



ComponentSpace

# SAML v2.0 for.NET Developer Guide

# Contents

1	Introduction.....	1
1.1	Features .....	1
1.2	Benefits.....	1
1.3	Prerequisites .....	2
2	Getting Started .....	2
3	An Introduction to SAML SSO .....	2
3.1	IdP-Initiated SSO .....	2
3.2	SP-Initiated SSO .....	3
3.3	IdP-Initiated SLO .....	4
3.4	SP-Initiated SLO .....	5
3.5	Security Considerations.....	6
3.5.1	Transport Level Security.....	6
3.5.2	XML Signatures.....	6
3.5.3	XML Encryption.....	7
4	Using the Class Library .....	8
4.1	Adding a Reference.....	8
4.2	Distribution.....	9
5	SAML High Level API.....	9
5.1	SAML Identity Provider APIs.....	9
5.1.1	SAMLIdentityProvider.InitiateSSO.....	9
5.1.2	SAMLIdentityProvider.ReceiveSSO .....	10
5.1.3	SAMLIdentityProvider.SendSSO .....	10
5.1.4	SAMLIdentityProvider.InitiateSLO .....	10
5.1.5	SAMLIdentityProvider.ReceiveSLO .....	10
5.1.6	SAMLIdentityProvider.SendSLO.....	11
5.2	SAML Service Provider APIs .....	11
5.2.1	SAMLServiceProvider.InitiateSSO .....	11
5.2.2	SAMLServiceProvider.ReceiveSSO .....	12
5.2.3	SAMLServiceProvider.InitiateSLO.....	12
5.2.4	SAMLServiceProvider.ReceiveSLO .....	13
5.2.5	SAMLServiceProvider.SendSLO .....	13
5.3	Configuration Management.....	13
5.4	ICertificateManager Interface .....	14
5.4.1	CertificateManager .....	14
5.4.2	Custom ICertificateManager.....	14
5.5	IIDCache Interface .....	16
5.5.1	InMemoryIDCache .....	16
5.5.2	DatabaseIDCache.....	16
5.5.3	Custom IDCache.....	18
5.6	ISSOSessionStore Interface .....	18
5.6.1	HttpSSOSessionStore .....	18
5.6.2	DatabaseSSOSessionStore .....	18
5.6.3	Custom ISSOSessionStore.....	20
6	SAML High Level API Configuration .....	20

6.1	SAML Configurations.....	27
6.2	SAML Configuration .....	28
6.3	Identity Provider Configuration .....	28
6.4	Service Provider Configuration.....	28
6.5	Partner Identity Provider Configuration.....	29
6.6	Partner Service Provider Configuration .....	30
6.7	Partner Provider Configuration .....	31
6.8	Provider Configuration.....	34
6.9	Miscellaneous Configuration .....	35
6.9.1	Name ID Format Types .....	35
6.9.2	Authentication Contexts.....	35
6.9.3	Binding Types .....	36
6.9.4	Key Encryption Methods .....	36
6.9.5	Data Encryption Methods .....	36
6.9.6	Digest Methods .....	37
6.9.7	Signature Methods .....	37
6.10	Specifying Configuration Programmatically .....	37
7	SAML High Level API Certificate Configuration.....	38
7.1	Local Provider Certificate File.....	39
7.2	Partner Provider Certificate File .....	39
7.3	Encrypting the Certificate File Password.....	39
7.4	Managing the Windows Certificate Store .....	40
7.4.1	Running the MMC Certificates Snap-in .....	40
7.4.2	Importing a PFX File .....	40
7.4.3	Private Key Security .....	50
7.4.4	Importing a CER File.....	52
7.5	Local Provider Certificate Store.....	60
7.6	Partner Provider Certificate Store .....	61
8	Selecting the Most Applicable Example.....	61
8.1	High Level APIs.....	62
8.2	Low Level APIs .....	63
9	Building the Example Applications .....	64
10	Example Applications – High Level APIs.....	64
10.1	Web Forms Identity Provider and Service Provider .....	64
10.1.1	Installing the Web Forms Identity Provider.....	64
10.1.2	Installing the Web Forms Service Provider .....	65
10.1.3	Configuring the Web Forms Identity Provider .....	66
10.1.4	Configuring the Web Forms Service Provider.....	66
10.1.5	IdP-Initiated SSO from the Web Forms Identity Provider .....	67
10.1.6	SP-Initiated SSO from the Web Forms Service Provider .....	68
10.1.7	Code Walkthrough - IdP-Initiated SSO .....	70
10.1.8	Code Walkthrough - SP-Initiated SSO .....	70
10.2	MVC Identity Provider and Service Provider.....	71
10.2.1	Installing the MVC Identity Provider .....	71
10.2.2	Installing the MVC Service Provider.....	72
10.2.3	Configuring the MVC Identity Provider.....	73

