# ComponentSpace
# SAML for ASP.NET
# Developer Guide


www.componentspace.com

# Contents

# Introduction

The ComponentSpace SAML for ASP.NET is a .NET framework 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 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 Examples Guide.

## Visual Studio and .NET Framework Support

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

- .NET v2.0 and above

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

- ASP.NET Web Application (.NET Framework)

The SAML examples include solution and project files for:

- Visual Studio 2015 and above

# SAML API

There are a number of overloads to the SAML API methods with optional parameters not included.

Methods with the full complement of parameters are documented but overloads with fewer parameters may be used instead.

## SAMLIdentityProvider

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

### Initiating SSO

The InitiateSSO 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.

```
public static void InitiateSSO(
   HttpResponse httpResponse,
   string userName,
   SAMLAttribute[] attributes,
   string authnContext,
   string relayState,
   string partnerSP,
   string assertionConsumerServiceUrl)
```

**httpResponse**

The HTTP response is used to send the SAML response to the service provider via the browser.

**userName**

The user name 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** [optional]

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** [optional]

The authentication context identifies how the user was authenticated.

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

**relayState** [optional]

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.

**partnerSP** [optional]

The partner SP 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.

**assertionConsumerServiceUrl** [optional]

The assertion consumer service URL is the service provider endpoint to receive the SAML response.

If not specified, the configured URL is used.

### Receiving SSO Requests

The ReceiveSSO 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 SendSSO method.

```
public static void ReceiveSSO(
   HttpRequest httpRequest,
```

```
    out string partnerSP,
    out SSOOptions ssoOptions)
```

**httpRequest**

The HTTP request is used to receive the SAML authn request from the service provider.

**partnerSP**

The name identifies the service provider that sent the SAML authn request.

**ssoOptions** [optional]

SSO options included in the SAML authn request are optionally returned.

## Sending SSO Responses

The SendSSO 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, ReceiveSSO must have been called and the user authenticated at the identity provider.

```
public static void SendSSO(
    HttpResponse httpResponse,
    string userName,
    SAMLAttribute[] attributes,
    string authnContext,
    string assertionConsumerServiceUrl)
```

**httpResponse**

The HTTP response is used to send the SAML response to the service provider via the browser.

**userName**

The user name 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** [optional]

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** [optional]

The authentication context identifies how the user was authenticated.

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

**assertionConsumerServiceUrl** [optional]

The assertion consumer service URL is the service provider endpoint to receive the SAML response.

If not specified, the configured URL 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.

```
public static void SendSSO(
   HttpResponse httpResponse,
   string statusCode,
   string statusMessage,
   string assertionConsumerServiceUrl)
```

**httpResponse**

The HTTP response is used to send the SAML response to the service provider via the browser.

**statusCode**

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

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

**statusMessage** [optional]

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

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

**assertionConsumerServiceUrl** [optional]

The assertion consumer service URL is the service provider endpoint to receive the SAML response.

If not specified, the configured URL is used.

### Initiating SLO

The InitiateSLO 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.

```
public static void InitiateSLO(
   HttpResponse httpResponse,
   string logoutReason,
   string relayState)
```

**httpResponse**

The HTTP response is used to send the SAML logout request to the service provider via the browser.

**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 ReceiveSLO 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 SendSLO method.

```
public static void ReceiveSLO(
   HttpRequest httpRequest,
   HttpResponse httpResponse,
   out bool isRequest,
   out bool hasCompleted,
   out string logoutReason,
   out string partnerSP,
   out string relayState)
```

**httpRequest**

The HTTP request is used to receive the SAML logout message from the service provider.

**httpResponse**

The HTTP response is used to send the SAML logout message to the service provider via the browser.

**isRequest**

This flag indicates whether a SAML logout request or logout response was received.

**hasCompleted**

This flag indicates whether SAML logout has completed with SAML messages having been sent to all participating service providers.

**logoutReason**

The logout reason identifies why the user logged out.

**partnerSP**

The name identifies the service provider that sent the SAML logout message.

**relayState**

The relay state is opaque data that is received with the logout message.

Its use is supported but not recommended.

### Sending SLO Responses

The SendSLO 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, ReceiveSLO must have been called and the user logged out at the identity provider.

```
public static void SendSLO(
   HttpResponse httpResponse,
   string errorMessage)
```

**httpResponse**

The HTTP response is used to send the SAML logout message to the service provider via the browser.

