ComponentSpace
SAML for ASP.NET Core
Configuration Guide

Contents

Introduction

The ComponentSpace SAML for ASP.NET Core is a .NET standard class library that provides SAML v2.0 assertions, protocol messages, bindings and profiles functionality.

The primary SAML APIs, defined by the ISamlIdentityProvider and ISamlServiceProvider interfaces, make use of SAML configuration for various settings such as SAML provider names, X.509 certificates, URLs and various flags affecting processing. For more information about these APIs, refer to the SAML for ASP.NET Core Developer Guide.

SAML configuration may be specified either as JSON or programmatically.

SAML Configuration Options

There are a number of options for specifying SAML configuration.

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

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

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

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

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

A SamlDatabaseConfigurationResolver is available for storing SAML configuration in a database accessed through the Entity Framework.

SAML Configuration JSON

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

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

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

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

```
WebHost.CreateDefaultBuilder(args)
```

```
  .ConfigureAppConfiguration((configurationBuilder) =>
  {
    configurationBuilder.AddJsonFile("saml-config.json");
  })
  .UseStartup<Startup>()
  .Build();
```

Identity Provider Example Configuration

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

```
"SAML": {
 "$schema": "https://www.componentspace.com/schemas/saml-config-schema-v1.0.json",
 "Configurations": [
  {
    "LocalIdentityProviderConfiguration": {
     "Name": "https://ExampleIdentityProvider",
     "Description": "Example Identity Provider",
     "SingleSignOnServiceUrl": "https://localhost:44313/SAML/SingleSignOnService",
     "LocalCertificates": [
      {
        "FileName": "certificates/idp.pfx",
        "Password": "password"
      }
     ]
    },
    "PartnerServiceProviderConfigurations": [
     {
       "Name": "https://ExampleServiceProvider",
       "Description": "Example Service Provider",
       "AssertionConsumerServiceUrl":
"https://localhost:44360/SAML/AssertionConsumerService",
       "SingleLogoutServiceUrl": "https://localhost:44360/SAML/SingleLogoutService",
       "PartnerCertificates": [
        {
          "FileName": "certificates/sp.cer"
        }
       ]
     }
    ]
   }
  ]
}
```

Service Provider Example Configuration

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

```
"SAML": {
 "$schema": "https://www.componentspace.com/schemas/saml-config-schema-v1.0.json",
```

```
   "Configurations": [
    {
      "LocalServiceProviderConfiguration": {
        "Name": "https://ExampleServiceProvider",
        "Description": "Example Service Provider",
        "AssertionConsumerServiceUrl":
 "https://localhost:44360/SAML/AssertionConsumerService",
        "LocalCertificates": [
          {
            "FileName": "certificates/sp.pfx",
            "Password": "password"
          }
        ]
      },
      "PartnerIdentityProviderConfigurations": [
        {
          "Name": "https://ExampleIdentityProvider",
          "Description": "Example Identity Provider",
          "SingleSignOnServiceUrl": "https://localhost:44313/SAML/SingleSignOnService",
          "SingleLogoutServiceUrl": "https://localhost:44313/SAML/SingleLogoutService",
          "PartnerCertificates": [
            {
              "FileName": "certificates/idp.cer"
            }
          ]
        }
      ]
    }
  ]
}
```

JSON Schema

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

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

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

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

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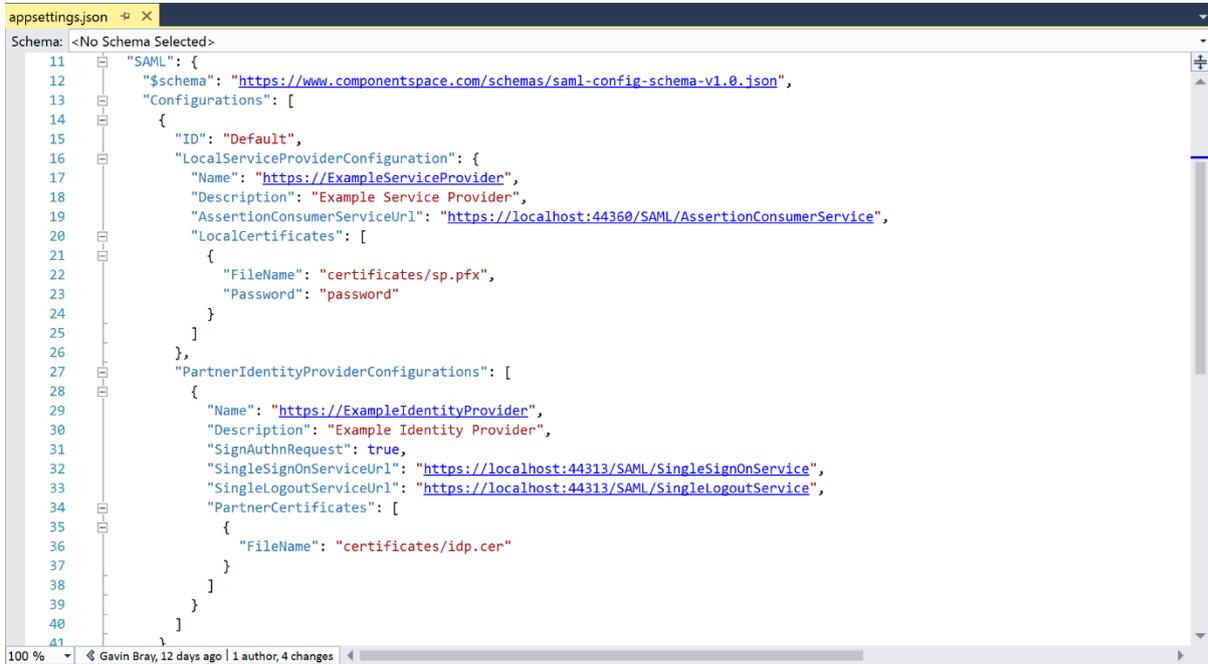
```
}
```

