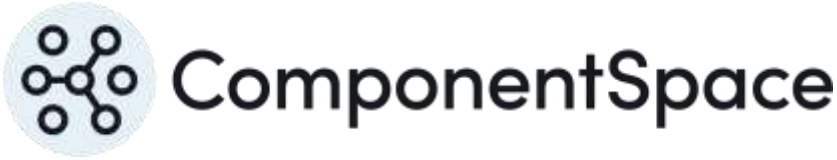

ComponentSpace

OpenID Connect for ASP.NET Core

Developer Guide

www.componentspace.com

Contents

Introduction

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

You can use this functionality to easily enable your ASP.NET Core web applications to participate in OpenID Connect flows as the OpenID provider. Example applications with full source code are included.

For a walkthrough of the example projects, refer to the OpenID Connect for ASP.NET Core Examples Guide.

Visual Studio and .NET Core Support

The ComponentSpace OpenID Connect for ASP.NET Core .NET class library is compatible with the following frameworks:

- .NET 6.0 and above

The class library may be used with the following project types:

- ASP.NET Core Web Application (.NET Core)

The OpenID Connect for ASP.NET Core examples include solution and project files for:

- Visual Studio 2022 and above

OpenID API

Application Startup

In the application's Program class, the following occurs:

The OpenID Connect services are added.

This includes registering the OpenID configuration. OpenID configuration may be specified as JSON either in the application's appsettings.json or in a separate JSON file (eg oidc.json).

```
// Add OpenID provider services.
builder.Services.AddOpenIDProvider(builder.Configuration.GetSection("OpenIDProvider")
```

Alternatively, OpenID configuration may be stored in a database or some other format and specified programmatically through the configuration API.

For information on OpenID configuration, refer to the OpenID Connect for ASP.NET Core Configuration Guide.

IOpenIDProvider

The IOpenIDProvider interface supports OpenID Connect when acting as the OpenID provider.

The IOpenIDProvider will be made available, through dependency injection, to the application's controllers wishing to call the API.

```
public class OpenIDController : ControllerBase
```

```
{
   private readonly IOpenIDProvider _openIDProvider;

   public OpenIDController(IOpenIDProvider openIDProvider)
   {
      _openIDProvider = openIDProvider;
   }
}
```

Specifying an OpenID Configuration

The ConfigurationName property specifies the OpenID configuration to use.

Setting this property is only required for 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 this property shouldn't be set.

If there are multiple configurations, this property must be set to the name of the applicable configuration prior to any other OpenID API calls.

Refer to the OpenID Connect for ASP.NET Core Configuration Guide for more information.

```
string? ConfigurationName { get; set; }
```

Retrieving the OpenID Provider's Metadata

The GetMetadataAsync method returns the OpenID provider's metadata.

This is the OpenID provider metadata defined in the OpenID Connect Discovery specification available at https://openid.net/specs/openid-connect-discovery-1_0.html.

This method may be called to implement the ".well-known/openid-configuration" endpoint.

```
Task<IActionResult> GetMetadataAsync ();
```

Retrieving the OpenID Provider's Keys

The GetKeysAsync method returns the OpenID provider's public keys as a JSON Web Key Set document.

This method may be called to implement the "jwks_uri" endpoint.

```
Task<IActionResult> GetKeysAsync();
```

Receiving an Authentication Request

The ReceiveAuthnRequestAsync method receives an OpenID Connect authentication request from a client.

The received authentication request is returned.

```
Task<AuthenticationRequest> ReceiveAuthnRequestAsync();
```

Sending an Authentication Response

The SendAuthnResponseAsync method sends an OpenID Connect authentication response to a client.

```
Task<IActionResult> SendAuthnResponseAsync(string subject, IEnumerable<Claim> claims,
    string? accessToken = null, string? refreshToken = null, DateTime? utcAccessTokenExpiresAt =
null);
```

subject

The subject to include in the authentication response.

claims

The claims to include in the authentication response.

accessToken

The access token to include in the authentication response.

refreshToken

The refresh token to include in the authentication response.

utcAccessTokenExpiresAt

The UTC time when the access token expires.

