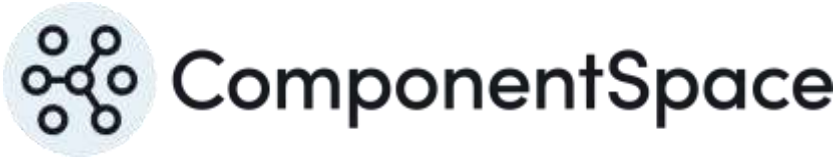

ComponentSpace
SAML for ASP.NET Core
Developer Guide

www.componentspace.com

Contents

Introduction

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

You can use this functionality to easily enable your ASP.NET Core web applications to participate in SAML v2.0 federated single sign-on (SSO) either as an Identity Provider (IdP) or Service Provider (SP).

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

Visual Studio and .NET Core Support

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

- .NET Core v3.1 and above

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

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

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

- Visual Studio 2019 and above

Middleware vs API

When adding SSO support to your application, you have a choice between:

- Adding the SAML authentication handler or SAML middleware

Adding the SAML authentication handler or SAML middleware has the following advantages:

1. Standard model for custom authentication
2. Fewer lines of application code

Explicitly calling the SAML API has the following advantages:

1. More control of SSO and SLO

Both approaches are described in the following sections.

Either approach is valid and, if required, switching between the two is relatively straightforward.

SAML API

Application Startup

In the ConfigureServices method in the application's Startup class, the following occurs:

1. The SAML services are added.
   This includes registering the SAML configuration.
   SAML configuration may be specified as JSON either in the application's appsettings.json or in a separate JSON file (eg saml.json).
   Alternatively, SAML configuration may be stored in a database or some other format and specified programmatically through the SAML configuration API.

```
// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML");
```

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

ISamlIdentityProvider

The ISamlIdentityProvider interface supports SAML single sign-on and logout when acting as the identity provider.

ISamlIdentityProvider extends the ISamlProvider interface.

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

```
public class SamlController : Controller
{
   private readonly ISamlIdentityProvider _samlIdentityProvider;

   public SamlController(ISamlIdentityProvider samlIdentityProvider)
   {
      _samlIdentityProvider = samlIdentityProvider;
   }
}
```

Initiating SSO

The InitiateSsoAsync method initiates single sign-on from the identity provider to the service provider (ie. IdP-initiated SSO).

A SAML response containing a SAML assertion is constructed and sent to the service provider's assertion consumer service.

Prior to calling this method, the user must have been authenticated at the identity provider.

```
Task InitiateSsoAsync(
   string partnerName = null,
   string userID = null,
   IList<SamlAttribute> attributes = null,
   string relayState = null,
   string authnContext = null);
```

partnerName

The partner name corresponds to one of the partner service provider names in the SAML configuration.

For example, if a configured partner service provider is "https://ExampleServiceProvider", this name must be specified in order to initiate SSO to this service provider.

If multiple partner service providers are configured, the partner name must be specified.

userID

The user ID is the primary information identifying the user. It's included as the SAML assertion's subject name identifier (ie. NameID).

For example, the user ID could be their email address.

attributes

SAML attributes are name/value pairs of additional information about the user that are included in the SAML assertion.

For example, the user's first and last name may be included as SAML attributes.

relayState

The relay state specifies the target URL the service provider should redirect to once SSO completes successfully.

Its purpose is to support deep web links.

If not specified, the service provider determines which page to display after SSO completes. Typically, this will be the home page.

authnContext

The authentication context identifies how the user was authenticated.

If not specified, the configured authentication context, if any, is used.

Receiving SSO Requests

The ReceiveSSoAsync method receives a single sign-on request from a service provider (ie. SP-initiated SSO).

A SAML authentication request is received and processed.

The identity provider must authenticate the user and respond to the service provider by calling the SendSsoAsync method.

```
Task<IIdpSsoResult> ReceiveSsoAsync();
```

Sending SSO Responses

The SendSsoAsync method sends a single sign-on response to the service provider (ie. SP-initiated SSO).

A SAML response containing a SAML assertion is constructed and sent to the service provider's assertion consumer service.

Prior to calling this method, ReceiveSSoAsync must have been called and the user authenticated at the identity provider.

```
Task SendSsoAsync(
   string userID = null,
   IList<SamlAttribute> attributes = null,
```

```
    string authnContext = null);
```

userID

The user ID is the primary information identifying the user. It's included as the SAML assertion's subject name identifier (ie. NameID).

For example, the user ID could be their email address.

attributes

SAML attributes are name/value pairs of additional information about the user that are included in the SAML assertion.

For example, the user's first and last name may be included as SAML attributes.

authnContext

The authentication context identifies how the user was authenticated.

If not specified, the configured authentication context, if any, is used.

Sending SSO Error Responses

If an error occurs at the identity provider during SP-initiated SSO, an error status may be returned to the service provider.

```
Task SendSsoAsync(Status status);
```

status

The status is returned in the SAML response to the service provider.

It should provide generic information that doesn't compromise security in any way.

Initiating SLO

The InitiateSloAsync method initiates single logout from the identity provider to all the service providers (ie. IdP-initiated SLO).

A SAML logout request is sent to each service provider to which there is a current SSO session for the user.

```
Task InitiateSloAsync(
    string logoutReason = null,
    string relayState = null);
```

logoutReason

The logout reason identifies why the user logged out.

relayState

The relay state is opaque data that is sent with the logout request and returned with the logout response. It should not be interpreted by the receiving service providers.

Its use is supported but not recommended.

Receiving SLO Messages

The ReceiveSloAsync method receives a single logout request (ie. IdP-initiated SLO) or a single logout response (ie. SP-initiated SLO) from a service provider.

A SAML logout request or response is received and processed.

For a logout request, the identity provider must logout the user and respond to the service provider by calling the SendSloAsync method.

```
Task<ISloResult> ReceiveSloAsync();
```

Sending SLO Responses

The SendSloAsync method sends a single logout response to the service provider (ie. SP-initiated SLO).

A SAML logout response is constructed and sent to the service provider's logout service.

Prior to calling this method, ReceiveSloAsync must have been called and the user logged out at the identity provider.

```
Task SendSloAsync(string errorMessage = null);
```

errorMessage

If specified, the error message indicates why logout failed.

ISamlServiceProvider

The ISamlServiceProvider interface supports SAML single sign-on and logout when acting as the service provider.

ISamlServiceProvider extends the ISamlProvider interface.

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

```
public class SamlController : Controller
{
   private readonly ISamlServiceProvider _samlServiceProvider;

   public HomeController(ISamlServiceProvider samlServiceProvider)
   {
      _samlServiceProvider = samlServiceProvider;
   }
}
```

Initiating SSO

The InitiateSsoAsync method initiates single sign-on from the service provider to the identity provider (ie. SP-initiated SSO).

A SAML authentication request is constructed and sent to the identity provider's SSO service.

```
Task InitiateSsoAsync(
   string partnerName = null,
   string relayState = null),
   ISsoOptions ssoOptions;
```

partnerName

The partner name corresponds to one of the partner identity provider names in the SAML configuration.

