Wilson 5.2.3

Microsoft.IdentityModel.Tokens.Jwt

# Introduction

The goal of this assembly is to improve the user experience by simplifying and improving performance when creating and validating JWT tokens. This library will make the following improvements:

* Remove automatic short-to-long claim type mapping that occurs when tokens are created.
* Prepare a framework for future work that will involve asynchronous token validation and creation.
* Improve the speed of JWT token validation and creation.
* Simplify the way in which JWT tokens are stored and dealt with.
* Provide a single extensibility model using delegates. Previous models provided two models: virtual methods and delegates.

# Services Provided

1. Token Creation
2. Token Validation

# API Set

The API set is focused around 2 main classes. JsonWebToken is used to represent JWT tokens in a simpler, more intuitive way than JwtSecurityToken. JsonWebTokenHandler is able to create, read, and validate JWT tokens. A few additional structures (such as TokenValidationResult) and utilities have been created to facilitate calling and returning results.

**NOTE**: We plan to have async APIs for both token validation and token creation. Eventually, token handlers will be added that will support SAML, SAML2, and CBOR tokens.

JsonWebToken is used to:

1. Create a JsonWebToken from a JWT encoded string.
2. Create a JsonWebToken from JObjects representing the JWT header and the JWT payload.
3. Easily retrieve properties and claims from a JWT token.

public class JsonWebToken : SecurityToken {

public JsonWebToken(JObject header, JObject payload);

public JsonWebToken(string jwtEncodedString);

public string Actor { get; }

public string Alg { get; }

public IEnumerable<string> Audiences { get; }

public virtual IEnumerable<Claim> Claims { get; }

public string Cty { get; }

public JObject Header { get; set; }

public override string Id { get; }

public DateTime IssuedAt { get; }

public override string Issuer { get; }

public string Kid { get; }

public JObject Payload { get; set; }

public string RawData { get; }

public override SecurityKey SecurityKey { get; }

public override SecurityKey SigningKey { get; set; }

public string Subject { get; }

public string Typ { get; }

public override DateTime ValidFrom { get; }

public override DateTime ValidTo { get; }

public string X5t { get; }

}

JsonWebTokenHandler is used to:

1. Create JsonWebTokens
2. Validate JsonWebTokens
3. Read JsonWebTokens

public class JsonWebTokenHandler : TokenValidator {

public JsonWebTokenHandler();

public override Type TokenType { get; }

public override bool CanReadToken(string token);

public override bool CanValidateToken();

public override bool CanWriteToken();

public string CreateToken(JObject payload, SigningCredentials signingCredentials);

public JsonWebToken ReadToken(string token);

public override SecurityToken ReadToken(string token);

public override SecurityToken ReadToken(XmlReader reader, TokenValidationParameters validationParameters);

public TokenValidationResult ValidateToken(string token, TokenValidationParameters validationParameters);

public override string WriteToken(SecurityToken token);

}

TokenValidationResult stores the results of a token validation operation:

public class TokenValidationResult

{

public TokenValidationResult();

public SecurityToken SecurityToken { get; set; }

}

# Sample code

### **Token Creation**

var tokenHandler = new JsonWebTokenHandler();

var signingCredentials = KeyingMaterial.JsonWebKeyRsa256SigningCredentials;

var payload = new JObject()

{

{ JwtRegisteredClaimNames.Email, "Bob@contoso.com"},

{ JwtRegisteredClaimNames.GivenName, "Bob"},

{ JwtRegisteredClaimNames.Iss, "http://Default.Issuer.com" },

{ JwtRegisteredClaimNames.Aud, "http://Default.Audience.com" },

{ JwtRegisteredClaimNames.Nbf, "2017-03-18T18:33:37.080Z" },

{ JwtRegisteredClaimNames.Exp, "2021-03-17T18:33:37.080Z" }

};

var accessToken = tokenHandler.CreateToken(payload, signingCredentials);