10.2.4	Configuring the MVC Service Provider .....	73
10.2.5	IdP-Initiated SSO from the MVC Identity Provider .....	74
10.2.6	SP-Initiated SSO from the MVC Service Provider.....	75
10.2.7	Code Walkthrough - IdP-Initiated SSO .....	76
10.2.8	Code Walkthrough - SP-Initiated SSO .....	77
10.3	OWIN Identity Provider and Service Provider .....	77
10.3.1	Installing the OWIN Identity Provider .....	77
10.3.2	Installing the OWIN Service Provider .....	78
10.3.3	Configuring the OWIN Identity Provider .....	79
10.3.4	Configuring the OWIN Service Provider .....	79
10.3.5	IdP-Initiated SSO from the OWIN Identity Provider .....	80
10.3.6	SP-Initiated SSO from the OWIN Service Provider.....	81
10.3.7	Code Walkthrough - IdP-Initiated SSO .....	82
10.3.8	Code Walkthrough - SP-Initiated SSO .....	83
10.4	ADFS Interoperability .....	83
10.4.1	Miscellaneous Configuration .....	83
10.4.2	Configuring the Service Provider .....	84
10.4.3	Configuring ADFS – Adding a Relying Party.....	85
10.4.4	Running the Service Provider with SP-Initiated SSO.....	100
10.4.5	Running the Service Provider with IdP-Initiated SSO .....	102
10.4.6	Configuring the Identity Provider.....	104
10.4.7	Configuring ADFS – Adding a Claims Provider.....	104
10.4.8	Running the Identity Provider with IdP-Initiated SSO .....	115
10.4.9	Troubleshooting ADFS SSO.....	117
10.5	Office 365 Interoperability .....	117
10.5.1	Configuring the Identity Provider.....	117
10.5.2	Configuring Office 365.....	118
10.5.3	Adding a User .....	127
10.5.4	Deleting a User .....	128
10.5.5	Running the Identity Provider with SP-Initiated SSO .....	128
10.5.6	Running the Identity Provider with IdP-Initiated SSO.....	131
10.5.7	Email Client Support.....	132
10.5.8	Configuring an Email Client.....	133
10.5.9	Running the Email Client.....	141
10.5.10	Troubleshooting Office 365 SSO.....	142
10.6	Google Apps Interoperability .....	142
10.6.1	Configuring the Identity Provider.....	142
10.6.2	Configuring Google Apps.....	143
10.6.3	Running Google Apps with SSO .....	144
10.6.4	Troubleshooting Google Apps SSO.....	145
10.7	Salesforce Interoperability .....	145
10.7.1	Configuring the Identity Provider .....	145
10.7.2	Configuring Salesforce as a Service Provider .....	145
10.7.3	Running the Example Identity Provider – IdP-Initiated SSO.....	147
10.7.4	Running the Example Identity Provider – SP-Initiated SSO.....	148
10.7.5	Configuring the Service Provider .....	148

10.7.6	Configuring Salesforce as an Identity Provider .....	148
10.7.7	Running the Example Service Provider – IdP-Initiated SSO .....	150
10.7.8	Running the Example Service Provider – SP-Initiated SSO .....	151
10.7.9	Troubleshooting Salesforce SSO .....	152
10.8	Shibboleth Interoperability .....	153
10.8.1	Configuring the Identity Provider .....	153
10.8.2	Configuring the Service Provider .....	153
10.8.3	Configuring Shibboleth.....	154
10.8.4	Running Shibboleth with SSO – Example Identity Provider.....	155
10.8.5	Running Shibboleth with SSO – Example Service Provider .....	157
10.8.6	Troubleshooting Shibboleth SSO .....	159
11	Example Applications - Low Level APIs .....	159
11.1	SP-Initiated SSO – Identity Provider .....	160
11.1.1	Installing the Identity Provider .....	160
11.1.2	Configuring the Identity Provider .....	160
11.1.3	Running the Identity Provider.....	160
11.1.4	Running the Identity Provider in Visual Studio.....	161
11.2	SP-Initiated SSO – Service Provider .....	162
11.2.1	Installing the Service Provider.....	162
11.2.2	Configuring the Service Provider .....	162
11.2.3	Running the Service Provider without SSO.....	162
11.2.4	Running the Service Provider with SSO.....	163
11.2.5	Running the Service Provider in Visual Studio .....	166
11.2.6	Service Provider SSO Execution Flow .....	166
11.3	IdP-Initiated SSO – Service Provider .....	167
11.3.1	Installing the Service Provider.....	167
11.3.2	Configuring the Service Provider .....	167
11.3.3	Running the Service Provider .....	167
11.3.4	Running the Service Provider in Visual Studio .....	168
11.4	IdP-Initiated SSO – Identity Provider.....	168
11.4.1	Installing the Identity Provider .....	168
11.4.2	Configuring the Identity Provider .....	168
11.4.3	Running the Identity Provider.....	168
11.4.4	Running the Identity Provider in Visual Studio.....	169
11.5	ADFS Interoperability – Service Provider.....	169
11.5.1	Installing the Service Provider.....	169
11.5.2	Configuring the Service Provider .....	169
11.5.3	Miscellaneous Configuration .....	170
11.5.4	Configuring ADFS.....	170
11.5.5	Running the Service Provider without SSO.....	180
11.5.6	Running the Service Provider with SSO.....	181
11.6	Google Apps Interoperability – Identity Provider .....	182
11.6.1	Installing the Identity Provider .....	182
11.6.2	Configuring the Identity Provider.....	182
11.6.3	Configuring Google Apps.....	182
11.6.4	Running Google Apps.....	183