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.
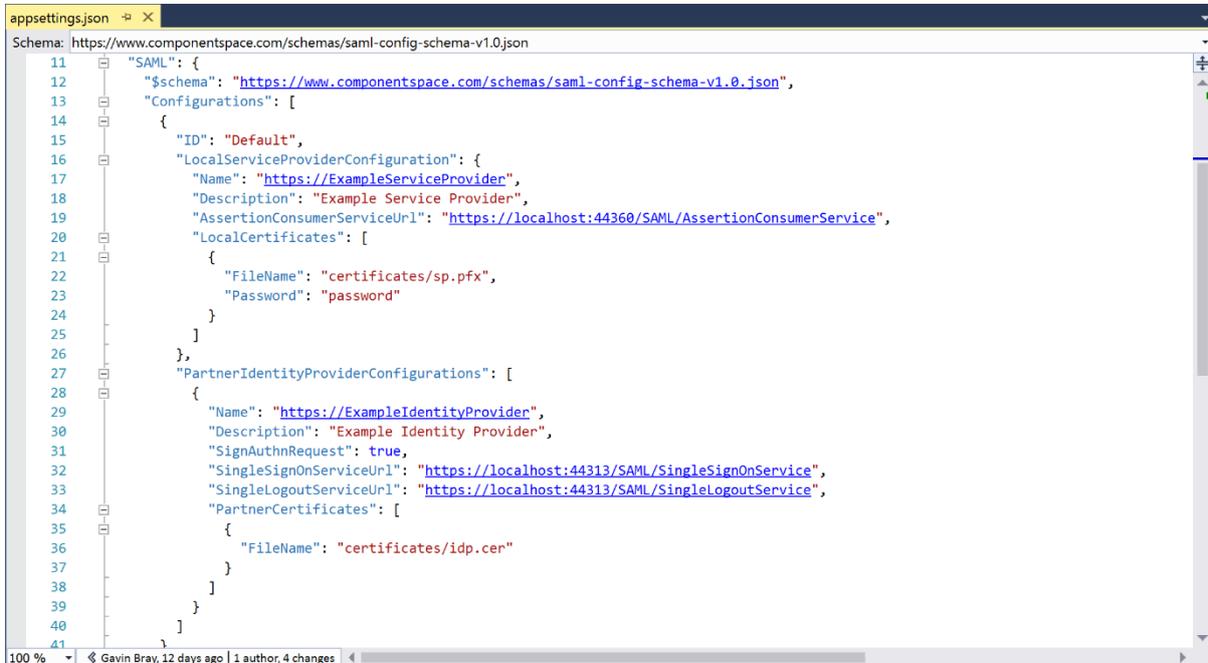
Intellisense now should be enabled.
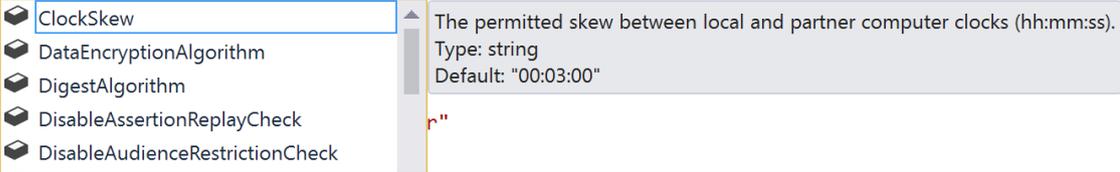
Note that schema validation is not performed at runtime.

Programmatically Specifying Configuration

In many scenarios, storing the SAML 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 SAML configuration API.

In the application's Program class, the configuration delegate is registered.

```
builder.Services.AddSaml(config => ConfigureSaml(config));
```

ConfigureSaml is an action delegate with the following method signature.

```
private void ConfigureSaml(SamlConfigurations samlConfigurations)
```

Identity Provider Example Configuration

The following is an example of setting the identity provider configuration programmatically.

Typically, rather than setting hard-coded values, these would be read from a custom database.

```
private void ConfigureSaml(SamlConfigurations samlConfigurations)
{
    samlConfigurations.Configurations = new List<SamlConfiguration>()
    {
        new SamlConfiguration()
        {
            LocalIdentityProviderConfiguration = new LocalIdentityProviderConfiguration()
            {
                Name = "https://ExampleIdentityProvider",
                Description = "Example Identity Provider",
                SingleSignOnServiceUrl = "https://localhost:44313/SAML/SingleSignOnService",
```

```
          LocalCertificates = new List<Certificate>()
          {
            new Certificate()
            {
              FileName = "certificates/idp.pfx",
              Password = "password"
            }
          }
        },
        PartnerServiceProviderConfigurations = new
List<PartnerServiceProviderConfiguration>()
        {
          new PartnerServiceProviderConfiguration()
          {
            Name = "https://ExampleServiceProvider",
            Description = "Example Service Provider",
            AssertionConsumerServiceUrl =
"https://localhost:44360/SAML/AssertionConsumerService",
            SingleLogoutServiceUrl = "https://localhost:44360/SAML/SLOService",
            PartnerCertificates = new List<Certificate>()
            {
              new Certificate()
              {
                FileName = "certificates/sp.cer"
              }
            }
          }
        }
      }
    }
  };
}
```

Service Provider Example Configuration

The following is an example of setting the service provider configuration programmatically.

Typically, rather than setting hard-coded values, these would be read from a custom database.

```
private void ConfigureSaml(SamlConfigurations samlConfigurations)
{
  samlConfigurations.Configurations = new List<SamlConfiguration>()
  {
    new SamlConfiguration()
    {
      LocalServiceProviderConfiguration = new LocalServiceProviderConfiguration()
      {
        Name = "https://ExampleServiceProvider",
        Description = "Example Service Provider",
        AssertionConsumerServiceUrl =
"https://localhost:44360/SAML/AssertionConsumerService",
```

```
          LocalCertificates = new List<Certificate>()
        {
          new Certificate()
          {
            FileName = "certificates/sp.pfx",
            Password = "password"
          }
        }
      },
      PartnerIdentityProviderConfigurations = new
List<PartnerIdentityProviderConfiguration>()
      {
        new PartnerIdentityProviderConfiguration()
        {
          Name = "https://ExampleIdentityProvider",
          Description = "Example Identity Provider",
          SingleSignOnServiceUrl = "https://localhost:44313/SAML/SingleSignOnService",
          SingleLogoutServiceUrl = "https://localhost:44313/SAML/SingleLogoutService",
          PartnerCertificates = new List<Certificate>()
          {
            new Certificate()
            {
              FileName = "certificates/idp.cer"
            }
          }
        }
      }
    }
  };
}
```

Updating Configuration

The current SAML configuration may be accessed through dependency injection.

The following is an example of accessing and updating the SAML configuration.

```
public class SamlController : Controller
{
  private readonly SamlConfigurations _samlConfigurations;

  public SamlController(IOptionsSnapshot<SamlConfigurations> samlConfigurations)
  {
    _samlConfigurations = samlConfigurations.Value;
  }

  public async Task<IActionResult> UpdateConfiguration()
  {
    var samlConfiguration = _samlConfigurations.Configurations.First();
```

```
    // Update the SAML configuration.
    samlConfiguration.PartnerIdentityProviderConfigurations.Add(new
PartnerIdentityProviderConfiguration()
    {
      Name = "https://ExampleIdentityProvider2",
      Description = "Example Identity Provider 2",
      SingleSignOnServiceUrl = "https://localhost:44314/SAML/SingleSignOnService",
      SingleLogoutServiceUrl = "https://localhost:44314/SAML/SingleLogoutService",
      PartnerCertificates = new List<Certificate>()
      {
        new Certificate()
        {
          FileName = "certificates/idp2.cer"
        }
      }
    });

    return new EmptyResult();
  }
}
```

Implementing ISamlConfigurationResolver

The ISamlConfigurationResolver interface provides an alternative mechanism for specifying SAML configuration. Rather than calling the SAML configuration API to specify configuration, the ISamlConfigurationResolver interface is implemented to return SAML configuration as requested. This approach might be preferred to support very dynamic SAML configurations.

SAML Database Configuration Resolver

The SamlDatabaseConfigurationResolver is an implementation of the ISamlConfigurationResolver interface that retrieves SAML configuration stored in an Entity Framework database.

The reader is assumed to be familiar with the Entity Framework. For more information, refer to:

https://docs.microsoft.com/en-us/ef/core/

Registration

In the application's Program class, the configuration resolver is registered.

```
builder.Services.AddTransient<ISamlConfigurationResolver,
SamlDatabaseConfigurationResolver>();
```

The SamlDatabaseConfigurationResolver is included in the separate ComponentSpace.Saml2.Configuration.Database NuGet package.

SQL Server

The application may be configured to use SQL Server to store the SAML configuration.