For example, if a configured partner identity provider is "https://ExampleIdentityProvider", this name must be specified in order to initiate SSO to this identity provider.

If multiple partner identity providers are configured, the partner name must be specified.

relayState

The relay state is opaque data that is sent with the logout request and returned with the logout response. It should not be interpreted by the receiving service providers.

Its use is supported but not recommended.

ssoOptions

The SSO options are included in the authentication request sent to the identity provider.

Receiving SSO Responses

The ReceiveSSoAsync method receives a single sign-on response from an identity provider (ie. IdP-initiated or SP-initiated SSO).

A SAML response is received and processed. If the SAML response indicates success, it will include a SAML assertion.

The service provider should perform an automatic login of the user using information retrieved from the SAML assertion.

```
Task<ISpSsoResult> ReceiveSsoAsync();
```

Initiating SLO

The InitiateSloAsync method initiates single logout from the service provider to all the identity provider (ie. SP-initiated SLO).

A SAML logout request is sent to the identity provider to which there is a current SSO session for the user.

```
Task InitiateSloAsync(
   string logoutReason = null,
   string relayState = null);
```

partnerName

The partner name corresponds to one of the partner identity provider names in the SAML configuration.

For example, if a configured partner identity provider is “https://ExampleIdentityProvider”, this name must be specified in order to initiate SSO to this identity provider.

If multiple partner identity providers are configured, the partner name must be specified.

logoutReason

The logout reason identifies why the user logged out.

relayState

The relay state is opaque data that is sent with the logout request and returned with the logout response. It should not be interpreted by the receiving identity provider.

Its use is supported but not recommended.

Receiving SLO Messages

The ReceiveSloAsync method receives a single logout request (ie. IdP-initiated SLO) or a single logout response (ie. SP-initiated SLO) from an identity provider.

A SAML logout request or response is received and processed.

For a logout request, the service provider must logout the user and respond to the identity provider by calling the SendSloAsync method.

```
Task<ISloResult> ReceiveSloAsync();
```

Sending SLO Responses

The SendSloAsync method sends a single logout response to the identity provider (ie. IdP-initiated SLO).

A SAML logout response is constructed and sent to the identity provider’s logout service.

Prior to calling this method, ReceiveSloAsync must have been called and the user logged out at the service provider.

```
Task SendSloAsync(string errorMessage = null);
```

errorMessage

If specified, the error message indicates why logout failed.

ISamlProvider

The ISamlProvider interface is the base interface for ISamlIdentityProvider and ISamlServiceProvider.

ISamlProvider extends the IArtifactResolver interface.

Configuration Selection

The configuration name specifies the SAML configuration to use when processing SSO and SLO requests for the current user.

It must match the name of one of the SAML configurations.

It's only required when there are multiple SAML configurations.

Typically, it's used in multi-tenant applications where each tenant has a separate SAML configuration.

There are no concurrency issues setting the configuration name for multiple users as it's stored in the SAML SSO session state.

```
Task SetConfigurationNameAsync(string configurationName);
```

configurationName

The configuration name specifies which SAML configuration to use when processing SSO and SLO requests for the current user.

It's only required if multiple SAML configurations exist (e.g., in a multi-tenanted environment).

SSO Status

The GetStatusAsync method returns the current SSO status for the user.

```
Task<ISsoStatus> GetStatusAsync();
```

Peeking Message Types

The PeekMessageTypeAsync method returns the type of the SAML message currently being received.

The recommendation is to use separate endpoints for different message types.

However, if a single endpoint is being used to receive all SAML messages, the message type may be peeked to determine how to continue with the processing of the message.

```
Task<SamlMessageType> PeekMessageTypeAsync();
```

IArtifactResolver

The IArtifactResolver interface supports the HTTP-Artifact binding and the processing of SAML artifact resolve requests.

Artifact Resolution

The ResolveArtifactAsync method receives a SAML artifact resolve request and returns a SAML artifact response.

```
Task ResolveArtifactAsync();
```

IIdPSsoResult

The IIdPSsoResult interface returns the result of SP-initiated SSO at the identity provider.

PartnerName

The partner name identifies the service provider that initiated SSO.

It's retrieved from the issuer field of the SAML authentication request.

It's provided in case the application has service provider specific processing to perform.

SsoOptions

The SSO options specifies options associated with the SSO.

These may, for example, affect how the user is authenticated.

ISPSsoResult

The ISPSsoResult interface returns the result of IdP-initiated or SP-initiated SSO at the service provider.

PartnerName

The partner name identifies the identity provider that initiated or responded to SSO.

It's retrieved from the issuer field of the SAML response.

It's provided in case the application has identity provider specific processing to perform.

InResponseTo

The InResponseTo flag indicates whether the SSO result is in response to a previous SSO request.

For IdP-initiated SSO, the flag is false. For SP-initiated SSO, the flag is true.

UserID

The user ID is the primary information identifying the user. It's retrieved from the SAML assertion's subject name identifier.

Attributes

SAML attributes are name/value pairs of additional information about the user that are retrieved from the SAML assertion.

AuthnContext

The authentication context identifies the method by which the user was authenticated. It's retrieved from the SAML assertion's authentication statement.

RelayState

The relay state is opaque data that is sent with the authentication request and returned with the SAML response.

Its use is supported but not recommended.

ISsoOptions

The ISsoOptions interface encapsulates various options associated with SSO.

RequestedUserName

The request user name identifies the user to be authenticated.

It's included in the SAML authentication request.

The default is none.

ForceAuthn

The ForceAuthn flag requests that the identity provider discards any existing user authentication session and establish a new user authentication session.

It's included in the SAML authentication request.

The default is false.

IsPassive

The IsPassive flag requests that the identity provider not visibly take control of the user interface.

It's included in the SAML authentication request.

The default is false.

AllowCreate

The AllowCreate flag indicates whether the identity provider is allowed to create a new user as part of processing the request.

It's included in the SAML authentication request.

The default is true.

ProviderName

The provider name is the human readable name of the requesting SAML provider.

It's included in the SAML authentication request.

The default is none.

SPNameQualifier

The service provider name qualifier specifies that the assertion subject's identifier be returned in the namespace of a service provider other than the requester.

It's included in the SAML authentication request.

The default is none.

RequestedAuthnContexts

The requested authentication contexts place requirements on the authentication process at the identity provider. For example, it may request multi-factor authentication of users.

They're included in the SAML authentication request.

The default is none.

RequestedAuthnContextComparison [optional]

The comparison method is used to evaluate the requested contexts.

It's included in the SAML authentication request.

The comparison methods are:

- exact
- minimum
- maximum
- better

The default is exact.

TrustedIdentityProviders

The trusted identity providers are included as scoping.

Scoping is included in the SAML authentication request.

The default is none.

Destination

The destination specifies the address to which the SAML authentication request is sent.

It's included in the SAML authentication request.

The default is the configured single sign-on service URL.

ITrustedIdentityProvider

The ITrustedIdentityProvider interface specifies scoping information.

ProviderID

The provider ID identifies the SAML provider by its ID.

Name [optional]

The name is the human readable name of the SAML provider.

ISsoStatus

The ISsoStatus interface encapsulates the SSO status for the user.