11.7	Salesforce Interoperability – Identity Provider.....	183
11.7.1	Installing the Identity Provider .....	183
11.7.2	Configuring the Identity Provider.....	183
11.7.3	Configuring Salesforce .....	183
11.7.4	Running Salesforce .....	184
11.7.5	Validating SAML Responses in Salesforce.....	184
11.8	Shibboleth Interoperability – Identity Provider .....	184
11.8.1	Installing the Identity Provider .....	184
11.8.2	Configuring the Identity Provider.....	185
11.8.3	Running the Identity Provider.....	185
11.8.4	Running the Identity Provider in Visual Studio.....	185
11.9	Shibboleth Interoperability – Service Provider.....	185
11.9.1	Installing the Service Provider.....	186
11.9.2	Configuring the Service Provider .....	186
11.9.3	Running the Service Provider without SSO.....	186
11.9.4	Running the Service Provider with SSO.....	187
11.9.5	Running the Service Provider in Visual Studio .....	187
11.10	Assertion Examples .....	187
11.10.1	SAML Assertion Example Application .....	187
11.11	Metadata Examples.....	188
11.11.1	Import Metadata Example Application .....	188
11.11.2	Export Metadata Example Application .....	188
11.11.3	SAML Metadata Example Application.....	189
11.11.4	ReadMetadata.....	189
11.12	Signature Examples .....	189
11.12.1	SHA-256 Signature Example Application .....	189
11.12.2	SignSAML .....	189
11.12.3	VerifySAML .....	190
11.13	Utility Applications.....	190
11.13.1	ValidateConfig .....	190
11.13.2	ValidateXML .....	190
11.13.3	EncryptSAML .....	191
11.13.4	DecryptSAML.....	191
11.13.5	ParseHttpRedirectUrl .....	191
11.13.6	Java Utilities.....	191
12	Creating your own SSO Application .....	192
12.1	Considerations.....	192
12.1.1	Error Handling .....	192
12.1.2	Configuration .....	192
12.1.3	Key Management .....	193
12.1.4	Security .....	193
13	Test Certificates and Keys .....	193
13.1	Makecert .....	193
13.1.1	Makecert and SHA-256 XML Signatures.....	194
13.2	Microsoft Certificate Server .....	194
13.3	Keytool.....	195

14	SAML Metadata.....	195
14.1	Metadata Production .....	195
14.2	Metadata Consumption .....	196
14.3	Importing and Exporting Metadata.....	196
15	Troubleshooting .....	196
15.1	Tracing .....	196
15.1.1	Diagnostic Tracing in Web Applications.....	196
15.1.2	Diagnostic Tracing in Non-Web Applications .....	198
15.2	Troubleshooting XML Signatures .....	199
15.2.1	VerifySAML .....	199
15.2.2	VerifySAML Log File .....	199
15.2.3	Java VerifyXMLSignature.....	200
15.2.4	XML Signatures and Prefixes.....	200
15.3	Troubleshooting Loading Certificates .....	201
15.3.1	Certificates Stored in Files.....	201
15.3.2	Certificates Stored in the Windows Certificate Store .....	204
16	Generating and Verifying Signatures.....	205
16.1	Signature Generation .....	205
16.2	Signature Verification.....	206
16.3	SHA-256, SHA-384 and SHA-512 Support.....	207
16.3.1	.NET 4.0 Framework Support.....	207
16.3.2	CLR Security Update.....	207
17	Extracting SAML Assertions from SAML Responses .....	209
17.1	Extracting a SAML Assertion.....	210
17.2	Extracting a Signed SAML Assertion.....	210
17.3	Extracting an Encrypted Assertion .....	210
18	Encrypted Assertions .....	210
19	Extracting Statements from SAML Assertions.....	211
20	Extracting SAML Attributes.....	211
21	Class Library Reference.....	212
22	Class Library Version .....	212
23	Frequently Asked Questions .....	214
24	Support.....	217