The following example connection string in appsettings.json specifies the SAML configuration database.

```
"SamlConfigurationConnection":
"Server=localhost;Database=SamlConfiguration;Trusted_Connection=True;MultipleActiveRe
sultSets=true"
```

The following example code application's Program class adds the SamlConfigurationContext using this configuration.

```
// Add the SAML configuration database context.
builder.Services.AddDbContext<SamlConfigurationContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("SamlConfigurationConnection"),
    builder => builder.MigrationsAssembly("DatabaseServiceProvider")));
```

The MigrationsAssembly method specifies that the migrations will be in the application's assembly rather than in the SamlConfigurationContext's assembly.

SQLite

The application may be configured to use SQLite to store the SAML configuration.

The following example connection string in appsettings.json specifies the SAML configuration database.

```
"SamlConfigurationConnection": "Data Source=SamlConfiguration.db"
```

The following example code in the application's Program class adds the SamlConfigurationContext using this configuration.

```
// Add the SAML configuration database context.
builder.Services.AddDbContext<SamlConfigurationContext>(options =>
  options.UseSqlite(Configuration.GetConnectionString("SamlConfigurationConnection"),
    builder => builder.MigrationsAssembly("DatabaseServiceProvider")));
```

The MigrationsAssembly method specifies that the migrations will be in the application's assembly rather than in the SamlConfigurationContext's assembly.

Initial Migration

The following example Visual Studio Package Manager Console command creates the initial migration.

```
Add-Migration InitialCreate -Context SamlConfigurationContext -Project
DatabaseServiceProvider -StartupProject DatabaseServiceProvider -OutputDir
Data\Migrations\SamlConfiguration
```

Note that this has already been done for the example projects.

Database Creation

The following example Visual Studio Package Manager Console command creates the database.

```
Update-Database -Context SamlConfigurationContext -Project DatabaseServiceProvider -
StartupProject DatabaseServiceProvider
```

Note that this has already been done for the example projects.

Data Seeding

The application is responsible for seeding the database with SAML configuration.

Note that this is demonstrated by the example projects.

Remove Migration

The following example Visual Studio Package Manager Console command removes the latest migration.

```
Remove-Migration -Context SamlConfigurationContext -Project DatabaseServiceProvider -
StartupProject DatabaseServiceProvider
```

List Migrations

The following example Visual Studio Package Manager Console command lists the migrations.

```
Get-Migration -Context SamlConfigurationContext -Project DatabaseServiceProvider -
StartupProject DatabaseServiceProvider
```

Script Migrations

For production environments, it's recommended an SQL script is created, reviewed and run.

The following example Visual Studio Package Manager Console command creates a script.

```
Script-Migration -Idempotent -Context SamlConfigurationContext -Project
DatabaseServiceProvider
```

Model Changes

On any SAML package update, it's recommended to create a migration to pick up any changes to the SAML configuration model.

If the model hasn't changed, the generated migration will be empty and may be removed.

Otherwise, the database should be updated using the migration.

SAML Cached Configuration Resolver

The SamlCachedConfigurationResolver is an implementation of the ISamlConfigurationResolver interface that caches SAML configuration in memory that's retrieved from a backing configuration resolver.

The configuration resolver is registered in the application's Program class. In this example, the SamlDatabaseConfigurationResolver is used as the backing configuration resolver.

```
// Use the cached resolver backed by the database configuration resolver.
builder.Services.AddTransient<ISamlConfigurationResolver,
SamlCachedConfigurationResolver>();

builder.Services.AddTransient<SamlDatabaseConfigurationResolver>();

builder.Services.Configure<SamlCachedConfigurationResolverOptions>(options =>
{
    options.CacheSamlConfigurationResolver<SamlDatabaseConfigurationResolver>();
});
```

Implementing ISamlConfigurationResolver

As a convenience, when not all interface methods are to be implemented, the AbstractSamlConfigurationResolver class may be extended.

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

```
builder.Services.AddTransient<ISamlConfigurationResolver,
CustomConfigurationResolver>();
```

Identity Provider Example Configuration

The following is an example of implementing ISamlConfigurationResolver as the identity provider.

Typically, rather than setting hard-coded values, these would be read from a custom database.

```
using ComponentSpace.Saml2.Configuration.Resolver;

public class CustomConfigurationResolver : AbstractSamlConfigurationResolver
{
    public override Task<bool> IsLocalIdentityProviderAsync(string configurationName)
    {
        return Task.FromResult(true);
    }

    public override Task<LocalIdentityProviderConfiguration>
GetLocalIdentityProviderConfigurationAsync(string configurationName)
    {
        var localIdentityProviderConfiguration = new LocalIdentityProviderConfiguration()
```

```csharp
    {
      Name = "https://ExampleIdentityProvider",
      Description = "Example Identity Provider",
      SingleSignOnServiceUrl = "https://localhost:44313/SAML/SingleSignOnService",
      SingleLogoutServiceUrl = "https://localhost:44313/SAML/SingleLogoutService",
      ArtifactResolutionServiceUrl =
"https://localhost:44313/SAML/ArtifactResolutionService",
      LocalCertificates = new List<Certificate>()
      {
        new Certificate()
        {
          FileName = "certificates/idp.pfx",
          Password = "password"
        }
      }
    };

    return Task.FromResult(localIdentityProviderConfiguration);
  }

  public override Task<PartnerServiceProviderConfiguration>
GetPartnerServiceProviderConfigurationAsync(string configurationName, string
partnerName)
  {
    if (partnerName != "https://ExampleServiceProvider")
    {
      throw new SamlConfigurationException($"The partner service provider {partnerName}
is not configured.");
    }

    var partnerServiceProviderConfiguration = new PartnerServiceProviderConfiguration()
    {
      Name = "https://ExampleServiceProvider",
      Description = "Example Service Provider",
      AssertionConsumerServiceUrl =
"https://localhost:44360/SAML/AssertionConsumerService",
      SingleLogoutServiceUrl = "https://localhost:44360/SAML/SingleLogoutService",
      ArtifactResolutionServiceUrl =
"https://localhost:44360/SAML/ArtifactResolutionService",
      PartnerCertificates = new List<Certificate>()
      {
        new Certificate()
        {
          FileName = "certificates/sp.cer"
        }
      }
    };

    return Task.FromResult(partnerServiceProviderConfiguration);
  }
```

```
    public override Task<IList<string>> GetPartnerServiceProviderNamesAsync(string
configurationName)
    {
        IList<string> partnerServiceProviderNames = new List<string> {
"https://ExampleServiceProvider" };

        return Task.FromResult(partnerServiceProviderNames);
    }
}
```

Service Provider Example Configuration

The following is an example of implementing ISamlConfigurationResolver as the service provider.