**errorMessage** [optional]

If specified, the error message indicates why logout failed.

## SAMLServiceProvider

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

### Initiating SSO

The InitiateSSO 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.

```
public static void InitiateSSO(
   HttpResponse httpResponse,
   string relayState,
   string partnerIdP,
   SSOOptions ssoOptions,
   string assertionConsumerServiceUrl,
   string singleSignOnServiceUrl)
```

**httpResponse**

The HTTP response is used to send the SAML authn request to the identity provider via the browser.

**relayState** [optional]

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

Its use is supported but not recommended.

**partnerIdP** [optional]

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.

**ssoOptions** [optional]

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

**assertionConsumerServiceUrl** [optional]

The assertion consumer service URL is the service provider endpoint to receive the SAML response.

If not specified, the configured URL is used.

**singleSignOnServiceUrl** [optional]

The single sign-on service URL is the identity provider endpoint to receive the authn request.

If not specified, the configured URL is used.

### Receiving SSO Responses

The ReceiveSSO 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.

```
public static void ReceiveSSO(
   HttpRequest httpRequest,
   out bool isInResponseTo,
   out string partnerIdP,
   out string authnContext,
   out string userName,
   out SAMLAttribute[] attributes,
   out string relayState)
```

**httpRequest**

The HTTP request is used to receive the SAML response from the identity provider.

**isInResponseTo**

This 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.

**partnerIdP**

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.

**authnContext**

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

**userName**

The user name 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.

**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.

### Initiating SLO

The InitiateSLO method initiates single logout from the service provider to 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.

```
public static void InitiateSLO(
   HttpResponse httpResponse,
   string logoutReason,
   string relayState,
   string partnerIdP)
```

**httpResponse**

The HTTP response is used to send the SAML logout request to the identity provider via the browser.

**logoutReason** [optional]

The logout reason identifies why the user logged out.

**relayState** [optional]

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.

**partnerIdP** [optional]

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 SLO to this identity provider.

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

### Receiving SLO Messages

The ReceiveSLO 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 SendSLO method.

```
public static void ReceiveSLO(
   HttpRequest httpRequest,
   out bool isRequest,
   out string logoutReason,
   out string partnerIdP,
   out string relayState)
```

**httpRequest**

The HTTP request is used to receive the SAML logout message from the identity provider.

**isRequest**

This flag indicates whether a SAML logout request or logout response was received.

**logoutReason**

The logout reason identifies why the user logged out.

**partnerIdP**

The name identifies the identity provider that sent the SAML logout message.

**relayState**

The relay state is opaque data that is received with the logout message.

Its use is supported but not recommended.

### Sending SLO Responses

The SendSLO 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, ReceiveSLO must have been called and the user logged out at the service provider.

```
public static void SendSLO(
   HttpResponse httpResponse,
   string errorMessage)
```

**httpResponse**

The HTTP response is used to send the SAML logout message to the identity provider via the browser.

**errorMessage** [optional]

If specified, the error message indicates why logout failed.

## SAMLController

The SAMLController class maintains the SAML configuration and other properties.

### Configuration

Configuration may be set programmatically.

Use the Configuration property to access the SAML configuration.

```
public static SAMLConfiguration Configuration
```

If multiple SAML configurations exist as part of a multi-tenanted environment, they may be accessed through the Configurations property.

```
public static SAMLConfigurations Configurations
```

Refer to the SAML for ASP.NET Configuration Guide for more information.

### Configuration Selection

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

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

It's only required if multiple SAML configurations exist.

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

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

```
Task SetConfigurationIDAsync(string configurationID);
```

**configurationID**

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

Refer to the SAML for ASP.NET Configuration Guide for more information.

### Configuration Resolution

SAML configuration may be supplied by implementing the ISAMLConfigurationResolver interface.

```
public static ISAMLConfigurationResolver ConfigurationResolver
```

Refer to the SAML for ASP.NET Configuration Guide for more information.

### SAML Session and ID Cache Storage

Custom SAML session and identifier cache implementations may be specified.

```
public static ISSOSessionStore SSOSessionStore
public static IIDCache IDCache
```

Refer to the Customizations section for more information.