IsSso returns true if the local provider is currently single signed-on with any partner provider.

```
public bool IsSso()
```

IsSso returns true if the local provider is currently single signed-on with the specified partner provider.

```
public bool IsSso(string partnerName)
```

IsSsoCompletionPending returns true if completion of single sign-on is pending.

SSO is pending if the service provider has sent an authentication request but the identity provider hasn't sent the SAML response.

```
public bool IsSsoCompletionPending()
```

IsSsoCompletionPending returns true if completion of single sign-on is pending with the specified partner provider.

SSO is pending if the service provider has sent an authentication request but the identity provider hasn't sent the SAML response.

```
public bool IsSsoCompletionPending(string partnerName)
```

IsSloCompletionPending returns true if completion of single logout is pending.

SLO is pending if a logout request has been received but a logout response hasn't been sent or a logout request has been sent but a logout response hasn't been received.

```
public bool IsSloCompletionPending()
```

IsSloCompletionPending returns true if completion of single logout is pending with the specified partner provider.

SLO is pending if a logout request has been received but a logout response hasn't been sent or a logout request has been sent but a logout response hasn't been received.

```
public bool IsSloCompletionPending(string partnerName)
```

CanSloAsync returns true if one or more partner providers have successfully completed single sign-on and also support logout.

```
public Task<bool> CanSloAsync()
```

CanSloAsync returns true if the specified partner provider has successfully completed single sign-on and also support logout.

```
public Task<bool> CanSloAsync(string partnerName)
```

ILicense

The ILicense interface returns licensing information.

Name

The product name identifies the SAML assembly.

Version

The version returns the SAML assembly version information.

IsLicensed

IsLicensed returns true if the product is licensed.

It returns false if the product is a time-limited evaluation version.

Only licensed versions of the product should be used in production.

Expires

Expires returns the date the evaluation license expires.

SAML Middleware

SAML Identity Provider Middleware

The SAML middleware provides SSO support for identity provider applications.

In the ConfigureServices method in the application's Startup class, the following occurs:

1. The SAML services are added.
   This includes registering the SAML configuration.
   SAML configuration may be specified as JSON either in the application's appsettings.json or in a separate JSON file (eg saml.json).
   Alternatively, SAML configuration may be stored in a database or some other format and specified programmatically through the SAML configuration API.

2. The SAML middleware is added.
   The options specify SAML middleware configuration.

```
// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));

// Add the SAML middleware services.
services.AddSamlMiddleware(options =>
{
   options.PartnerName = (httpContext) => Configuration["PartnerName"];
});
```

In the Configure method in the application's Startup class, the following occurs:

1. Authentication is enabled.

2. The SAML middleware is enabled.

```
app.UseAuthentication();

// Use SAML middleware.
app.UseSaml();
```

The SAML middleware supports both IdP-initiated and SP-initiated SSO and SLO.

To initiate SSO from the IdP, redirect to InitiateSingleSignOnPath.

```
public ActionResult SingleSignOn()
{
   return Redirect(SamlMiddlewareDefaults.InitiateSingleSignOnPath);
}
```

To initiate SLO from the IdP, redirect to InitiateSingleLogoutPath.

In this example, the returnUrl distinguishes between IdP-initiated (i.e. no returnUrl) and SP-initiated SLO (i.e. a returnUrl).

```
public async Task<IActionResult> Logout(string returnUrl)
{
   await _signInManager.SignOutAsync();
   _logger.LogInformation("User logged out.");

   if (string.IsNullOrEmpty(returnUrl))
   {
      returnUrl = SamlMiddlewareDefaults.InitiateSingleLogoutPath;
   }

   return Redirect(returnUrl);
}
```

A SAML controller or any SAML API calls within the application are not required as SSO and SLO are managed by the SAML middleware.

SAML Middleware Options

The SamlMiddlewareOptions class supplies options to the SAML middleware.

ConfigurationName

The optional delegate returns the configuration name identifying which SAML configuration to use when processing SSO and SLO requests for the current user.

It's only required if multiple SAML configurations exist (e.g., in a multi-tenanted environment).

PartnerName

The optional delegate returns the name of one of the partner service providers in the SAML configuration.

For example, if a configured partner service provider is “https://ExampleServiceProvider”, this name must be specified in order to initiate SSO to this service provider.

If multiple partner service providers are configured, the partner name must be specified.

RelayState

The optional delegate returns the relay state to include when initiating SSO to the partner service provider.

The relay state specifies the target URL the service provider should redirect to once SSO completes successfully.

Its purpose is to support deep web links.

If not specified, the service provider determines which page to display after SSO completes. Typically, this will be the home page.

InitiateSingleSignOnPath

The initiate single sign-on path is the endpoint within the SAML middleware that initiates SSO.

The application may redirect to this endpoint for IdP-initiated SSO.

The default is “/SAML/InitiateSingleSignOn”.

InitiateSingleLogoutPath

The initiate single logout path is the endpoint within the SAML middleware that initiates SLO.

The application may redirect to this endpoint for IdP-initiated SLO.

The default is “/SAML/InitiateSingleLogout”.

SingleSignOnServicePath

The single sign-on service path is the endpoint within the SAML middleware that receives SAML authn requests from partner service providers.

The default is “/SAML/SingleSignOnService”.

SingleSignOnServiceCompletionPath

The single sign-on service completion path is the endpoint within the SAML middleware that completes SSO.

The application redirects to this endpoint once the used is authenticated.

The default is “/SAML/SingleSignOnServiceCompletion”.

SingleLogoutServicePath

The single logout service path is the endpoint within the SAML middleware that receives SAML logout messages from partner service providers.

The default is "/SAML/SingleLogoutService".

SingleLogoutServiceCompletionPath

The single logout service completion path is the endpoint within the SAML middleware that completes SLO.

The application redirects to this endpoint once the used is logged off.

The default is "/SAML/SingleLogoutServiceCompletion".

ArtifactResolutionServicePath

The artifact resolution service path is the endpoint within the SAML authentication handler that receives SAML artifact resolutions messages from partner service providers.

The default is "/SAML/ArtifactResolutionService".

LoginUrl

The login URL is redirected to as part of SSO if the user isn't authenticated.

The application should login the user.

The default is "/Identity/Account/Login".

LogoutUrl

The logout URL is redirected to as part of SLO if the user is authenticated.

The application should logout the user.

The default is "/Identity/Account/Logout".

LogoutCompletionUrl

The logout completion URL is redirected to once SLO completes and if this URL hasn't been specified using a ReturnUrl query string.

The default is "/".

ErrorUrl

The optional delegate returns the URL to redirect if SSO or SLO fails.

The default is "/Error".

ConfigurationNameParameter

The optional configuration name query string parameter specifies the configuration name identifying which SAML configuration to use when processing SSO and SLO requests for the current user.

It's only required if multiple SAML configurations exist (e.g., in a multi-tenanted environment).

This is an alternative to the ConfigurationName delegate.

The default is "ConfigurationName".

PartnerNameParameter

The optional partner name query string parameter specifies the name of one of the partner service providers in the SAML configuration.