# 1 Introduction

The ComponentSpace SAML v2.0 component 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 applications to participate in SAML v2.0 federated single sign-on (SSO) either as an Identity Provider (IdP) or Service Provider (SP). Samples applications with full source code are included.

If you're looking for SAML v1.1 support, please refer to the separate ComponentSpace SAML v1.1 component.

## 1.1 Features

The class library supports the SAML v2.0 Assertions, Protocol, Bindings and Profiles as defined by the OASIS standard ([www.oasis-open.org](http://www.oasis-open.org)) including:

- ❖ SAML 2.0 assertions and all protocol messages
- ❖ SAML 2.0 metadata
- ❖ All bindings (SOAP, PAOS, HTTP Redirect, HTTP POST, HTTP Artifact and SAML URI)
- ❖ Web browser single sign-on profile
- ❖ Single logout profile
- ❖ Artifact resolution profile
- ❖ Identity provider discovery profile
- ❖ Authentication, attribute, assertion query profiles
- ❖ Name identifier management and mapping profiles
- ❖ Generation and verification of XML signatures
- ❖ XML encryption

The library features classes for:

- ❖ Creating, modifying and accessing SAML assertions and protocol messages
- ❖ Sending and receiving SAML protocol messages across the various SAML bindings.
- ❖ Supporting the SAML profiles

## 1.2 Benefits

- ❖ Easy to use class library enabling single sign-on support to be quickly added to your ASP.NET applications
- ❖ Developer based licensing with no runtime royalties meaning you don't pay per deployment or end user
- ❖ Developed in C# with full source code available for purchase

- ❖ Includes class library reference and example applications with full source code

### **1.3 Prerequisites**

The class library requires the .NET v2.0 framework or above and is for use with Visual Studio 2005 or above.

It has been tested with .NET framework versions 2.0, 3.0, 3.5, 4.0, 4.5 and 4.6, using Visual Studio 2005, 2008, 2010, 2012, 2013 and 2015, on 32-bit and 64-bit Windows Server 2003, 2008, and 2012, as well as Windows 7, 8 and 10.

## **2 Getting Started**

1. If you haven't already done so, install this product by double clicking the Microsoft Installer (MSI) file and following the installation steps.

A free evaluation copy is available from our web site.

2. If you're not familiar with SAML single sign-on, refer to section 3 for a brief introduction to SAML.
3. See section 5 for assistance in determining the most applicable example project to review.
4. Compile and run the example applications (see section 9).
5. See section 12 for assistance in enabling single sign-on in your own applications.

Please feel free to contact us if you need any assistance (see section 24).

## **3 An Introduction to SAML SSO**

This is a brief introduction to SAML single sign-on (SSO). For more detailed information you should contact us or refer to the SAML v2.0 specification documents at [www.oasis-open.org](http://www.oasis-open.org).

SAML single sign-on's goal is to minimize the number of times a user has to login at various web sites. It does this by having the user manually login at one site (called the identity provider or IdP) and then automatically logging in, without having to provide credentials, at one or more other sites (called the service providers or SPs).