Sending an Error Authentication Response

If an error occurs at the OpenID provider, an error authentication response may be returned to the client.

```
Task<IActionResult> SendAuthnResponseAsync(Exception exception);
```

exception

The exception that occurred.

```
Task<IActionResult> SendAuthnErrorResponseAsync(string error, string? errorDescription = null,
    string? errorUri = null);
```

error

The error. Refer to the OpenIDConstants.ErrorCodes for standard error codes.

errorDescription

The error description.

errorUri

The error URI.

Retrieving an ID Token

The GetTokensAsync method returns the ID token as part of the authorization code flow.

```
Task<IActionResult> GetTokensAsync(GetRefreshTokenResultAsync? getRefreshTokenResult =
null, GetClientCredentialsResultAsync? getClientCredentialsResult = null,
GetUserCredentialsResultAsync? getUserCredentialsResult = null);
```

getRefreshTokenResult

The delegate that returns the access token, its expiry, and the new refresh token for the given refresh token.

This is only required if supporting the refresh_token grant type.

```
public delegate Task<RefreshTokenResult> GetRefreshTokenResultAsync(string clientID, string
refreshToken);
```

getClientCredentialsResult

The delegate that returns the access token and its expiry.

This is only required if supporting the client_credentials grant type.

```
public delegate Task<ClientCredentialsResult> GetClientCredentialsResultAsync(string clientID,
string? scope = null);
```

getUserCredentialsResult

The delegate that returns the access token and its expiry.

This is only required if supporting the password grant type.

```
public delegate Task<UserCredentialsResult> GetUserCredentialsResultAsync(string clientID,
string? userName = null, string? userPassword = null, string? scope = null);
```

Retrieving the User Information

The GetUserInfoAsync method returns the user information.

```
Task<IActionResult> GetUserInfoAsync();
```

Receiving a Logout Request

The ReceiveLogoutRequestAsync method receives an OpenID Connect logout request from a client.

The received logout request is returned.

```
Task<LogoutRequest> ReceiveLogoutRequestAsync();
```

Sending a LogoutResponse

The SendLogoutResponseAsync method sends an OpenID Connect logout response to a client.

```
Task<IActionResult> SendLogoutResponseAsync();
```

Getting the Status

The GetStatusAsync method returns the OpenID session status.

The returned dictionary's keys are the client ID. The dictionary's values are the corresponding status for that client.

```
Task<IDictionary<string, IOpenIDStatus>> GetStatusAsync();
```

Creating a JWT Access Token

The CreateJwtAccessTokenAsync method creates a JWT access token.

```
Task<string> CreateJwtAccessTokenAsync(string clientID, string audience, string? subject, string? scope, IList<Claim>? claims = null, DateTime? utcJwtExpiresAt = null);
```

clientID

The client ID.

audience

The audience to include in the JWT.

subject

The subject to include in the JWT.

claims

The claims to include in the JWT.

utcJwtExpiresAt

The UTC time when the JWT expires.

Revoking an Access Token

The RevokeAccessTokenAsync method removes the access token from the internal access token cache so it won't be sent to the client.

```
Task RevokeAccessTokenAsync(string accessToken);
```

accessToken

The access token.

Clearing the Current Session

The ClearSessionAsync method clears the internal state for the current session.

```
Task ClearSessionAsync();
```

Error Handling

If an error occurs an exception is thrown.

It's recommended that all API calls be wrapped in a try/catch with the exception being processed as required.

Customizations

Several interfaces are exposed to enable custom implementations.

However, for most use cases, it's not expected this will be required.

Adding Services

The IServiceCollection extension method, AddOpenIDProvider, adds the various default implementations of the OpenID interfaces to the .NET default services container. This should be called in the application's Program class.

Some, or all, of these implementations may be replaced by calling IServiceCollection Add methods.

For example, the following code makes use of the default implementations of the OpenID interfaces except it replaces the IOidcSessionStore with a custom implementation.

```
// Add OpenID provider services.
builder.Services.AddOpenIDProvider(builder.Configuration.GetSection("OpenIDProvider"));

// Add a custom OpenID session store.
builder.Services.AddScoped<ISessionStore, CustomSessionStore>();
```

Dependency Injection and Third-Party IoC Containers

The following table specifies the interfaces, default implementations and lifetimes that must be defined to a third-party Inversion of Control container if the .NET default services container isn't being used.