For example, if a configured partner service provider is “https://ExampleServiceProvider”, this name must be specified in order to initiate SSO to this service provider.

If multiple partner service providers are configured, the partner name must be specified.

This is an alternative to the PartnerName delegate.

The default is “PartnerName”.

RelayStateParameter

The optional relay state query string parameter specifies the relay state to include when initiating SSO to the partner service provider.

The relay state specifies the target URL the service provider should redirect to once SSO completes successfully.

Its purpose is to support deep web links.

If not specified, the service provider determines which page to display after SSO completes. Typically, this will be the home page.

This is an alternative to the RelayState delegate.

The default is “RelayState”.

ReturnUrlParameter

The return URL query string parameter specifies the URL to return to once SSO or SLO completes.

The default is “ReturnUrl”.

Events

The optional event delegates.

SAML Authentication Handler

The SAML authentication handler provides SSO support for service provider applications.

In the ConfigureServices method in the application’s Startup class, the following occurs:

1. The SAML services are added.
   This includes registering the SAML configuration.
   SAML configuration may be specified as JSON either in the application’s appsettings.json or in a separate JSON file (eg saml.json).
   Alternatively, SAML configuration may be stored in a database or some other format and specified programmatically through the SAML configuration API.

2. The SAML authentication handler is added.
   The options specify SAML authentication handler configuration.

```
// Add SAML SSO services.
```

```
services.AddSaml(Configuration.GetSection("SAML"));

// Add SAML authentication services.
services.AddAuthentication().AddSaml(options =>
{
   options.PartnerName = (httpContext) => Configuration["PartnerName"];
});
```

In the Configure method in the application's Startup class, the following occurs:

1. Authentication is enabled.

```
app.UseAuthentication();
```

The SAML authentication handler supports both IdP-initiated and SP-initiated SSO and SLO.

The SAML authentication handler will accept challenge requests from the application as part of login as well as sign out requests as part of logout.

Upon a challenge request, a SAML authn request is sent to the identity provider as part of SP-initiated SSO. The SAML response returned by the identity provider is automatically received and processed to login the user.

Upon a sign out request, a SAML logout request is sent to the identity provider as part of SP-initiated SLO. The SAML logout response returned by the identity provider is automatically received and processed to logout the user.

A SAML controller or any SAML API calls within the application are not required as SSO and SLO are managed by the SAML authentication handler.

SAML Authentication Options

The SamlAuthenticationOptions class supplies options to the SAML authentication handler.

ConfigurationName

The optional delegate returns the configuration name identifying which SAML configuration to use when processing SSO and SLO requests for the current user.

It's only required if multiple SAML configurations exist (e.g., in a multi-tenanted environment).

PartnerName

The optional delegate returns the name of one of the partner identity providers in the SAML configuration.

For example, if a configured partner identity provider is "https://ExampleIdentityProvider", this name must be specified to initiate SSO to this identity provider.

If multiple partner identity providers are configured, the partner name must be specified.

AssertionConsumerServicePath

The assertion consumer service path is the endpoint within the SAML authentication handler that receives SAML responses from partner identity providers.

The default is “/SAML/AssertionConsumerService”.

SingleLogoutServicePath

The single logout service path is the endpoint within the SAML authentication handler that receives SAML logout messages from partner identity providers.

The default is “/SAML/SingleLogoutService”.

ArtifactResolutionServicePath

The artifact resolution service path is the endpoint within the SAML authentication handler that receives SAML artifact resolutions messages from partner identity providers.

The default is “/SAML/ArtifactResolutionService”.

LoginCompletionUrl

The optional delegate returns the URL to redirect to once SSO completes and if this URL hasn’t been specified using a ReturnUrl query string.

The default is “/Identity/Account/ExternalLogin?handler=Callback”.

LogoutCompletionUrl

The optional delegate returns the URL to redirect to once SLO completes and if this URL hasn’t been specified using a ReturnUrl query string.

The default is “/”.

ErrorUrl

The optional delegate returns the URL to redirect if SSO or SLO fails.

The default is “/Error”.

SignInScheme

The sign-in scheme is responsible for persisting user's identity after successful authentication.

The default is “Identity.External”.

SignOutScheme

The signout scheme is responsible for signout. If not specified, the SignInScheme is used.

The default is “Identity.Application”.

Events

The optional event delegates.

Authentication Properties

The SAML authentication handler recognizes the following AuthenticationProperties items during login and logout.

ConfigurationName

The optional configuration name identifies which SAML configuration to use when processing SSO and SLO requests for the current user.

It's only required if multiple SAML configurations exist (e.g., in a multi-tenanted environment).

This is an alternative to the ConfigurationName delegate.

PartnerName

The optional partner name specifies the name of one of the partner identity providers in the SAML configuration.

For example, if a configured partner identity provider is "https://ExampleIdentityProvider", this name must be specified in order to initiate SSO to this identity provider.

If multiple partner identity providers are configured, the partner name must be specified.

This is an alternative to the PartnerName delegate.

Error Handling

If an error occurs an exception is thrown.

For example, if an XML signature cannot be verified, a SamlSignatureException is thrown.

It's recommended that applications not attempt to process the various types of exception separately.

Instead, if any exception occurs, the error should be logged and either the user presented with a generic error page requesting they try again or the application automatically attempts the operation again but protected by a retry limit.

SAML Events

A number of call back delegates are defined that enable SAML protocol messages and SAML assertions to be accessed and optionally updated when being created or accessed when being received.

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

ISamlIdentityProvider Events

OnAuthnRequestReceived is called when a SAML authentication request is received from a service provider.

```
Action<HttpContext, AuthnRequest, string> OnAuthnRequestReceived { get; set; }
```

OnSamlAssertionCreated is called when a SAML assertion has been created.

The SAML assertion may be modified if required.

```
Func<HttpContext, SamlAssertion, SamlAssertion> OnSamlAssertionCreated { get; set; }
```

OnSamlResponseCreated is called when a SAML response has been created.

The SAML response may be modified if required.

```
Func<HttpContext, SamlResponse, SamlResponse> OnSamlResponseCreated { get; set; }
```

ISamlServiceProvider Events

OnAuthnRequestCreated is called when a SAML authentication request has been created.

The SAML authentication request may be modified if required.

```
Func<HttpContext, AuthnRequest, AuthnRequest> OnAuthnRequestCreated { get; set; }
```

OnSamlResponseReceived is called when a SAML response is received from an identity provider.

```
Action<HttpContext, SamlResponse, string> OnSamlResponseReceived { get; set; }
```

OnSamlAssertionReceived is called when a SAML assertion is received from an identity provider.

```
Action<HttpContext, SamlAssertion> OnSamlAssertionReceived { get; set; }
```

ISamlProvider Events

OnResolveUrl is called to resolve the destination URL when sending a SAML message.

The URL may be modified if required.

```
Func<HttpContext, SamlEndpointType, string, string> OnResolveUrl { get; set; }
```

OnSendMessage is called when sending a SAML message.

The SAML message may be modified if required.

```
Func<HttpContext, XmlElement, XmlElement> OnSendMessage { get; set; }
```

OnReceiveMessage is called when receiving a SAML message.