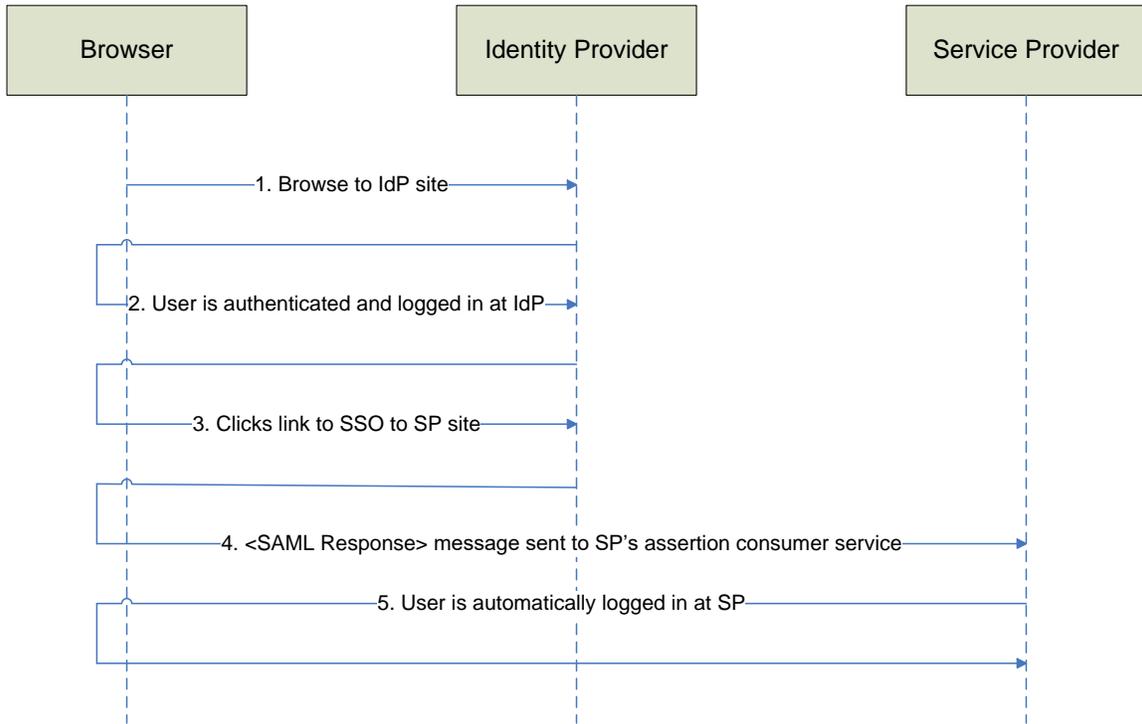
A trust relationship must exist between the identity provider and service providers. Service providers trust that the identity provider has authenticated the user.

SAML supports two single sign-on flows – IdP-initiated SSO and SP-initiated SSO.

### **3.1 IdP-Initiated SSO**

In IdP-initiated SSO, the user starts at the IdP site, logs in and clicks a link to the SP site which initiates SSO.

The following diagram outlines the IdP-initiated SSO flow.



**Figure 1 IdP-initiated SSO**

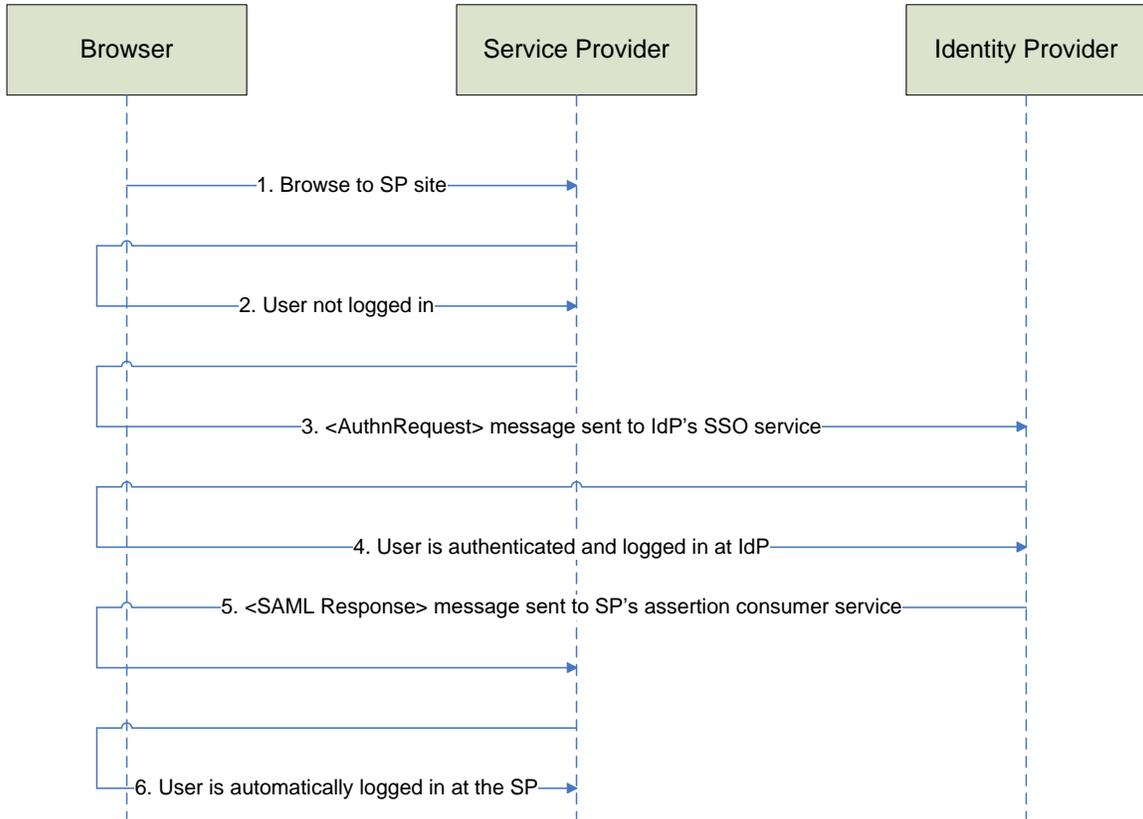
1. The user browses to the IdP site.
2. If the user is not already authenticated at the IdP, the user must present their credentials and login.
3. The user clicks a link to the SP site.
4. The IdP sends a SAML response containing a SAML assertion to the SP.
5. The SP uses the information contained in the SAML assertion, including the user's name and any associated attributes, and performs an automatic login.