Typically, rather than setting hard-coded values, these would be read from a custom database.

```
using ComponentSpace.Saml2.Configuration.Resolver;

public class CustomConfigurationResolver : AbstractSamlConfigurationResolver
{
    public override Task<bool> IsLocalServiceProviderAsync(string configurationName)
    {
        return Task.FromResult(true);
    }

    public override Task<LocalServiceProviderConfiguration>
GetLocalServiceProviderConfigurationAsync(string configurationName)
    {
        var localServiceProviderConfiguration = new LocalServiceProviderConfiguration()
        {
            Name = "https://ExampleServiceProvider",
            Description = "Example Service Provider",
            AssertionConsumerServiceUrl =
"https://localhost:44360/SAML/AssertionConsumerService",
            SingleLogoutServiceUrl = "https://localhost:44360/SAML/SingleLogoutService",
            ArtifactResolutionServiceUrl =
"https://localhost:44360/SAML/ArtifactResolutionService",
            LocalCertificates = new List<Certificate>()
            {
                new Certificate()
                {
                    FileName = "certificates/sp.pfx",
                    Password = "password"
                }
            }
        };

        return Task.FromResult(localServiceProviderConfiguration);
    }
```

```
    public override Task<PartnerIdentityProviderConfiguration>
GetPartnerIdentityProviderConfigurationAsync(string configurationName, string
partnerName)
    {
      if (partnerName != "https://ExampleIdentityProvider")
      {
        throw new SamlConfigurationException($"The partner identity provider {partnerName}
is not configured.");
      }

      var partnerIdentityProviderConfiguration = new PartnerIdentityProviderConfiguration()
      {
        Name = "https://ExampleIdentityProvider",
        Description = "Example Identity Provider",
        SingleSignOnServiceUrl = "https://localhost:44313/SAML/SingleSignOnService",
        SingleLogoutServiceUrl = "https://localhost:44313/SAML/SingleLogoutService",
        ArtifactResolutionServiceUrl =
"https://localhost:44313/SAML/ArtifactResolutionService",
        PartnerCertificates = new List<Certificate>()
        {
          new Certificate()
          {
            FileName = "certificates/idp.cer"
          }
        }
      };

      return Task.FromResult(partnerIdentityProviderConfiguration);
    }

    public override Task<IList<string>> GetPartnerIdentityProviderNamesAsync(string
configurationName)
    {
      IList<string> partnerIdentityProviderNames = new List<string> {
"https://ExampleIdentityProvider" };

      return Task.FromResult(partnerIdentityProviderNames);
    }
}
```

Configuration Name Resolution

The ISamlConfigurationNameResolver interface maps SAML message issuer names to partner configuration names.

A default implementation performs a one-to-one mapping which is appropriate for most use cases.

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

```
using ComponentSpace.Saml2.Configuration.Resolver;

builder.Services.AddTransient<ISamlConfigurationNameResolver,
CustomSamlConfigurationNameResolver>();
```

Regular Expression Configuration Name Resolution

An implementation that supports regular expressions is available.

```
using ComponentSpace.Saml2.Configuration.Resolver;

builder.Services.AddTransient<ISamlConfigurationNameResolver,
RegexSamlConfigurationNameResolver>();
```

Using the regular expression configuration name resolver, the following example SAML partner identity provider configuration matches any tenant in an Entra ID multi-tenant configuration.

The Name "https://sts.windows.net/" is used as a regular expression pattern and will match SAML message issuer names such as "https://sts.windows.net/f2f933ec-d7c9-433f-8926-d3a0732a7dcf/".

```
{
  "Name": "https://sts.windows.net/",
  "Description": "Entra ID (Azure AD) multitenant",
  "SingleSignOnServiceUrl": "https://login.microsoftonline.com/common/saml2",
  "SingleLogoutServiceUrl": "https://login.microsoftonline.com/common/saml2",
  "PartnerCertificates": [
   {
     "FileName": "certificates/azure.cer"
   },
   {
     "FileName": "certificates/azure-common-1.cer"
   },
   {
     "FileName": "certificates/azure-common-2.cer"
   },
   {
     "FileName": "certificates/azure-common-3.cer"
   },
   {
     "FileName": "certificates/azure-common-4.cer"
   }
  ]
}
```

Multi-Tenancy Support

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

For the majority of use cases, a single SAML configuration will suffice, and multi-tenancy support is not required.

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

The following is an example outline of multiple SAML configurations.

```json
"SAML": {
  "$schema": "https://www.componentspace.com/schemas/saml-config-schema-v1.0.json",
  "Configurations": [
   {
     "Name": "Tenant1",
     "LocalServiceProviderConfiguration": {
     },
     "PartnerIdentityProviderConfigurations": [
     ]
   },
   {
     "Name": "Tenant2",
     "LocalServiceProviderConfiguration": {
     },
     "PartnerIdentityProviderConfigurations": [
     ]
   },
   {
     "Name": "Tenant3",
     "LocalServiceProviderConfiguration": {
     },
     "PartnerIdentityProviderConfigurations": [
     ]
   }
  ]
}
```

The Name property uniquely identifies each of the SAML configurations.

Configuration Selection

Prior to processing SSO and SLO requests, a SAML configuration must be selected. This is done by calling the SetConfigurationNameAsync method on the ISamlProvider interface. Refer to the SAML for ASP.NET Core Developer Guide for more information.

The following example specifies the SAML configuration to use when processing the SAML response.

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

// Specify the SAML configuration.
await _samlServiceProvider.SetConfigurationNameAsync(tenantName);
```