The SAML message may be modified if required.

```
Func<HttpContext, XmlElement, XmlElement> OnReceiveMessage { get; set; }
```

OnLogoutRequestCreated is called when a SAML logout request has been created.

The SAML logout request may be modified if required.

```
Func<HttpContext, LogoutRequest, LogoutRequest> OnLogoutRequestCreated { get; set; }
```

OnLogoutResponseCreated is called when a SAML logout response has been created.

The SAML logout response may be modified if required.

```
Func<HttpContext, LogoutResponse, LogoutResponse> OnLogoutResponseCreated { get; set; }
```

OnLogoutRequestReceived is called when a SAML logout request is received.

```
Action<HttpContext, LogoutRequest, string> OnLogoutRequestReceived { get; set; }
```

OnLogoutResponseReceived is called when a SAML logout response is received.

```
Action<HttpContext, LogoutResponse, string> OnLogoutResponseReceived { get; set; }
```

OnArtifactResolveCreated is called when a SAML artifact resolve request is created.

```
Func<HttpContext, ArtifactResolve, ArtifactResolve> OnArtifactResolveCreated { get; set; }
```

OnArtifactResponseCreated is called when a SAML artifact response is created.

```
Func<HttpContext, ArtifactResponse, ArtifactResponse> OnArtifactResponseCreated { get; set; }
```

OnArtifactResolveReceived is called when a SAML artifact resolve request is received.

```
Action<HttpContext, ArtifactResolve> OnArtifactResolveReceived { get; set; }
```

OnArtifactResponseReceived is called when a SAML artifact response is received.

```
Action<HttpContext, ArtifactResponse> OnArtifactResponseReceived { get; set; }
```

Event Examples

The following example demonstrates including SAML extensions in a SAML authentication request.

Once the authentication request is created, the delegate is called.

The delegate updates the authentication request by including some SAML extensions.

The updated authentication request is then sent to the identity provider.

```
// Include SAML extensions in the authn request.
```

```
_samlServiceProvider.Events.OnAuthnRequestCreated += (httpContext, authnRequest) =>
{
   var xmlDocument = new XmlDocument();
   xmlDocument.LoadXml("<test xmlns=\"urn:test\">This is a test</test>");

   authnRequest.Extensions = new Extensions()
   {
      Items = new List<XmlElement>() { xmlDocument.DocumentElement }
   };

   return authnRequest;
};

// To login automatically at the service provider, initiate single sign-on to the identity provider (SP-initiated SSO).
var partnerName = _configuration["PartnerName"];

await _samlServiceProvider.InitiateSsoAsync(partnerName);
```

The following example demonstrates receiving SAML extensions in a SAML authentication request.

Once the authentication request is received, the delegate is called.

The SAML extensions are retrieved for subsequent processing.

```
// Receive the SAML extensions included in the authn request.
IEnumerable<XmlElement> extensionsItems = null;

_samlIdentityProvider.Events.OnAuthnRequestReceived += (authnRequest, relayState) =>
{
   extensionsItems = authnRequest.Extensions?.Items;
};

// Receive the authn request from the service provider (SP-initiated SSO).
await _samlIdentityProvider.ReceiveSsoAsync();
```

The following example demonstrates adding a query string to the single sign-on service URL.

```
_samlServiceProvider.Events.OnResolveUrl += (httpContext, samlEndpointType, url) =>
{
   return QueryHelpers.AddQueryString(url, "username", "joeuser@componentspace.com");
};

await _samlServiceProvider.InitiateSsoAsync(partnerName, returnUrl);
```

The following example demonstrates adding advice to the SAML assertion.

```
// Add advice to the SAML assertion.
```

```
_samlIdentityProvider.Events.OnSamlAssertionCreated += (httpContext, samlAssertion) =>
{
   samlAssertion.Advice = new Advice()
   {
      AdviceList = new List<AdviceListItem>()
      {
         new AdviceListItem()
         {
            SamlAssertion = new SamlAssertion()
            {
               Issuer = samlAssertion.Issuer,
               Subject = new Subject()
               {
                  NameID = new NameID()
                  {
                     Name = "joeuser@componentspace.com"
                  }
               }
            }
         }
      }
   };

   return samlAssertion;
};

return _samlIdentityProvider.SendSsoAsync(userName, attributes);
```

The following example demonstrates adding a name qualifier to the SAML assertion.

```
// Add a name qualifier to the SAML assertion.
_samlIdentityProvider.Events.OnSamlAssertionCreated += (httpContext, samlAssertion) =>
{
   samlAssertion.Subject.NameID.NameQualifier = "example.com";

   return samlAssertion;
};

return _samlIdentityProvider.SendSsoAsync(userName, attributes);
```

SAML Middleware Events

In addition to the ISamlIdentityProvider events, the SAML middleware supports the following events.

OnInitiateSso is called prior to calling InitiateSsoAsync.

The InitiateSsoAsync parameters may be modified if required.

```
Func<HttpContext, string, string, IList<SamlAttribute>, string, string, (string, string,
IList<SamlAttribute>, string, string)> OnInitiateSso { get; set; }
```

OnSendSso is called prior to calling SendSsoAsync.

The SendSsoAsync parameters may be modified if required.

```
Func<HttpContext, string, IList<SamlAttribute>, string, (string, IList<SamlAttribute>, string, Status)> OnSendSso { get; set; }
```

OnInitiateSlo is called prior to calling InitiateSloAsync.

The InitiateSloAsync parameters may be modified if required.

```
Func<HttpContext, string, string, (string, string)> OnInitiateSlo { get; set; }
```

OnSendSlo is called prior to calling SendSloAsync.

The SendSloAsync parameters may be modified if required.

```
Func<HttpContext, string, string> OnSendSlo { get; set; }
```

OnError is called when an error occurs during SAML SSO or SLO.

```
Func<HttpContext, Exception, bool> OnError { get; set; }
```

The error details are supplied in the exception. The delegate should return true if it has processed the error and no further action is required by the SAML middleware.

SAML Middleware Event Examples

The following example demonstrates setting the authentication context prior to the call to InitiateSsoAsync.

```
// Add SAML middleware services.
services.AddSamlMiddleware(options =>
{
   options.Events.OnInitiateSso = (httpContext, partnerName, userID, attributes, relayState, authnContext) =>
   {
      return (partnerName, userID, attributes, relayState, AuthnContextClasses.Password);
   };
});
```

The following example demonstrates receiving SAML extensions in a SAML authentication request.

```
// Add the SAML middleware services.
services.AddSamlMiddleware(options =>
```

```
{
   options.Events.OnAuthnRequestReceived = (httpContext, authnRequest, relayState) =>
   {
      var extensionsItems = authnRequest.Extensions?.Items; ;
   };
});
```

The following example redirects to a special error handling page.

```
// Add the SAML middleware services.
services.AddSamlMiddleware(options =>
{
   options.Events.OnError = (httpContext, exception) =>
   {
      httpContext.Response.Redirect("/Error");

      return true;
   };
});
```

SAML Authentication Handler Events

In addition to the ISamlServiceProvider events, the SAML authentication handler supports the following events.