Note that steps 2 and 3 may be in reverse order.

### **3.2 SP-Initiated SSO**

In SP-initiated SSO, the user starts at the SP site and, instead of logging in at the SP site, SSO is initiated with the IdP.

The following diagram outlines the SP-initiated SSO flow.



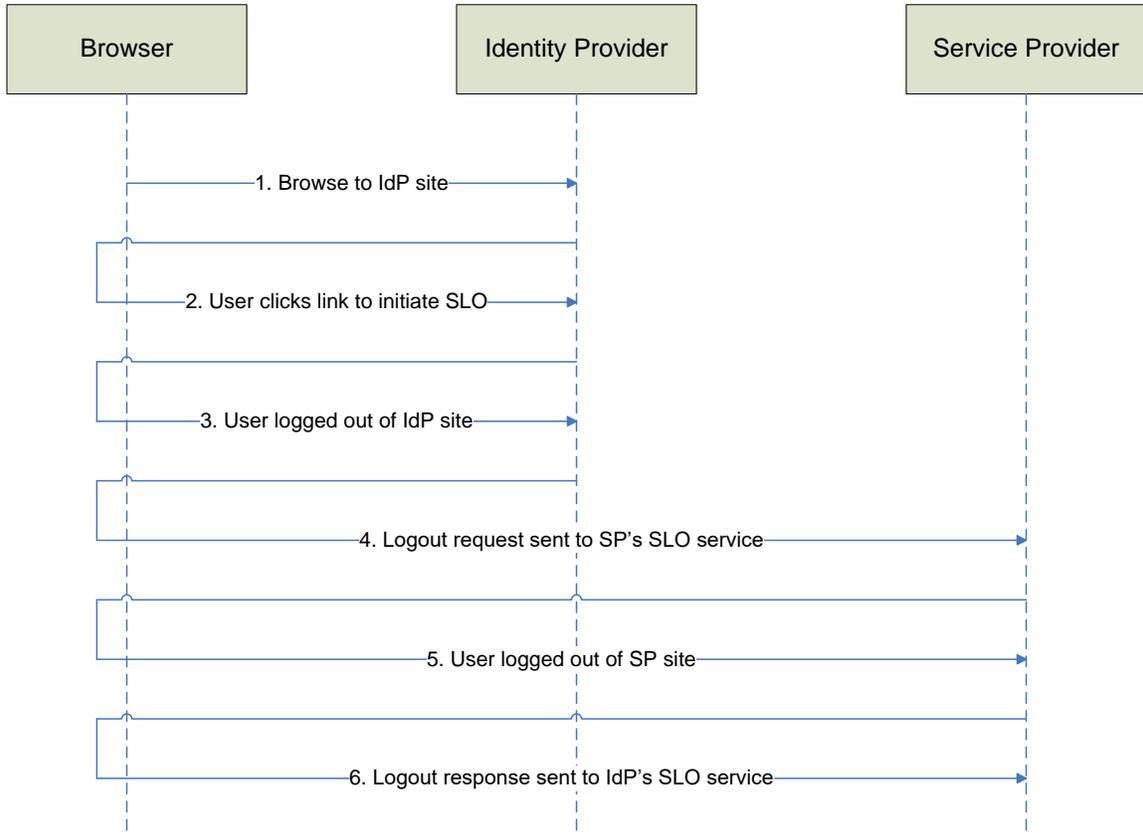
**Figure 2 SP-initiated SSO**

1. The user browses to the SP site.
2. The user attempts to access a protected page requiring the user to be authenticated.
3. The SP sends an authentication request to the IdP's SSO service endpoint.
4. If the user is not already authenticated at the IdP, the user must present their credentials and login.
5. The IdP sends a SAML response containing a SAML assertion to the SP.
6. The SP uses the information contained in the SAML assertion, including the user's name and any associated attributes, and performs an automatic login.