```
// Receive and process the SAML assertion contained in the SAML response.
var ssoResult = await _samlServiceProvider.ReceiveSsoAsync();
```

Identifying the Tenant

The application is responsible for identifying the tenant and therefore the name to specify when calling SetConfigurationNameAsync.

Possible methods include:

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

SamlConfigurations

The SamlConfigurations class is the top-level class specifying the SAML configurations.

Configurations [required]

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

SamlConfiguration

The SamlConfiguration class specifies a single SAML configuration for a local identity provider or service provider.

Name [optional]

Each SamlConfiguration is identified by a unique name. This name is internal to the configuration and is not exposed to partner providers.

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

LocalIdentityProviderConfiguration [optional]

The LocalIdentityProviderConfiguration specifies the local identity provider's configuration.

LocalServiceProviderConfiguration [optional]

The LocalServiceProviderConfiguration specifies the local service provider's configuration.

PartnerIdentityProviderConfigurations [optional]

The PartnerIdentityProviderConfigurations is the list of one or more PartnerIdentityProviderConfiguration items. Each PartnerIdentityProviderConfiguration specifies the configuration to participate in SSO with a partner identity provider.

PartnerServiceProviderConfigurations [optional]

The PartnerServiceProviderConfigurations is the list of one or more PartnerServiceProviderConfiguration items. Each PartnerServiceProviderConfiguration specifies the configuration to participate in SSO with a partner service provider.

LocalIdentityProviderConfiguration

The LocalIdentityProviderConfiguration specifies the configuration for the local identity provider.

Its base class is LocalProviderConfiguration.

SingleSignOnServiceUrl [optional]

The single sign-on service URL is the location of the local identity provider's SSO service where SAML authn requests are received as part of SP-initiated SSO.

If specified, it may be used to perform certain security checks as part of the SAML protocol.

Its use is optional but recommended.

LocalServiceProviderConfiguration

The LocalServiceProviderConfiguration specifies the configuration for the local service provider.

Its base class is LocalProviderConfiguration.

AssertionConsumerServiceUrl [optional]

The assertion consumer service URL is the location of the local service provider's ACS where SAML responses are received as part of SSO.

If specified, it may be used to perform certain security checks as part of the SAML protocol.

Its use is optional but recommended.

PartnerIdentityProviderConfiguration

The PartnerIdentityProviderConfiguration specifies the configuration for a partner identity provider.

Its base class is PartnerProviderConfiguration.

SingleSignOnServiceUrl [optional]

The single sign-on service URL is the location of the partner identity provider's SSO service where SAML authn requests are sent as part of SP-initiated SSO. If only IdP-initated SSO is supported, this URL may be omitted.

SingleSignOnServiceBinding [optional]

The single sign-on service binding specifies the transport mechanism (i.e. SAML binding) to use when sending SAML authn requests to the partner identity provider.

The binding options are:

- urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect

- urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST
- urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact

The default is urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect.

SignAuthnRequest [optional]

The flag specifies whether SAML authn requests sent to the partner identity provider should be signed. Signing authn requests is recommended but optional.

The default is true.

ForceAuthn [optional]

The flag specifies whether the force authentication attribute in SAML authn requests sent to the partner identity provider should be set.

The default is false.

WantAssertionOrResponseSigned [optional]

The flag specifies whether either SAML responses or assertions received from the partner identity provider should be signed. If the flag is set and neither the SAML response nor SAML assertion is signed or the signature cannot be verified, this is considered an error.

Signing ensures the identity of the sender and the integrity of the content. Signatures will be generated by the partner provider with its private key and verified by the local provider with the partner provider's public key.

Generally, it doesn't matter whether the SAML response or assertion is signed. The payload of the SAML response is the SAML assertion so signing the SAML response includes the SAML assertion.

It's recommended that WantAssertionOrResponseSigned is set to true.

The default is true.

WantSamlResponseSigned [optional]

The flag specifies whether SAML responses received from the partner identity provider should be signed. If the flag is set and either the SAML response isn't signed or the signature cannot be verified, this is considered an error.

Signing ensures the identity of the sender and the integrity of the content. Signatures will be generated by the partner provider with its private key and verified by the local provider with the partner provider's public key.

It's recommended that WantAssertionOrResponseSigned, WantSamlResponseSigned or WantAssertionSigned is set to true.

The default is false.

WantAssertionSigned [optional]

The flag specifies whether SAML assertions received from the partner identity provider should be signed. If the flag is set and either the SAML assertion isn't signed or the signature cannot be verified, this is considered an error.

Signing ensures the identity of the sender and the integrity of the content. Signatures will be generated by the partner provider with its private key and verified by the local provider with the partner provider's public key.

It's recommended that WantAssertionOrResponseSigned, WantSamlResponseSigned or WantAssertionSigned is set to true.

The default is false.

WantAssertionEncrypted [optional]

The flag specifies whether SAML assertions received from the partner identity provider should be encrypted. If the flag is set and either the SAML assertion isn't encrypted or cannot be decrypted, this is considered an error.

Encrypting ensures the privacy of the content. Assertions will be encrypted by the partner provider with the local provider's public key and decrypted by the local provider with its private key.

If the SAML assertion includes sensitive information it's recommended that it's encrypted. This SAML assertion encryption is in addition to the privacy provided at the transport layer when using the recommended HTTPS protocol. In many scenarios, encryption of the assertion is not required.

The default is false.

WantNameIDEncrypted [optional]

The flag specifies whether Name IDs received from the partner identity provider should be encrypted. If the flag is set and either the Name ID isn't encrypted or cannot be decrypted, this is considered an error.

Encrypting ensures the privacy of the content. Name IDs will be encrypted by the partner provider with the local provider's public key and decrypted by the local provider with its private key.

In many scenarios, encryption of the Name ID is not required.

The default is false.

ProviderName [optional]

The provider name is included in the SAML authn requests sent to the partner identity provider.

RequestedAuthnContexts [optional]

The request authentication context URIs are included in the SAML authn requests sent to the partner identity provider.

RequestedAuthnContextComparison [optional]

The comparison method is included in the SAML authn requests sent to the partner identity provider.

The comparison method is used to evaluate the requested contexts.

The comparison methods are:

- exact
- minimum
- maximum
- better

The default is to not include a comparison which is equivalent to specifying exact.

ExpectedAuthnContext [optional]

If the received SAML assertion includes an authentication statement with an authentication context and this doesn't match the expected authentication context, it's considered an error.

DisableIdPInitiatedSso [optional]

The flag specifies whether IdP-initiated SSO is supported.

Both IdP-initiated and SP-initiated SSO are supported.

Setting the flag to true disables IdP-initiated SSO.

The default is false.

DisableAssertionReplayCheck [optional]

The flag specifies whether checks for SAML assertion replay attacks are disabled.

Each SAML assertion includes a unique ID. A cache of received SAML assertion IDs is maintained and if an ID matches a previously received ID this is considered an error.

Setting the flag to true disables this check.

The default is false.

DisableRecipientCheck [optional]

A SAML assertion may include a subject confirmation recipient URI. This identifies the intended recipient of the SAML assertion. If included, it should specify the service provider's assertion consumer service URL.

Setting the flag to true disables this check.

The default is false.

DisableRecipientCheck [optional]

A SAML assertion may include a subject confirmation recipient URI. This identifies the intended recipient of the SAML assertion. If included, it should match the service provider's assertion consumer service URL specified by the AssertionConsumerServiceUrl configuration property.

Setting the flag to true disables this check.

The default is false.

DisableAudienceRestrictionCheck [optional]

A SAML assertion may include an audience restriction URI. This identifies the intended recipient of the SAML assertion. If included, it should match the service provider's name.

Setting the flag to true disables this check.

The default is false.

DisableAuthnContextCheck [optional]

A SAML assertion may include an authentication context. This identifies the mechanism by which the user was authenticated at the identity provider. For example, if the user was authenticated by password, the authentication context would be "urn:oasis:names:tc:SAML:2.0:ac:classes:Password". If included, it should match the authentication context class specified by the optional ExpectedAuthnContext configuration property.

Setting the flag to true disables this check.

The default is false.

PartnerServiceProviderConfiguration

The PartnerServiceProviderConfiguration specifies the configuration for a partner service provider.

Its base class is PartnerProviderConfiguration.

AssertionConsumerServiceUrl [optional]

The assertion consumer service URL is the location of the partner service provider's ACS where SAML responses are sent as part of SSO.

ValidAssertionConsumerServiceUrls [optional]

The valid assertion consumer service URLs are those accepted from the service provider.

If the service provider specifies an assertion consumer service URL as part of SP-initiated SSO, it must match with one of the URL patterns.

This may be used, for example, to ensure SAML responses are only sent to the intended domain or server.

If not specified, any URL is accepted.

WantAuthnRequestSigned [optional]