### Certificate Management

A custom certificate manager implementation may be specified.

```
public static ICertificateManager CertificateManager
```

Refer to the Customizations section for more information.

## SSOOptions

The SSOOptions class encapsulates various options associated with SSO.

**Subject**

The SAML subject.

It's included in the SAML authentication request.

The default is none.

**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.

It's 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.

### TrustedIdentityProvider

The TrustedIdentityProvider class 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.

## SSO Status

The SAMLIdentityProvider and SAMLServiceProvider classes include a number of methods that return the SSO status for the user.

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

```
public static bool IsSSO()
```

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

```
public static 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 static 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 static 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 static bool IsSLOCompletionPending(string partnerName)
```

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

```
public static bool CanSLO()
```

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

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

### License

The License class 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.

## 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.

## Customizations

A number of interfaces are exposed to enable custom implementations.

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

### 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.

A number of implementations are provided.

#### In-Memory Session Store

A default implementation stores SSO session data in a memory cache.

This is suitable for single server deployments.

For web farm deployments, a central store should be specified. For more information, refer to the SAML for ASP.NET Web Farm Guide.

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_SessionId" and it's marked as secure and HTTP only.

An example set-cookie response header is shown below.

```
set-cookie: SAML_SessionId=d74b7c8d-1a21-4322-ba76-af2671d5c856; path=/; secure; httponly
```

The cookie name may be changed at application start-up.

```
using ComponentSpace.SAML2.Data;

protected void Application_Start(object sender, EventArgs e)
{
   SAMLCookieOptions.Name = "My_SAML_SessionId";
}
```

The secure (i.e. HTTPS only) flag may be disabled at application start-up.

However, it's recommended that all communication is over HTTPS and that the secure flag is enabled.

```
using ComponentSpace.SAML2.Data;
```

```
protected void Application_Start(object sender, EventArgs e)
{
   SAMLCookieOptions.Secure = false;
}
```

### ASP.NET Session Store

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

```
SAMLController.SSOSessionStore = new HttpSSOSessionStore();
```

### Database Session Store

An alternative implementation stores SSO session data in a custom SQL database.

```
SAMLController.SSOSessionStore = new DatabaseSSOSessionStore();
```

The SSO session data is stored in a single table, SSOSessions, with the following schema.

| Column Name | Data Type | Allow Nulls |
|---|---|---|
| SessionID | nvarchar(64) | ☐ |
| UpdateDateTime | datetime | ☐ |
| SessionObject | varbinary(MAX) | ☐ |

The SessionID column is the unique identifier for the SSO session data.

The UpdateDateTime is the UTC date/time when the SSO session data was last updated.

The SessionObject is the opaque session data stored as a byte array.

The following SQL script creates the SSOSessions table in a SQL Server database.

```
USE [SAML]
GO
DROP TABLE [dbo].[SSOSessions]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[SSOSessions](
        [SessionID] [nvarchar](64) NOT NULL,
        [UpdateDateTime] [datetime] NOT NULL,
        [SessionObject] [varbinary](max) NOT NULL,
```

```
 CONSTRAINT [PK_SSOSessions] PRIMARY KEY CLUSTERED
(
	[SessionID] ASC
) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
```

The default DatabaseSSOSessionStore constructor assumes a web.config connection string named SAML and that the table name is SSOSessions.

```
<connectionStrings>
  <add name="SAML" providerName="System.Data.SqlClient" connectionString="data
source=localhost;database=SAML;uid=saml;pwd=password"/>
</connectionStrings>
```

It's anticipated that, in many instances, the SSOSessions table would be added to an application's existing database.

Alternatively, a separate database deployed to SQL Server or some other DBMS may be used.

The only requirement is that the table includes the specified columns.

Additional DatabaseSSOSessionStore constructors and properties are available to specify the connection string name, table name and column names.

The DatabaseSSOSessionStore class includes a DeleteExpired method that removes any expired SSO session data from the table. This method should be called periodically to maintain the table. Alternatively, the Delete method may be called to delete specific SSO session data from the table.

There is also the option to maintain this table independently as part of general database maintenance.

Care should be taken to ensure this table doesn't grow too large with expired rows.

## 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.

A number of implementations are provided.

### In-Memory ID Cache

A default implementation stores identifiers in a memory cache.

This is suitable for single server deployments.