### **3.3 IdP-Initiated SLO**

In IdP-initiated single logout (SLO), the user starts at the IdP site, and clicks a link to logout out of the IdP site and every SP site to which there is an SSO session.

The following diagram outlines the IdP-initiated SLO flow.



**Figure 3 IdP-initiated SLO**

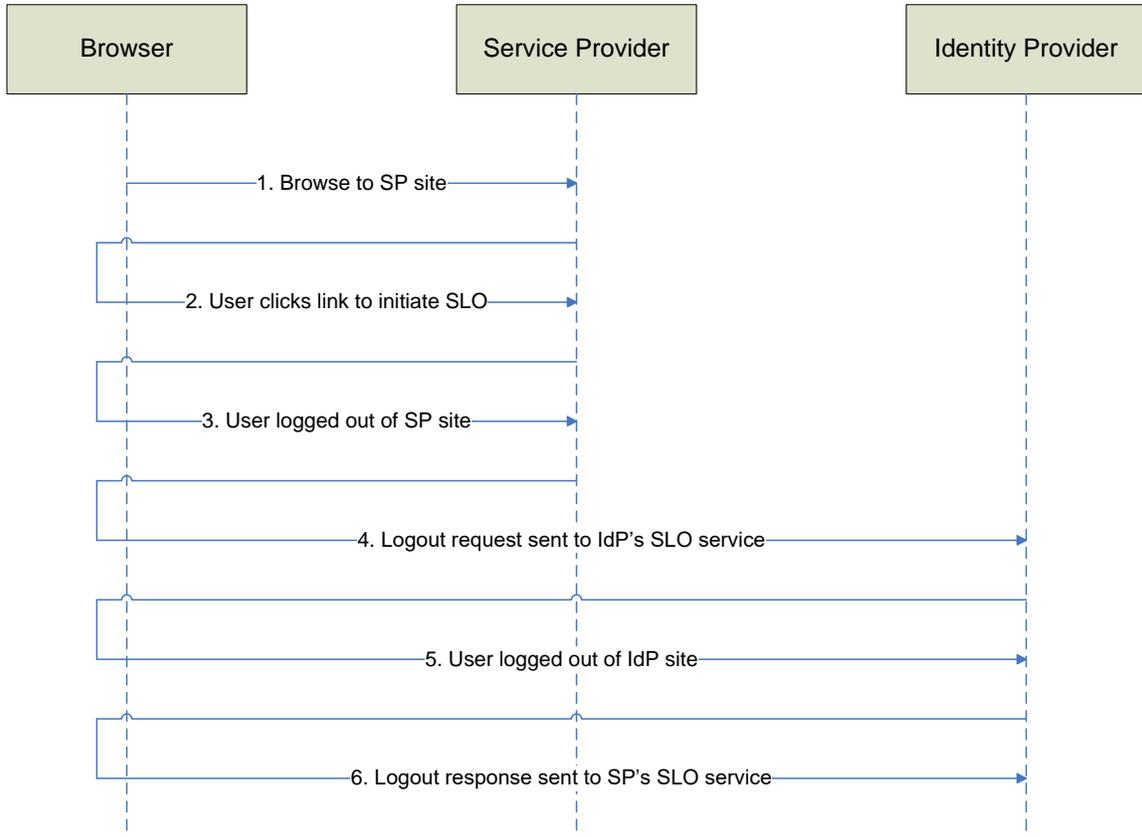
1. The user has already SSO'd to one or more service providers.
2. The user clicks a link at the IdP site to initiate SLO.
3. The user is logged out of the IdP site.
4. A logout request is sent to the SP site.
5. The user is logged out of the SP site.
6. A logout response is sent to the IdP site.

Note that steps 4 through 6 are repeated for each service provider.

### **3.4 SP-Initiated SLO**

In SP-initiated single logout (SLO), the user starts at the SP site, and clicks a link to logout out of the IdP site and every SP site to which there is an SSO session.

The following diagram outlines the SP-initiated SLO flow.