OnInitiateSso is called prior to calling InitiateSsoAsync.

The InitiateSsoAsync parameters may be modified if required.

```
Func<HttpContext, string, string, ISsoOptions, (string, string, ISsoOptions)> OnInitiateSso
{ get; set; }
```

OnInitiateSlo is called prior to calling InitiateSloAsync.

The InitiateSloAsync parameters may be modified if required.

```
Func<HttpContext, string, string, string, (string, string, string)> OnInitiateSlo { get; set; }
```

OnSendSlo is called prior to calling SendSloAsync.

The SendSloAsync parameters may be modified if required.

```
Func<HttpContext, string, string> OnSendSlo { get; set; }
```

OnError is called when an error occurs during SAML SSO or SLO.

```
Func<HttpContext, Exception, bool> OnError { get; set; }
```

The error details are supplied in the exception. The delegate should return true if it has processed the error and no further action is required by the SAML authentication handler.

SAML Authentication Handler Event Examples

The following example demonstrates setting SSO options prior to the call to InitiateSsoAsync.

```
// Add SAML authentication services.
services.AddAuthentication().AddSaml(options =>
{
   options.Events = new SamlAuthenticationEvents
   {
      OnInitiateSso = (httpContext, partnerName, relayState, ssoOptions) =>
      {
         ssoOptions = new SsoOptions
         {
            RequestedUserName = "joe@componentspace.com"
         };

         return (partnerName, relayState, ssoOptions);
      }
   };
});
```

The following example demonstrates including SAML extensions in a SAML authentication request.

```
// Add SAML authentication services.
services.AddAuthentication().AddSaml(options =>
{
   options.Events = new SamlAuthenticationEvents
   {
      OnAuthnRequestCreated = (httpContext, authnRequest) =>
      {
         var xmlDocument = new XmlDocument();
         xmlDocument.LoadXml("<test xmlns=\"urn:test\">This is a test</test>");

         authnRequest.Extensions = new Extensions()
         {
            Items = new List<XmlElement>() { xmlDocument.DocumentElement }
         };

         return authnRequest;
      }
   };
});
```

The following example also demonstrates including SAML extensions in a SAML authentication request. However, the EventsType rather than the Events property is set. This is useful if other services are to be made available through dependency injection.

```
using ComponentSpace.Saml2.Authentication;

public class MySamlAuthenticationEvents : SamlAuthenticationEvents
{
   public MySamlAuthenticationEvents()
   {
      OnAuthnRequestCreated = (httpContext, authnRequest) =>
      {
         var xmlDocument = new XmlDocument();
         xmlDocument.LoadXml("<test xmlns=\"urn:test\">This is a test</test>");

         authnRequest.Extensions = new Extensions()
         {
            Items = new List<XmlElement>() { xmlDocument.DocumentElement }
         };

         return authnRequest;
      }
   }
}

// Add the authentication events handler.
services.AddTransient<MySamlAuthenticationEvents>();

// Add SAML authentication services.
services.AddAuthentication().AddSaml(options =>
{
   options.EventsType = typeof(MySamlAuthenticationEvents);
});
```

The following example redirects to a special error handling page if an AuthnFailed error status is returned by the identity provider. All other errors are handled by the authentication handler.

```
// Add SAML authentication services.
services.AddAuthentication().AddSaml(options =>
{
   options.PartnerName = () => Configuration["PartnerName"];
   options.Events = new SamlAuthenticationEvents
   {
      OnError = (httpContext, exception) =>
      {
         if (!(exception is SamlErrorStatusException))
         {
            return false;
         }
```

```
        var samlErrorStatusException = exception as SamlErrorStatusException;

        if (samlErrorStatusException?.Status?.StatusCode?.Code !=
SamlConstants.PrimaryStatusCodes.Responder ||
            samlErrorStatusException?.Status?.StatusCode?.SubordinateStatusCode?.Code !=
SamlConstants.SecondaryStatusCodes.AuthnFailed)
        {
            return false;
        }

        httpContext.Response.Redirect("/AuthnFailedError");

        return true;
      }
    };
});
```

Customizations

A number of 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, AddSaml, adds the various default implementations of the SAML interfaces to the .NET Core default services container. This should be called in the ConfigureServices method of the application's Startup 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 SAML interfaces except it replaces the ISsoSessionStore with a custom implementation.

```
// Add SAML SSO services.
services.AddSaml();

// Add a custom SSO session store.
services.AddScoped<ISsoSessionStore, CustomSsoSessionStore>();
```

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 Core default services container isn't being used.

Interface	Implementation	Lifetime
-	IOptionsMonitor<CertificateCacheOptions>	Transient
-	IOptionsMonitor<CertificateValidationOptions>	Transient
-	IOptionsMonitor<CookieSsoSessionStoreOptions>	Transient
-	IOptionsMonitor<DistributedSsoSessionStoreOptions>	Transient

-	IOptionsMonitor<HttpPostFormOptions>	Transient
-	IOptionsMonitor<SamlCachedConfigurationResolverOptions>	Transient
-	IOptionsMonitor<SamlConfigurations>	Transient
-	IOptionsMonitor<SamlSchemaValidatorOptions>	Transient
IArtifactCache	DistributedArtifactCache	Transient
ICertificateImporter	CertificateFileImporter, CertificateStringImporter	Transient
ICertificateLoader	CachedCertificateLoader	Transient
ICertificateManager	CertificateManager	Transient
ICertificateValidator	CertificateValidator	Transient
IConfigurationToMetadata	IConfigurationToMetadata	Transient
IDistributedCache	MemoryDistributedCache	Singleton
IHttpArtifactBinding	HttpArtifactBinding	Transient
IHttpContextAccessor	HttpContextAccessor	Transient
IHttpPostBinding	HttpPostBinding	Transient
IHttpPostForm	HttpPostForm	Transient
IHttpRedirectBinding	HttpRedirectBinding	Transient
IHttpRequest	AspNetHttpRequest	Transient
IHttpResponse	AspNetHttpResponse	Transient
IIDCache	DistributedIDCache	Transient
ILicense	License	Transient
ILoggerFactory	LoggerFactory	Transient
IMetadataLoader	MetadataLoader	Transient
IMiddleware	SamlMiddleware	Transient
ISamlClaimFactory	SamlClaimFactory	Transient
ISamlConfigurationResolver	SamlCachedConfigurationResolver	Transient
ISamlConfigurationResolver	SamlConfigurationResolver	Transient
ISamlIdentityProvider	SamlIdentityProvider	Transient
ISamlSchemaValidator	SamlSchemaValidator	Transient
ISamlServiceProvider	SamlServiceProvider	Transient
ISoapBinding	SoapBinding	Transient
ISsoSessionStore	DistributedSsoSessionStore	Scoped
IUrlUtility	UrlUtility	Transient
IXmlDataEncryptionExtension	TripleDesXmlDataEncryptionExtension	Transient
IXmlDataEncryptionExtension	AesXmlDataEncryptionExtension	Transient
IXmlDataEncryptionExtension	AesGcmXmlDataEncryptionExtension	Transient
IXmlEncryption	XmlEncryption	Transient
IXmlKeyEncryptionExtension	RsaXmlKeyEncryptionExtension	Transient
IXmlKeyEncryptionExtension	RsaOaepMgf1pXmlKeyEncryptionExtension	Transient
IXmlSignature	XmlSignature	Transient

ISsoSessionStore

The ISsoSessionStore interface supports storing SAML SSO session data. It's used by the identity provider and service provider to support the SAML protocol.