For web farm deployments, a central store should be specified. For more information, refer to the SAML for ASP.NET Web Farm Guide.

## Database ID Cache

An alternative implementation stores SSO session data in a custom SQL database.

```
SAMLController.IDCache = new DatabaseIDCache();
```

The SSO session data is stored in a single table, SAMLIdentifiers, with the following schema.

| | Column Name | Data Type | Allow Nulls |
|---|---|---|---|
| 🔑 | ID | nvarchar(256) | ☐ |
| | ExpirationDateTime | datetime | ☐ |

The ID column is the SAML identifier e.g. the SAML assertion ID.

The ExpirationDateTime is the UTC date/time when the SAML identifier expires.

The following SQL script creates the SAMLIdentifiers table in a SQL Server database.

```
USE [SAML]
GO
DROP TABLE [dbo].[SAMLIdentifiers]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[SAMLIdentifiers](
	[ID] [nvarchar](256) NOT NULL,
	[ExpirationDateTime] [datetime] NOT NULL,
 CONSTRAINT [PK_SAMLIdentifiers] PRIMARY KEY CLUSTERED
(
	[ID] ASC
) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

The default DatabaseIDCache constructor assumes a web.config connection string named SAML and that the table name is SAMLIdentifiers.

```
<connectionStrings>
  <add name="SAML" providerName="System.Data.SqlClient" connectionString="data
source=localhost;database=SAML;uid=saml;pwd=password"/>
</connectionStrings>
```

It's anticipated that, in many instances, the SAMLIdentifiers table would be added to an application's existing database.

Alternatively, a separate database deployed to SQL Server or some other DBMS may be used.

The only requirement is that the table includes the specified columns.

Additional DatabaseIDCache constructors and properties are available to specify the connection string name, table name and column names.

The DatabaseIDCache class includes a DeleteExpired method that removes any expired identifiers from the table. This method should be called periodically to maintain the table. Alternatively, the Delete method may be called to delete specific identifiers from the table.

There is also the option to maintain this table independently as part of general database maintenance.

Care should be taken to ensure this table doesn't grow too large with expired rows.

## ISAMLObserver

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

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

The SAMLObservable 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.

```
SAMLController.CertificateManager = new CertificateManager(new CertificateLoader());
```

## ISAMLConfigurationResolver

The ISAMLConfigurationResolver interface resolves local and partner provider configurations.

A default implementation is included that resolves configuration through the SAML configuration specified either using a configuration file (e.g. saml.config) or programmatically.

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

## HTMLFormTemplate

The HTTPPostBinding.HTMLFormTemplate property specifies 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.

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

    if (document.readyState === "loading") {
      document.addEventListener("DOMContentLoaded", submitForm);
    } else {
      submitForm();
    }
  </script>
  <body>
    <noscript>
      <p>
        Since your browser doesn't support JavaScript, you must press the Continue button to
proceed.
      </p>
    </noscript>
    <form id="samlform" action="{url}" method="post" target="_self">
      <div>
        {hiddenFormVariables}
      </div>
      <noscript>
        <div>
          <input type="submit" value="Continue"/>
        </div>
      </noscript>
    </form>
  </body>
</html>
```

The following substitution parameters are supported.

**url**

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

**hiddenFormVariables**

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

## Content-Security-Policy Header Support

Content Security Policy (CSP) permits the control of resources, including JavaScripts, that the browser is allowed to 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 'self' 'unsafe-inline'
```

The application would need to update the Content-Security-Policy header.

### *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 'self' 'nonce-2BAC238EBCE24A24ABCC11132361D228'
```

The application would be responsible for setting the nonce on a frequent basis.

The application would need to update the Content-Security-Policy header with this source.

### *Hash*

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

There are various online tools, including the Chrome developer tools, for generating the hash.

The corresponding policy would include the hash.

```
Content-Security-Policy: script-src 'self' 'sha256-
N0e3VBNAeTyeExalDvUSJfWBKhi4UDjKB74Zq4l85+s='
```

The application would need to update the Content-Security-Policy header with this source.

### *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 HTML template specifies a script source rather than inline script.

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

The script source contains the JavaScript.

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

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

The corresponding policy would include the trusted source.

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

The application would be responsible for specifying the HTML template.

The application would need to update the Content-Security-Policy header with this source.