The flag specifies whether SAML authn requests received from the partner service provider should be signed. Receiving signed authn requests is recommended but optional.

The default is true.

SignSamlResponse [optional]

The flag specifies whether SAML responses sent to the partner service provider should be signed.

Signing ensures the identity of the sender and the integrity of the content. Signatures will be generated by the local provider with its private key and verified by the partner provider with the local provider's public key.

It's recommended that either SignSamlResponse or SignAssertion is set to true.

The default is false.

SignAssertion [optional]

The flag specifies whether SAML assertions sent to the partner service provider should be signed.

Signing ensures the identity of the sender and the integrity of the content. Signatures will be generated by the local provider with its private key and verified by the partner provider with the local provider's public key.

It's recommended that either SignSamlResponse or SignAssertion is set to true.

The default is true.

EncryptAssertion [optional]

The flag specifies whether SAML assertions sent to the partner service provider should be encrypted.

Encrypting ensures the privacy of the content. Assertions will be encrypted by the local provider with the partner provider's public key and decrypted by the partner provider with its private key.

If the SAML assertion includes sensitive information, it's recommended that it's encrypted. This SAML assertion encryption is in addition to the privacy provided at the transport layer when using the recommended HTTPS protocol. In many scenarios, encryption of the assertion is not required.

The default is false.

EncryptNameID [optional]

The flag specifies whether Name IDs sent to the partner service provider should be encrypted.

Encrypting ensures the privacy of the content. Name IDs will be encrypted by the local provider with the partner provider's public key and decrypted by the partner provider with its private key.

In many scenarios, encryption of the Name ID is not required.

The default is false.

AssertionLifeTime [optional]

The assertion lifetime specifies the time span for which the SAML assertion is valid. It is the current UTC time plus or minus the assertion lifetime time span.

The time span should be kept short but not so short as to cause issues when server clocks are not synchronized exactly.

The default is 3 minutes.

AuthnContext [optional]

The authentication context specifies the mechanism by which the user was authenticated at the identity provider.

For example, if the user was authenticated by password, the authentication context would be "urn:oasis:names:tc:SAML:2.0:ac:classes:Password".

The authentication context is included with the SAML assertion authentication statement.

RelayState [optional]

The relay state is sent as part of IdP-initiated SSO and specifies the URL the service provider should redirect to once SSO completes.

LocalProviderConfiguration

The LocalProviderConfiguration is an abstract base class.

Its base class is ProviderConfiguration.

DisableSchemaCheck [optional]

SAML messages should validate against the SAML XML schema.

Setting the flag to true disables this check.

The default is false.

ResolveToHttps [optional]

The flag specifies whether local URLs should be resolved to HTTPS.

This is useful when using an SSL terminating device such as a load balancer.

For example, if true, a local URL of http://www.sp.com/SAML/AssertionConsumerService would be resolved to https://www.sp.com/SAML/AssertionConsumerService when included in the SAML authn request sent to the identity provider.

The default is true.

SingleLogoutServiceUrl [optional]

The single logout service URL is the location of the local provider's SLO service where SAML logout messages are received. If SLO is not supported, this URL may be omitted.

If specified, it may be used to perform certain security checks as part of the SAML protocol.

Its use is optional but recommended.

ArtifactResolutionServiceUrl [optional]

The artifact resolution service URL is the location of the local provider's service where SAML artifact resolve requests are received. If the HTTP-Artifact binding is not supported, this URL may be omitted.

If specified, it may be used to perform certain security checks as part of the SAML protocol.

Its use is optional but recommended.

PartnerProviderConfiguration

The PartnerProviderConfiguration is an abstract base class.

Its base class is ProviderConfiguration.

PartnerCertificates [optional]

The PartnerCertificates specifies one or more X.509 certificates issued to the partner provider and used by the local provider. Typically, only a single certificate is specified. If more than one certificate is specified and a security operation using the certificates fails, the operation is retried using the next certificate in the list until either successful or all certificates have been tried.

As an example, if the SAML assertion received by the local service provider is signed each partner certificate is used in an attempt to verify the signature.

Multiple partner certificates support scenarios including the phased rollover of expired certificates.

AssertionConsumerServiceBinding [optional]

The assertion consumer service binding specifies the transport mechanism (i.e. SAML binding) to use when sending SAML responses to the service provider.

The binding options are:

- urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST
- urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact

The default is urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST.

SingleLogoutServiceUrl [optional]

The single logout service URL is the location of the partner provider's SLO service where SAML logout messages are sent. If SLO is not supported, this URL may be omitted.

SingleLogoutServiceResponseUrl [optional]

The single logout service response URL is the location of the partner provider's SLO service where SAML logout responses are sent. If SLO is not supported or the same partner provider endpoint receives logout requests and responses, this URL may be omitted.

SingleLogoutServiceBinding [optional]

The single logout service binding specifies the transport mechanism (i.e. SAML binding) to use when sending SAML logout messages to the partner provider.

The binding options are:

- urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect
- urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST
- urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact

The default is urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect.

ArtifactResolutionServiceUrl [optional]

The artifact resolution service URL is the location of the partner provider's service where SAML artifact resolve requests are received. If the HTTP-Artifact binding is not supported, this URL may be omitted.

ArtifactEncoding [optional]

The artifact encoding specifies the transport mechanism to use when sending SAML artifacts to the partner provider. If the HTTP-Artifact binding is not supported, this setting may be omitted.

The artifact encoding options are:

- Form
- Url

The default is to send the artifact encoded in the URL.

LogoutRequestLifeTime [optional]

The assertion lifetime specifies the time span for which the SAML logout request is valid. It is the current UTC time plus or minus the logout request lifetime time span.

The time span should be kept short but not so short as to cause issues when server clocks are not synchronized exactly.

The default is 3 minutes.

SignLogoutRequest [optional]

The flag specifies whether SAML logout requests sent to the partner provider should be signed.

Signing ensures the identity of the sender and the integrity of the content. Signatures will be generated by the local provider with its private key and verified by the partner provider with the local provider's public key.

The default is true.

SignLogoutResponse [optional]

The flag specifies whether SAML logout responses sent to the partner provider should be signed.

Signing ensures the identity of the sender and the integrity of the content. Signatures will be generated by the local provider with its private key and verified by the partner provider with the local provider's public key.

The default is true.

WantLogoutRequestSigned [optional]

The flag specifies whether SAML logout requests received from the partner provider should be signed. If the flag is set and either the logout request isn't signed or the signature cannot be verified, this is considered an error.

Signing ensures the identity of the sender and the integrity of the content. Signatures will be generated by the partner provider with its private key and verified by the local provider with the partner provider's public key.

The default is true.

WantLogoutResponseSigned [optional]

The flag specifies whether SAML logout responses received from the partner provider should be signed. If the flag is set and either the logout response isn't signed or the signature cannot be verified, this is considered an error.

Signing ensures the identity of the sender and the integrity of the content. Signatures will be generated by the partner provider with its private key and verified by the local provider with the partner provider's public key.

The default is true.

SignArtifactResolve [optional]

The flag specifies whether SAML artifact resolve requests sent to the partner provider should be signed.

Signing ensures the identity of the sender and the integrity of the content. Signatures will be generated by the local provider with its private key and verified by the partner provider with the local provider's public key.

The default is false.

SignArtifactResponse [optional]

The flag specifies whether SAML artifact responses sent to the partner provider should be signed.

Signing ensures the identity of the sender and the integrity of the content. Signatures will be generated by the local provider with its private key and verified by the partner provider with the local provider's public key.

The default is false.

WantArtifactResolveSigned [optional]

The flag specifies whether SAML artifact resolve requests received from the partner provider should be signed. If the flag is set and either the artifact resolve request isn't signed or the signature cannot be verified, this is considered an error.

Signing ensures the identity of the sender and the integrity of the content. Signatures will be generated by the partner provider with its private key and verified by the local provider with the partner provider's public key.

The default is false.

WantArtifactResponseSigned [optional]

The flag specifies whether SAML artifact responses received from the partner provider should be signed. If the flag is set and either the artifact response isn't signed or the signature cannot be verified, this is considered an error.

Signing ensures the identity of the sender and the integrity of the content. Signatures will be generated by the partner provider with its private key and verified by the local provider with the partner provider's public key.

The default is false.

EncryptLogoutNameID [optional]

The flag specifies whether Name IDs in logout requests sent to the partner service provider should be encrypted.