Several implementations are provided.

Distributed Cache Session Store

A default implementation stores SSO session data in an IDistributedCache.

The implementation of IDistributedCache may be specified through dependency injection.

The default implementation of IDistributedCache caches to memory.

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 SAML for ASP.NET Core Web Farm Guide.

Distributed Cache Session Store Cookie

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

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

An example set-cookie response header is shown below.

```
set-cookie: saml-session=f0225c85-20c5-4535-a289-4b173ff23e4a; path=/; secure;
samesite=none; httponly
```

SameSite Cookie Considerations

A set-cookie header may include an optional SameSite attribute whose purpose is to help protect against cross-site request forgery attacks (CSRF).

SAML protocol exchanges are, in most use cases, cross-site. The identity provider (IdP) and service provider (SP) are different sites. Furthermore, these flows do not involve users clicking navigation links from one site to the other. For example, when an IdP sends an SP a SAML response, it returns a 200 HTTP response to the browser containing an HTML form and some JavaScript to automatically submit the form to the SP via an HTTP Post. From the browser's perspective, the current site is the IdP and destination site for the HTTP Post is the SP.

If the SAML session cookie is marked as SameSite=Strict, the browser won't include it with the SAML response as the sites are different. If the SAML session cookie is marked as SameSite=Lax, the browser still won't include it as this isn't considered a top-level navigation action. Specifically, the SameSite specification doesn't consider Post to be a safe HTTP method.

Consequently, the SAML 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 SAML session cookie will take on this minimum setting. This will mean the browser won't return the cookie and the corresponding SAML 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.

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

Note that these issues aren't specific to SAML SSO or ASP.NET Core. Other external authentication protocols and other platforms potentially have the same issues.

One other point to note is that no cookies, not just the SAML session cookie, marked as Strict or Lax will be included with the HTTP Post. This may impact other aspects of the site's functionality.

However, a subsequent redirect by the site back to itself will make these cookies available as this will no longer be cross-site.

Distributed Cache Session Store Options

Options associated with the distributed cache session store may be set.

For example, the following code in the ConfigureServices specifies the name of the SAML cookie.

```
using ComponentSpace.Saml2.Bindings;
using ComponentSpace.Saml2.Configuration;
using ComponentSpace.Saml2.Session;

services.Configure<DistributedSsoSessionStoreOptions>(options =>
{
   options.CookieName = "my-saml-session";
});

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

Alternatively, the options may be specified through configuration.

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

```
"DistributedSsoSessionStore": {
  "CookieName": "my-saml-session"
}
```

The following code in the ConfigureServices makes use of this configuration.

```
using ComponentSpace.Saml2.Configuration;
using ComponentSpace.Saml2.Session;

// Configure the distributed cache SSO session store.
services.Configure<DistributedSsoSessionStoreOptions>(
   Configuration.GetSection("DistributedSsoSessionStore"));

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

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

```
using ComponentSpace.Saml2.Bindings;
using ComponentSpace.Saml2.Configuration;
using ComponentSpace.Saml2.Session;

services.Configure<DistributedSsoSessionStoreOptions>(options =>
```

```
{
   options.CookieOptions.Domain = "componentspce.com";
});

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

ASP.NET Session Store

An alternative implementation stores SSO session data in the ASP.NET session. This requires ASP.NET sessions to be enabled.

For example, the following code in the ConfigureServices makes use of the default implementations of the SAML interfaces except it replaces the ISsoSessionStore with the ASP.NET session store.

```
// Add ASP.NET sessions.
services.AddSession(options =>
{
   options.Cookie.Name = "ExampleIdentityProvider";
});

// Add SAML SSO services.
services.AddSaml();

// Add the ASP.NET SSO session store.
services.AddScoped<ISsoSessionStore, AspNetSsoSessionStore>();
```

And in the Configure method of the application's Startup class, enable ASP.NET sessions.

```
app.UseSession();
```

Cookie Session Store

An alternative implementation stores SSO session data in a cookie.

For example, the following code in the ConfigureServices makes use of the default implementations of the SAML interfaces except it replaces the ISsoSessionStore with the cookie session store.

```
// Add SAML SSO services.
services.AddSaml();

// Add the cookie SSO session store.
services.AddScoped<ISsoSessionStore, CookieSsoSessionStore>();
```

The cookie value is encrypted using Microsoft's IDataProtector interface accessed through the IDataProtectionProvider interface.

For more information, refer to:

https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/introduction

In a web farm deployment, cipher keys may be stored in a central location such as a Redis cache.

For more information, refer to:

https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/implementation/key-storage-providers

Cookie Session Store Options

Options associated with the cookie session store may be set.

For example, the following code in the ConfigureServices specifies the name of the SAML cookie.

```
using ComponentSpace.Saml2.Bindings;
using ComponentSpace.Saml2.Configuration;
using ComponentSpace.Saml2.Session;

services.Configure<CookieSsoSessionStoreOptions>(options =>
{
   options.CookieName = "my-saml-session";
});

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

Alternatively, the options may be specified through configuration.

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

```
"CookieSsoSessionStore": {
  "CookieName": "my-saml-session"
}
```

The following code in the ConfigureServices makes use of this configuration.

```
using ComponentSpace.Saml2.Configuration;
using ComponentSpace.Saml2.Session;

// Configure the cookie SSO session store.
services.Configure<CookieSsoSessionStoreOptions>(
   Configuration.GetSection("CookieSsoSessionStore"));

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

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

```
using ComponentSpace.Saml2.Bindings;
using ComponentSpace.Saml2.Configuration;
using ComponentSpace.Saml2.Session;

services.Configure<CookieSsoSessionStoreOptions>(options =>
{
   options.CookieOptions.Domain = "componentspace.com";
});

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

IIDCache

The IIDCache interface supports storing identifiers in a cache. It's used by the service provider to store SAML assertion identifiers as part of detecting potential replay attacks.

Distributed ID Cache

A default implementation uses an IDistributedCache.

The specific implementation of IDistributedCache may be specified through dependency injection.

The default implementation of IDistributedCache caches to memory.

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 SAML for ASP.NET Core Web Farm Guide.

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

IArtifactCache

The IArtifactCache interface supports storing artifacts in a cache. It's used to support the HTTP Artifact binding.

Distributed Artifact Cache

A default implementation uses an IDistributedCache.

The specific implementation of IDistributedCache may be specified through dependency injection.

The default implementation of IDistributedCache caches to memory.

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 SAML for ASP.NET Core Web Farm Guide.

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

ISamlObserver

The ISamlObserver interface supports receiving notifications of SAML SSO and logout events.

An abstract class, SamlObserver, implements this interface and can be extended to receive only the events of interest.

The SamlSubject class includes methods for subscribing and unsubscribing observers.