Interface	Implementation	Lifetime
-	IOptionsMonitor<CertificateValidationOptions>	Transient
-	IOptionsMonitor<DistributedSessionStoreOptions>	Transient
-	IOptionsMonitor<HttpPostFormOptions>	Transient
-	IOptionsMonitor<OpenIDConfigurations>	Transient
IAccessTokenCache	AccessTokenCache	Transient
IAuthCodeCache	AuthCodeCache	Transient
ICache	DistributedCache	Transient

ICertificateLoader	CertificateLoader	Transient
ICertificateValidator	CertificateValidator	Transient
IClientAuthenticator	ClientAuthenticator	Transient
IConfigurationResolver	ConfigurationResolver	Scoped
IDistributedCache	MemoryDistributedCache	Singleton
IHttpContextAccessor	HttpContextAccessor	Transient
IHttpGetTransport	HttpGetTransport	Transient
IHttpPostForm	HttpPostForm	Transient
IHttpPostTransport	HttpPostTransport	Transient
IIDGenerator	IDGenerator	Transient
ILicense	License	Transient
ILoggerFactory	LoggerFactory	Transient
IOpenIDProvider	OpenIDProvider	Transient
IOpenIDStatus	OpenIDStatus	Transient
ISessionStore	DistributedSessionStore	Scoped
ITokenSecurity	TokenSecurity	Transient
ITransport	Transport	Transient

For most use cases, it's not expected that custom implementations of the above interfaces will be required.

ICache

The ICache interface caches session state and tokens.

For most use cases, it's not expected that custom implementations will be required.

Default Cache

A default implementation is included that caches to an in-memory implementation of the IDistributedCache interface.

This is suitable for single server deployments.

For web farm deployments, an IDistributedCache implementation such as the RedisCache or SqlServerCache should be specified.

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

IAuthCodeCache

The IAuthCodeCache interface caches authorization codes.

For most use cases, it's not expected that custom implementations will be required.

Default Authorization Code Cache

A default implementation is included that caches to the ICache implementation.

IAccessTokenCache

The IAccessTokenCache interface caches access tokens.

For most use cases, it's not expected that custom implementations will be required.

Default Access Token Cache

A default implementation is included that caches to the ICache implementation.

ICertificateValidator

The ICertificateValidator interface validates X.509 certificates to ensure they haven't expired or aren't otherwise invalid.

Consideration must be given to the performance impact associated with certificate validation during OpenID operations.

Certificate validators may be chained together if multiple implementations are required.

Default Certificate Validator

A default certificate validator is available and options for it may be set.

For example, the following code in the Program class turns on certificate chain checking.

```
using ComponentSpace.OpenID.Certificates;

builder.Services.Configure<CertificateValidationOptions>(options =>
{
   options.EnableChainCheck = true;
});

// Add OpenID Connect services.
builder.Services.AddOpenIDProvider(builder.Configuration.GetSection("OpenIDProvider"));
```

Alternatively, the options may be specified through configuration.

For example, the following section in appsettings.json turns on certificate chain checking.

```
"CertificateValidation": {
  " EnableChainCheck ": true
}
```

The following code makes use of this configuration.

```
using ComponentSpace.OpenID.Certificates;

// Configure the certificate validation.
builder.Services.Configure<CertificateValidationOptions>(
   Configuration.GetSection("CertificateValidation"));

// Add OpenID Connect services.
builder.Services.AddOpenIDProvider(builder.Configuration.GetSection("OpenIDProvider"));
```

The following code turns off expired certificate checking.

```
using ComponentSpace.OpenID.Certificates;

builder.Services.Configure<CertificateValidationOptions>(options =>
```

```
{
   options.EnableNotAfterCheck= false;
});

// Add OpenID Connect services.
builder.Services.AddOpenIDProvider(builder.Configuration.GetSection("OpenIDProvider"));
```

IConfigurationResolver

The IConfigurationResolver interface resolves OpenID configuration.

For most use cases, it's not expected that custom implementations will be required.

Default Configuration Resolver

A default implementation is included that resolves configuration through the configuration specified either using a configuration file (eg. appsettings.json) or programmatically.

Refer to the OpenID Connect for ASP.NET Core Configuration Guide for more details.

IHttpPostForm

The IHttpPostForm interface creates the HTML form that's used when sending messages via HTTP Post.

A default implementation is included.

For most use cases, it's not expected that custom implementations will be required.

Default Http Post Form

A default implementation is included which uses the following HTML template.