Encrypting ensures the privacy of the content. Name IDs will be encrypted by the local provider with the partner provider's public key and decrypted by the partner provider with its private key.

If the Name ID is sensitive information, it's recommended that it's encrypted. This Name ID encryption is in addition to the privacy provided at the transport layer when using the recommended HTTPS protocol. In many scenarios, encryption of the Name ID is not required.

The default is false.

IssuerFormat [optional]

The issuer format specifies the format of the issuer field included in SAML messages.

IssuerQualifier [optional]

The issuer qualifier specifies the qualifier of the issuer field included in SAML messages.

NameIDFormat [optional]

The name ID format specifies the format of the name identifier. For a local identity provider, the format is included with the SAML assertion name identifier. For a local service provider, the format is included with the SAML authentication request name identifier policy.

NameIDQualifier [optional]

The name ID qualifier specifies the qualifier of the name identifier. For a local identity provider, the qualifier is included with the SAML assertion name identifier. For a local service provider, the qualifier is included with the SAML authentication request name identifier policy.

DigestAlgorithm [optional]

The digest algorithm specifies how to generate the digest for XML signatures.

The supported digest algorithms are:

- http://www.w3.org/2000/09/xmldsig#sha1
- http://www.w3.org/2001/04/xmlenc#sha256
- http://www.w3.org/2001/04/xmldsig-more#sha384
- http://www.w3.org/2001/04/xmlenc#sha512

The default is http://www.w3.org/2001/04/xmlenc#sha256.

SignatureAlgorithm [optional]

The signature algorithm specifies how to generate XML signatures.

The supported signature algorithms are:

- http://www.w3.org/2000/09/xmldsig#rsa-sha1
- http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
- http://www.w3.org/2001/04/xmldsig-more#rsa-sha384
- http://www.w3.org/2001/04/xmldsig-more#rsa-sha512

- http://www.w3.org/2007/05/xmldsig-more#sha1-rsa-MGF1
- http://www.w3.org/2007/05/xmldsig-more#sha256-rsa-MGF1
- http://www.w3.org/2007/05/xmldsig-more#sha384-rsa-MGF1
- http://www.w3.org/2007/05/xmldsig-more#sha512-rsa-MGF1

- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha1
- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256
- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha384
- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha512

The default is http://www.w3.org/2001/04/xmldsig-more#rsa-sha256.

WantDigestAlgorithm [optional]

The digest algorithm specifies the required digest algorithm of received XML signatures.

Refer to DigestAlgorithm for valid values.

If unspecified, any digest algorithm is permitted.

WantSignatureAlgorithm [optional]

The signature algorithm specifies the required signature algorithm of received XML signatures.

Refer to SignatureAlgorithm for valid values.

If unspecified, any signature algorithm is permitted.

KeyEncryptionAlgorithm [optional]

The key encryption algorithm specifies how to encrypt the symmetric key used in XML encryption.

The supported key encryption algorithms are:

- http://www.w3.org/2001/04/xmlenc#rsa-1_5
- http://www.w3.org/2009/xmlenc11#rsa-oaep
- http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p

The default is http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p.

Note that the http://www.w3.org/2009/xmlenc11#rsa-oaep algorithm requires the use of the ComponentSpace.Saml2.XmlSecurity.RsaOaep extension package.

KeyEncryptionDigestAlgorithm [optional]

The key encryption digest algorithm is used by the http://www.w3.org/2009/xmlenc11#rsa-oaep and http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p key encryption algorithms.

The supported key encryption digest algorithms are:

- http://www.w3.org/2000/09/xmldsig#sha1
- http://www.w3.org/2001/04/xmlenc#sha256
- http://www.w3.org/2001/04/xmldsig-more#sha384
- http://www.w3.org/2001/04/xmlenc#sha512

The default is http://www.w3.org/2000/09/xmldsig#sha1.

KeyEncryptionMaskGenerationFunction [optional]

The key encryption mask generation function is used by the http://www.w3.org/2009/xmlenc11#rsa-oaep key encryption algorithm.

The supported key encryption mask generation functions are:

- http://www.w3.org/2009/xmlenc11#mgf1sha1
- http://www.w3.org/2009/xmlenc11#mgf1sha256
- http://www.w3.org/2009/xmlenc11#mgf1sha384
- http://www.w3.org/2009/xmlenc11#mgf1sha512

The default is http://www.w3.org/2009/xmlenc11#mgf1sha1.

DataEncryptionAlgorithm [optional]

The data encryption algorithm specifies how to encrypt the data in XML encryption.

The supported data encryption algorithms are:

- http://www.w3.org/2001/04/xmlenc#tripledes-cbc

- http://www.w3.org/2001/04/xmlenc#aes128-cbc
- http://www.w3.org/2001/04/xmlenc#aes192-cbc
- http://www.w3.org/2001/04/xmlenc#aes256-cbc

- http://www.w3.org/2009/xmlenc11#aes128-gcm
- http://www.w3.org/2009/xmlenc11#aes192-gcm
- http://www.w3.org/2009/xmlenc11#aes256-gcm

The AES-GCM algorithms require .NET Core 3.1 or later.

The default is http://www.w3.org/2001/04/xmlenc#aes256-cbc.

ClockSkew [optional]

The clock skew specifies the time span to allow for differences between local and partner computer clocks when checking time intervals.

The time span should be kept short but not so short as to cause issues when server clocks are not synchronized exactly.

The default is 3 minutes.

UseEmbeddedCertificate [optional]

The flag specifies whether to use the X.509 certificate embedded in the XML signature when verifying the signature.

If the embedded certificate is used, no assumptions can be made about the identity of the sender.

Embedded certificates should not be used in production.

The default is false.

EnableSha1Support [optional]

The flag specifies whether SHA-1 algorithms are supported.

The use of SHA-1 is not recommended.

The default is false.

DisableDestinationCheck [optional]

A SAML message may include a destination URI identifying the address to which the message has been sent. If included, it should match the provider's URL where the message was received.

For example, for a SAML response the destination should be the local service provider's assertion consumer service URL as specified by the AssertionConsumerServiceUrl configuration property.

Setting the flag to true disables this check.

The default is false.

DisableTimePeriodCheck [optional]

A SAML assertion or logout request may include attributes identifying a time period in which it is valid. If included, the time at which the SAML assertion or logout request is received should be within this time period.

Setting the flag to true disables this check.

The default is false.

DisableInboundLogout [optional]

Setting the flag to true disables inbound SAML logout requests.

The default is false.

DisableOutboundLogout [optional]

Setting the flag to true disables outbound SAML logout requests.

The default is false.

DisableInResponseToCheck [optional]

All SAML messages includes a unique ID. SAML responses that are in response to a particular SAML request include an in-response-to attribute identifying the SAML request.

Setting the flag to true disables checking to ensure the in-response-to attribute is present and correct.

The default is false.

DisablePendingLogoutCheck [optional]

If a SAML logout response is received without having previously sent a logout request, this is considered an error.

Setting the flag to true disables this check.

The default is false.

DisableLogoutResponseStatusCheck [optional]

If a SAML logout response is received with an error status, this is considered an error.

Setting the flag to true disables this check.

The default is false.

DisableClearAllSessionsOnLogout [optional]

On SAML logout, all sessions to the partner provider for the user are cleared as most partner providers don't support separate logout of multiple SSO sessions in the one browser session.

Setting the flag to true disables this and sessions are be logged out individually.

The default is false.

MappingRules [optional]

SAML mapping rules transform the SAML identity at either the identity provider or service provider.

The SAML identity consists of the SAML subject name identifier and the SAML attributes.

Mapping rules are applied in sequence.

Mapping rules are useful for moving partner provider specific differences in SAML identity information from the application into the SAML configuration.

ProviderConfiguration

The ProviderConfiguration specifies properties common to all local and partner providers.

Name [required]

All local and partner providers must have a unique name. Partner providers will supply their names. Local names should be universally unique and, for maximum interoperability, be in the form of a URL. The URL doesn't have to locate a resource but it's common for it to point to the home page of the web application or the download link to the local provider's SAML metadata.

The name corresponds to the entity ID, if SAML metadata is used.

Description [optional]

The description is purely for documentation and is not part of SAML SSO.

LocalCertificates [optional]

The LocalCertificates specifies one or more X.509 certificates issued to and used by the local provider. Typically, only a single certificate is specified. If more than one certificate is specified and a security operation using the certificates fails, the operation is retried using the next certificate in the list until either successful or all certificates have been tried.

As an example, if the SAML assertion received by the local service provider is encrypted each local certificate is used in an attempt to decrypt the SAML assertion.

Multiple local certificates support scenarios including the staggered rollover of expired certificates.

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.

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

Local and partner provider URLs may be absolute or relative.

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

For example, an assertion consumer service URL may be specified absolutely.

https://localhost:44360/SAML/AssertionConsumerService

Alternatively, it may be specified as a path.

/SAML/AssertionConsumerService

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

Although the more common use case it to specify relative local URLs, relative partner URLs may be specified if, for example, the local and partner provider are installed on the same server.

Mapping Rules

A SAML mapping rule transforms the SAML identity at either the identity provider or service provider.

The SAML identity consists of the SAML subject name identifier and the SAML attributes.

Mapping rules are executed in the order they're specified.

Rule [required]

The rule identifies the mapping rule by name.

Valid rule names are:

- Clear
- Constant
- Copy
- Keep
- Remove
- Rename

Name [optional]

The name identifies the SAML attribute by its name. If omitted, the rule applies to the SAML subject name identifier.

Value [optional]

The value is additional data required when applying the rule.

Clear Mapping Rule

The Clear mapping rule removes the SAML subject and all SAML attributes.

The following example removes the SAML subject name identifier and all SAML attributes.