One possible use of this interface is for auditing.

ICertificateManager

The ICertificateManager interface manages X.509 certificates referenced by the SAML configuration.

It delegates to the ICertificateLoader to handle the loading of the certificate.

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

ICertificateLoader

The ICertificateLoader interface loads X.509 certificates stored in configuration strings, on the file system, or the Windows certificate store.

Two implementations are provided.

Cached Certificate Loader

A default implementation is included which retrieves certificates from configuration strings, the file system or Windows certificate store and caches them in memory for performance.

Non-Cached Certificate Loader

An alternative implementation retrieves certificates from configuration strings, the file system or Windows certificate store and does not cache certificates each time they're accessed. This is less performant but will pick up certificate changes immediately.

For example, the following code in the ConfigureServices makes use of the default implementations of the SAML interfaces except it replaces the ICertificateLoader with the non-cached loader.

```
// Add SAML SSO services.
services.AddSaml();

// Add non-cached certificate loader.
services.AddTransient<ICertificateLoader, CertificateLoader>();
```

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 SSO.

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

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

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

```
using ComponentSpace.Saml2.Certificates;
```

```
using ComponentSpace.Saml2.Configuration;
using ComponentSpace.Saml2.Session;

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

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

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 in the ConfigureServices makes use of this configuration.

```
using ComponentSpace.Saml2.Certificates;
using ComponentSpace.Saml2.Configuration;
using ComponentSpace.Saml2.Session;

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

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

The following code in the ConfigureServices turns off expired certificate checking.

```
using ComponentSpace.Saml2.Certificates;
using ComponentSpace.Saml2.Configuration;
using ComponentSpace.Saml2.Session;

services.Configure<CertificateValidationOptions>(options =>
{
   options.EnableNotAfterCheck= false;
});

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

ISamlConfigurationResolver

The ISamlConfigurationResolver interface resolves local and partner provider configurations.

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

Default SAML Configuration Resolver

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

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

SAML Database Configuration Resolver

The database configuration resolver retrieves configuration stored in an entity framework database.

For example, the following code in the ConfigureServices makes use of the default implementations of the SAML interfaces except it replaces the ISamlConfigurationResolver with the database configuration resolver.

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

// Add SAML SSO services using the database configuration resolver.
services.AddSaml().AddSamlDatabaseConfigurationResolver();
```

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

SAML Cached Configuration Resolver

The cached configuration resolver retrieves configuration stored in a cached and backed by an underlying configuration resolver.

For example, the following code in the ConfigureServices makes use of the default implementations of the SAML interfaces except it replaces the ISamlConfigurationResolver with the cached configuration resolver.

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

// Add SAML SSO services using the cached database configuration resolver.
services.AddSaml().AddCachedSamlDatabaseConfigurationResolver();
```

IHttpRedirectBinding

The IHttpRedirectBinding interface supports the SAML v2.0 HTTP-Redirect binding.

A default implementation of this binding is included.

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

IHttpPostBinding

The IHttpPostBinding interface supports the SAML v2.0 HTTP-Post binding.

A default implementation of this binding is included.

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

IHttpPostForm

The IHttpPostForm interface creates the HTML form that's used with the HTTP Post binding to send SAML messages.

A default implementation is included.

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

HttpPostForm

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=""samlform"" 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 SAML 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.samlform.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 in the ConfigureServices changes the target to a new tab in the browser.

```
using ComponentSpace.Saml2.Bindings.Post;
using ComponentSpace.Saml2.Configuration;

services.Configure<HttpPostFormOptions>(options =>
{
   options.Target = "_blank";
});

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

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 in the ConfigureServices makes use of this configuration.

```
using ComponentSpace.Saml2.Bindings.Post;
using ComponentSpace.Saml2.Configuration;

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

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

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.samlform.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.Saml2.Bindings.Post;

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

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

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.Saml2.Bindings.Post;

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

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

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.samlform.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.Saml2.Bindings.Post;

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

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

IHttpRequest

The IHttpRequest interface represents an HTTP request that's supplied to a binding interface.

ASP.NET Core Request

A default implementation is included which uses the Microsoft.AspNetCore.Http.HttpRequest and is suitable for ASP.NET Core web applications.

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

SamlHttpClient

The SOAP, PAOS and URI bindings send and receive messages whilst acting as an HTTP client. The SamlHttpClient is a typed HTTP client class that's used for this communication.

The following example adds a client certificate for authentication at the server.

```
using ComponentSpace.Saml2.Bindings;

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));

// Configure a client certificate for the SAML HTTP client.
var x509Certificate = new X509Certificate2("certificates/client.pfx", "password");
```

```
services.AddHttpClient<SamlHttpClient>()
   .ConfigurePrimaryHttpMessageHandler(() =>
   {
      var handler = new HttpClientHandler();

      handler.ClientCertificates.Add(x509Certificate);

      return handler;
   });
```

IHttpRequest

The IHttpRequest interface represents an HTTP request that's supplied to a binding interface.

ASP.NET Core Request

A default implementation is included which uses the Microsoft.AspNetCore.Http.HttpRequest and is suitable for ASP.NET Core web applications.

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

IHttpResponse

The IHttpResponse interface represents an HTTP response that's supplied to a binding interface.

ASP.NET Core Response

A default implementation is included which uses the Microsoft.AspNetCore.Http.HttpResponse and is suitable for ASP.NET Core web applications.

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

ISamlSchemaValidator

The ISamlSchemaValidator interface supports SAML XML schema validation.

A default implementation is included.

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

SAML Schema Validator Options

Options associated with the SAML schema validator may be set.

The SAML schema validator uses inbuilt XML schemas included in the SAML specification. However, SAML attribute values that specify a datatype other than an XML Schema simple type require the presence of an extension schema that defines the datatype.

For example, the following code in the ConfigureServices specifies an additional XML schema file to be used during validation.

```
using ComponentSpace.Saml2.Utility;

services.Configure<SamlSchemaValidatorOptions>(options =>
{
   options.ExtensionSchemaFileNames = new List<string>
```

```
   {
      "my-custom-datatypes.xsd"
   };
});

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

Alternatively, the options may be specified through configuration.

For example, the following section in appsettings.json specifies an additional XML schema file to be used during validation.

```
"SamlSchemaValidator": {
  "ExtensionSchemaFileNames": { "my-custom-datatypes.xsd" }
}
```

The following code in the ConfigureServices makes use of this configuration.

```
using ComponentSpace.Saml2.Utility;

// Configure the distributed cache SSO session store.
services.Configure<SamlSchemaValidatorOptions>(
   Configuration.GetSection("SamlSchemaValidator"));

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

IXmlSignature

The IXmlSignature interface supports XML signatures.

A default implementation is included.

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

IXmlEncryption

The IXmlEncryption interface supports XML encryption.

A default implementation is included.

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

ISamlClaimFactory

The ISamlClaimFactory interface supports the creation of security claims from a SAML assertion and vice versa.

The ISamlClaimFactory is used by the SAML authentication handler and SAML middleware as part of SAML SSO.

A default implementation is included that maps the SAML name identifier to the name and any SAML attributes as additional claims.