The default HTML template may be replaced through the HttpPostFormOptions.FormTemplate property.

```
<html>
   <body>
      <noscript>
         <p>
            Since your browser doesn't support JavaScript, you must press the Continue button to
proceed.
         </p>
      </noscript>
      {displayMessage}
      <form id=""openidform"" action=""{url}"" method=""post"" target=""{target}"">
         <div>
            {hiddenFormVariables}
         </div>
         <noscript>
            <div>
               <input type=""submit"" value=""Continue""/>
            </div>
         </noscript>
      </form>
```

```
  </body>
  {javaScript}
</html>
```

The following substitution parameters are supported.

displayMessage [optional]

The {displayMessage} is displayed in the browser while the HTML form is being posted.

url

The {url} is the action URL for the HTTP Post.

target

The {target} is the target URL for the HTTP Post (i.e. _self, _blank, _parent or _top).

hiddenFormVariables

The {hiddenFormVariables} are the hidden form inputs containing the information to be posted.

javaScript

The {javaScript} is the inline JavaScript used to automatically submit the HTML form.

The default inline JavaScript may be replaced through the HttpPostFormOptions.JavaScript property.

```
<script>
  function submitForm() {
    document.forms.openidform.submit();
  }

  if (document.readyState === ""loading"") {
                document.addEventListener(""DOMContentLoaded"", submitForm);
        } else {
                submitForm();
  }
</script>
```

HTTP Post Form Options

Options associated with the HTTP Post form may be set.

For example, the following code changes the target to a new tab in the browser.

```
using ComponentSpace.OpenID.Transport.Post;

builder.Services.Configure<HttpPostFormOptions>(options =>
{
  options.Target = "_blank";
});
```

```
// Add OpenID Connect services.
builder.Services.AddOpenIDProvider(builder.Configuration.GetSection("OpenIDProvider"));
```

Alternatively, the options may be specified through configuration.

For example, the following section in appsettings.json specifies the browser target.

```
"HttpPostForm": {
  "Target": "_blank"
}
```

The following code makes use of this configuration.

```
using ComponentSpace.OpenID.Transport.Post;

// Configure the HTTP Post Form.
builder.Services.Configure<HttpPostFormOptions>(Configuration.GetSection("HttpPostForm"));

// Add OpenID Connect services.
builder.Services.AddOpenIDProvider(builder.Configuration.GetSection("OpenIDProvider"));
```

Content-Security-Policy Header Support

Content Security Policy (CSP) permits the control of resources, including JavaScripts, that the browser may load. It helps detect and protect against Cross Site Scripting (XSS) and other forms of attack.

CSP is specified through a Content-Security-Policy header sent to the browser. This also may be achieved through an equivalent <meta> element.

As the HTML form used to support HTTP-Post includes JavaScript, the CSP, if specified, must enable its loading.

Unsafe Inline

A policy allowing all inline script to load is possible but not recommended.

```
Content-Security-Policy: script-src 'unsafe-inline'
```

Nonce

A nonce may be added to the JavaScript to identify it and permit its loading through policy.

```
<script nonce="2BAC238EBCE24A24ABCC11132361D228">
   function submitForm() {
      document.forms.openidform.submit();
   }

   if (document.readyState === "loading") {
```

```
    document.addEventListener("DOMContentLoaded", submitForm);
  } else {
    submitForm();
  }
</script>
```

The corresponding policy would include the nonce.

```
Content-Security-Policy: script-src 'nonce-2BAC238EBCE24A24ABCC11132361D228'
```

A nonce may be included by specifying the HttpPostFormOptions.ContentSecurityPolicy.

```
using ComponentSpace.OpenID.Transport.Post;

// When using HTTP-Post, include a Nonce Content-Security-Policy header.
builder.Services.Configure<HttpPostFormOptions>(options =>
{
  options.ContentSecurityPolicy = HttpPostFormOptions.ContentSecurityPolicyOption.Nonce;
});

// Add OpenID Connect services.
builder.Services.AddOpenIDProvider(builder.Configuration.GetSection("OpenIDProvider"));
```

Hash

A hash may be used to identify the JavaScript and permit its loading through policy.