### **Token Validation**

var tokenHandler = new JsonWebTokenHandler();

var accessToken = "eyJhbGciOiJSUzI1NiIsImtpZCI6IlJzYVNlY3VyaXR5S2V5XzIwNDgiLCJ0eXAiOiJKV1QifQ.eyJlbWFpbCI6IkJvYkBjb250b3NvLmNvbSIsImdpdmVuX25hbWUiOiJCb2IiLCJpc3MiOiJodHRwOi8vRGVmYXVsdC5Jc3N1ZXIuY29tIiwiYXVkIjoiaHR0cDovL0RlZmF1bHQuQXVkaWVuY2UuY29tIiwibmJmIjoiMjAxNy0wMy0xOFQxODozMzozNy4wODBaIiwiZXhwIjoiMjAyMS0wMy0xN1QxODozMzozNy4wODBaIn0.JeUhB3r\_BBiImzySSQ5qBO0HqE6-mkW5vQDr6Yocfu7pLluAxS854PXMXuIOlbiV9TCQAUDw8UjaxryaCEFRDqfAxl\_nfMXn4K7iRc691ft9TL1qw9y40cjc16McBHc-lpu1F0lnXYNW9vGdxkQHpSQLDsVxAzyKXNypLYyNPwlZJp\_G1Gx7fuVxOQOyMgZ-wcTx1c-mQmozLVQJ6r8-XC4LLVVotwjTQqZzVRhyPoMFHP\_6auPA77P0JaiFnl3KMsASDmE3EMF5iOLBWzR0XqHLB9HNqdp0cVQQroSxvU7YJoE9jVFX6KfHusg5blsudlR0v4vv-1rhL9uFqRDNfw";

var tokenValidationParameters = new TokenValidationParameters()

{

ValidAudience = "http://Default.Audience.com",

ValidIssuer = "http://Default.Issuer.com",

IssuerSigningKey = KeyingMaterial.JsonWebKeyRsa256SigningCredentials.Key

};

var tokenValidationResult = tokenHandler.ValidateToken(accessToken, tokenValidationParameters);

var jsonWebToken = tokenValidationResult.SecurityToken as JsonWebToken;

var email = jsonWebToken.Payload.Value<string>(JwtRegisteredClaimNames.Email);

// Retrieving a claim value that isn’t provided as a JsonWebToken property

if (!email.Equals("Bob@contoso.com"))

### throw new SecurityTokenException("Token does not contain the correct value for the 'email' claim.");

### **Token Reading**

var tokenHandler = new JsonWebTokenHandler();

var accessToken = "eyJhbGciOiJSUzI1NiIsImtpZCI6IlJzYVNlY3VyaXR5S2V5XzIwNDgiLCJ0eXAiOiJKV1QifQ.eyJlbWFpbCI6IkJvYkBjb250b3NvLmNvbSIsImdpdmVuX25hbWUiOiJCb2IiLCJpc3MiOiJodHRwOi8vRGVmYXVsdC5Jc3N1ZXIuY29tIiwiYXVkIjoiaHR0cDovL0RlZmF1bHQuQXVkaWVuY2UuY29tIiwibmJmIjoiMjAxNy0wMy0xOFQxODozMzozNy4wODBaIiwiZXhwIjoiMjAyMS0wMy0xN1QxODozMzozNy4wODBaIn0.JeUhB3r\_BBiImzySSQ5qBO0HqE6-mkW5vQDr6Yocfu7pLluAxS854PXMXuIOlbiV9TCQAUDw8UjaxryaCEFRDqfAxl\_nfMXn4K7iRc691ft9TL1qw9y40cjc16McBHc-lpu1F0lnXYNW9vGdxkQHpSQLDsVxAzyKXNypLYyNPwlZJp\_G1Gx7fuVxOQOyMgZ-wcTx1c-mQmozLVQJ6r8-XC4LLVVotwjTQqZzVRhyPoMFHP\_6auPA77P0JaiFnl3KMsASDmE3EMF5iOLBWzR0XqHLB9HNqdp0cVQQroSxvU7YJoE9jVFX6KfHusg5blsudlR0v4vv-1rhL9uFqRDNfw";

var jsonWebToken = tokenHandler.ReadToken(accessToken)

NOTE: You can simply pass the accessToken string into the constructor for a JsonWebToken and achieve the same result:

var jsonWebToken = new JsonWebToken(accessToken)

## Potential additions for the 5.2.4 release

* TokenValidator
  + Handlers such as the newly added JsonWebTokenHandler will call into TokenValidator for the purposes of token validation.
* TokenCreator
  + Handlers such as the newly added JsonWebTokenHandler will call into TokenCreator for the purposes of token creation.

TokenValidator is used to:

1. Validate JWT, SAML, and SAML2 tokens.

public class TokenValidator

{

public TokenValidationResult Validate(string token, TokenValidationParameters validationParameters, string tokenType);

public TokenValidationResult Validate(string token, string audience, string authority, string tokenType);

}

TokenCreator is used to:

1. Create JWT, SAML, and SAML2 tokens.

The specific API set is currently to be determined.