```
{
  "Rule": "Clear"
}
```

Constant Mapping Rule

The Constant mapping rule creates a SAML subject or SAML attribute with the specified value.

The following example creates a SAML attribute with the specified name and value.

```
{
  "Rule": "Constant",
  "Name": "Email",
  "Value": "test@user.com"
}
```

If no name is specified, the SAML subject name identifier is assumed.

The following example sets the SAML subject name identifier to the specified value.

```
{
  "Rule": "Constant",
  "Value": "test@user.com "
}
```

Copy Mapping Rule

The Copy mapping rule copies the SAML subject to a SAML attribute, or a SAML attribute to the SAML subject or another SAML attribute.

The following example copies a SAML attribute.

The UserPrincipalName attribute will be copied to Email.

```
{
  "Rule": "Copy",
  "Name": "Email",
  "Value": "UserPrincipalName"
}
```

If no value is specified, the SAML subject name identifier is assumed.

The following example creates a SAML attribute with the specified name whose value is the SAML subject name identifier.

```
{
  "Rule": "Copy",
  "Name": "Email"
}
```

If no name is specified, the SAML subject name identifier is assumed.

The following example sets the SAML subject name identifier to the value of the specified SAML attribute.

```
{
  "Rule": "Copy",
  "Value": "Email"
}
```

```
}
```

Keep Mapping Rule

The Keep mapping rule keeps the specified SAML attributes and removes all others.

The following example keeps a SAML attribute.

The Email attribute is kept, and all other attributes are removed.

```
{
  "Rule": "Keep",
  "Name": "Email"
}
```

Multiple, comma separated SAML attribute names may be specified.

The following example keeps several SAML attributes.

The Email, GivenName and Surname attributes are kept, and all other attributes are removed.

```
{
  "Rule": "Keep",
  "Name": "Email, GivenName, Surname"
}
```

If no name is specified, no SAML attributes are kept.

The following example removes all SAML attributes.

```
{
  "Rule": "Keep"
}
```

Remove Mapping Rule

The Remove mapping rule removes either the SAML subject name identifier or SAML attributes.

The following example removes a SAML attribute.

The Email attribute will be removed.

```
{
  "Rule": "Remove",
  "Name": "Email"
}
```

Multiple, comma separated SAML attribute names may be specified.

The following example removes several SAML attributes.

The Email, GivenName and Surname attributes will be removed.

```
{
  "Rule": "Remove",
  "Name": "Email, GivenName, Surname"
}
```

If no value is specified, the SAML subject name identifier is assumed.

The following example removes the SAML subject name identifier.

```
{
  "Rule": "Remove"
}
```

Rename Mapping Rule

The following example renames a SAML attribute.

The UserPrincipalName attribute will be renamed to Email.

```
{
  "Rule": "Rename",
  "Name": "Email",
  "Value": "UserPrincipalName"
}
```

If no value is specified, the SAML subject name identifier is assumed.

The following example creates a SAML attribute with the specified name whose value is the SAML subject name identifier. The SAML subject name identifier is removed.

```
{
  "Rule": "Rename",
  "Name": "Email"
}
```

If no name is specified, the SAML subject name identifier is assumed.

The following example sets the SAML subject name identifier to the value of the specified SAML attribute. The SAML attribute is removed.

```
{
  "Rule": "Rename",
  "Value": "Email"
}
```

Creating SAML Configuration

The CreateConfiguration console application project may be used to generate SAML configuration.

It may be used to generate SAML configuration for the local identity provider or service provider.

CreateConfiguration may be run as follows.

```
dotnet CreateConfiguration.dll
```

It will prompt for various input required to generate the SAML configuration.

The prompts will vary depending on whether identity provider or service provider configuration is to be generated.

Creating Local Identity Provider Configuration

Create Identity Provider or Service Provider configuration (IdP | SP):

Specify identity provider (IdP) configuration is to be generated.

Name:

Specify a name that uniquely identifies the local identity provider.

For maximum compatibility, a URL is recommended.

For example, it could be the URL of the web site or application although it doesn't necessarily have to point to a web resource.

Single Sign-On Service URL [None]:

Specify the single sign-on service URL.

This is the identity provider endpoint that will receive SAML authn requests.

If SP-initiated SSO will not be supported, this input is not required.

Single Logout Service URL [None]:

Specify the single logout service URL.

This is the identity provider endpoint that will receive SAML logout messages.

If SAML logout will not be supported, this input is not required.

X.509 signature certificate PFX file [None]:

Specify the path to the X.509 certificate file (i.e. PFX file) whose private key will be used for generating signatures.

The identity provider should sign either the SAML response or assertion and so a signature certificate PFX normally is required.

X.509 certificate PFX password [None]:

Specify the password that protects the PFX file.

SAML configuration file [saml.json]:

Specify the file where the generated SAML configuration will be saved.

Creating Local Service Provider Configuration

Create Identity Provider or Service Provider configuration (IdP | SP):

Specify service provider (SP) configuration is to be generated.

Name:

Specify a name that uniquely identifies the local service provider.

For maximum compatibility, a URL is recommended.

For example, it could be the URL of the web site or application although it doesn't necessarily have to point to a web resource.

Assertion Consumer Service URL [None]:

Specify the assertion consumer service URL.

This is the service provider endpoint that will receive SAML responses.

Normally this input should be specified.

Single Logout Service URL [None]:

Specify the single logout service URL.

This is the service provider endpoint that will receive SAML logout messages.

If SAML logout will not be supported, this input is not required.

X.509 signature certificate PFX file [None]:

Specify the path to the X.509 certificate file (i.e. PFX file) whose private key will be used for generating signatures.

If SAML messages will be signed a signature certificate PFX is required.

X.509 certificate PFX password [None]:

Specify the password that protects the PFX file.

SAML configuration file [saml.json]:

Specify the file where the generated SAML configuration will be saved.

SAML Metadata

SAML configuration is different from SAML metadata.

SAML metadata is defined by the SAML v2.0 specification as a standard format for exchanging configuration information between SAML providers.

SAML configuration includes enough information to implement SAML SSO at the local provider.

The SAML for ASP.NET Core Metadata Guide describes how to generate, import and export metadata.