The corresponding policy would include the hash.

```
Content-Security-Policy: script-src 'sha256- oJqv2rhhrRCF1O504qOiwpGkD/R3s5/Btx1EFtIkfcU='
```

A hash may be included by specifying the HttpPostFormOptions.ContentSecurityPolicy.

```
using ComponentSpace.OpenID.Transport.Post;

// When using HTTP-Post, include a Hash Content-Security-Policy header.
builder.Services.Configure<HttpPostFormOptions>(options =>
{
  options.ContentSecurityPolicy = HttpPostFormOptions.ContentSecurityPolicyOption.Hash;
});

// Add OpenID Connect services.
builder.Services.AddOpenIDProvider(builder.Configuration.GetSection("OpenIDProvider"));
```

Trusted Site

Rather than using inline script, a separate script file may be downloaded from a trusted site.

Typically, this will be the application site.

The script file contains the following JavaScript.

```
function submitForm() {
   document.forms.openidform.submit();
}

if (document.readyState === "loading") {
   document.addEventListener("DOMContentLoaded", submitForm);
} else {
   submitForm();
}
```

The corresponding policy would include self (i.e. the origin site) as a trusted source.

```
Content-Security-Policy: script-src 'self'
```

Self may be included by specifying the HttpPostFormOptions.ContentSecurityPolicy and JavaScript source path.

```
using ComponentSpace.OpenID.Transport.Post;

// When using HTTP-Post, include a Self Content-Security-Policy header
// and use a JavaScript file rather than inline JavaScript.
builder.Services.Configure<HttpPostFormOptions>(options =>
{
   options.ContentSecurityPolicy = HttpPostFormOptions.ContentSecurityPolicyOption.Self;
   options.JavaScript = "<script src=\"/js/openid.js\"></script>";
});

// Add OpenID Connect services.
builder.Services.AddOpenIDProvider(builder.Configuration.GetSection("OpenIDProvider"));
```

ISessionStore

The ISessionStore interface supports storing OpenID session data.

For most use cases, it's not expected that custom implementations will be required.

Default Session Store

The default implementation stores OpenID session data in an ICache.

Session Store Cookie

The key to the cache for individual session data is kept in an OpenID-specific HTTP cookie.

By default, the cookie's name is "openid-session" and it's marked as secure, samesite=none and HTTP only.

An example set-cookie response header is shown below.

```
set-cookie: openid-session=74225c85-20c5-4535-a289-4b173ff23e4a; path=/; secure;
samesite=none; httponly
```

SameSite Cookie Considerations

The OpenID session cookie is created with a SameSite value of None.

However, if the MinimumSameSitePolicy for the application is set to SameSiteMode.Lax or SameSiteMode.Strict, the OIDC session cookie will take on this minimum setting. This will mean the browser won't return the cookie and the corresponding OpenID session state cannot be identified.

To circumvent these issues, the recommended approach is to set MinimumSameSitePolicy to SameSiteMode.None and to specify the SameSite setting on individual cookies as required.

```
builder.Services.Configure<CookiePolicyOptions>(options =>
{
  options.MinimumSameSitePolicy = SameSiteMode.None;
});
```

Session Store Options

Options associated with the session store may be set.

For example, the following code specifies the name of the OpenID cookie.

```
using ComponentSpace.OpenID.Session;

builder.Services.Configure<SessionStoreOptions>(options =>
{
   options.CookieName = "my-openid-session";
});

// Add OpenID Connect services.
builder.Services.AddOpenIDProvider(builder.Configuration.GetSection("OpenIDProvider"));
```

Alternatively, the options may be specified through configuration.

For example, the following section in appsettings.json specifies the name of the OpenID cookie.

```
"SessionStore": {
  "CookieName": "my-openid-session"
}
```

The following code makes use of this configuration.

```
using ComponentSpace.OpenID.Session;

// Configure the OpenID session store.
```

```
builder.Services.Configure<SessionStoreOptions>(Configuration.GetSection("SessionStore"));

// Add OpenID Connect services.
builder.Services.AddOpenIDProvider(builder.Configuration.GetSection("OpenIDProvider"));
```

The cookie's domain defaults to the host name. The following example sets the session cookie's domain. This is necessary if OpenID Connect occurs across subdomains.

```
using ComponentSpace.OpenID.Session;

builder.Services.Configure<SessionStoreOptions>(options =>
{
   options.CookieOptions.Domain = "componentspace.com";
});

// Add OpenID Connect services.
builder.Services.AddOpenIDProvider(builder.Configuration.GetSection("OpenIDProvider"));
```