**Figure 4 SP-initiated SLO**

1. The user has already SSO'd to one or more service providers.
2. The user clicks a link at the SP site to initiate SLO.
3. The user is logged out of the SP site.
4. A logout request is sent to the IdP site.
5. The user is logged out of the IdP site.
6. A logout response is sent to the SP site.

Note that the identity provider sends a logout request and expects a logout response from every other service provider apart from the initiating service provider. This occurs between steps 5 and 6.

### **3.5 Security Considerations**

#### **3.5.1 Transport Level Security**

The SAML specification recommends that all communications are over HTTPS.

#### **3.5.2 XML Signatures**

XML signatures may be used to sign SAML messages, assertions and metadata. For example, a SAML response containing a SAML assertion may be signed. Alternatively, just the SAML assertion may be signed.





Once the reference has been added you can refer to the various SAML v2.0 classes from within your project.

## **4.2 Distribution**

The class library's runtime is royalty free which means it may be freely distributed with your application. The only file that should be distributed is the class library DLL.

## **5 SAML High Level API**

The class library includes both high level and low level APIs. For the majority of use cases, it's recommend the high level APIs are used as these provide the greatest ease of use. The low level APIs are available for when maximum flexibility is required.

The following sub-sections outline the high level API.

Refer to the class library reference (see section 21) for more information on the high level and low level APIs.

### **5.1 SAML Identity Provider APIs**

The following APIs may be called when acting as an identity provider.

#### **5.1.1 SAMLIdentityProvider.InitiateSSO**

The InitiateSSO method sends a SAML response to the specified service provider as part of IdP-initiated SSO.

For example:

```
SAMLIdentityProvider.InitiateSSO(  
    Response,  
    "testuser",  
    new Dictionary<string, string>() {  
        { "membership-level", "platinum" },  
        { "membership-number", "12345678" } },  
    null,  
    null);
```

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

The second parameter is the name of the user.

The third parameter is the user's optional attribute names and values.

The fourth parameter is the target service provider URL or null if the default page should be displayed.

The fifth parameter is the partner service provider's name or null if there's only one configured partner service provider.

### 5.1.2 SAMLIdentityProvider.ReceiveSSO

The ReceiveSSO method receives an authn request from a service provider as part of SP-initiated SSO.

For example:

```
SAMLIdentityProvider.ReceiveSSO(  
    Request,  
    out partnerSP);
```

The Request is used to receive the authn request.

The partnerSP receives the name of the service provider that sent the authn request.

### 5.1.3 SAMLIdentityProvider.SendSSO

The SendSSO method sends a SAML response to the service provider as part of SP-initiated SSO.

For example:

```
SAMLIdentityProvider.SendSSO(  
    Response,  
    "testuser",  
    new Dictionary<string, string>() {  
        { "membership-level", "platinum" },  
        { "membership-number", "12345678" } });
```

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

The second parameter is the name of the user.

The third parameter is the user's optional attribute names and values.

### 5.1.4 SAMLIdentityProvider.InitiateSLO

The InitiateSLO method sends a logout request to each service provider in session as part of IdP-initiated SLO.

For example:

```
SAMLIdentityProvider.InitiateSLO(  
    Response,  
    null);
```

The Response is used to send the logout request to the service provider via the browser.

The second parameter is the logout reason or null if none.

### 5.1.5 SAMLIdentityProvider.ReceiveSLO

The ReceiveSLO method receives a logout request from a service provider as part of SP-initiated SLO or a logout response from a service provider as part of IdP-initiated SLO.

For example:

```
SAMLIdentityProvider.ReceiveSLO(
    Request,
    Response,
    out isRequest,
    out hasCompleted,
    out logoutReason,
    out partnerSP);
```

The Request is used to receive the logout message.

The Response is used to send a logout message.

The isRequest receives the flag indicating whether a logout request or response has been received.

The hasCompleted receives the flag indicating whether the IdP-initiated SLO has completed.

The logoutReason receives the logout reason.

The partnerSP receives the name of the service provider that sent the logout message.

### 5.1.6 SAMLIdentityProvider.SendSLO

The SendSLO method sends a logout message to the service provider.

For example:

```
SAMLIdentityProvider.SendSLO(
    Response,
    null);
```

The Response is used to send the logout message to the service provider via the browser.

The second parameter is the error message or null if none.

## 5.2 SAML Service Provider APIs

The following APIs may be called when acting as a service provider.

### 5.2.1 SAMLServiceProvider.InitiateSSO

The InitiateSSO method sends an authn request to the specified identity provider as part of SP-initiated SSO.

For example:

```
SAMLServiceProvider.InitiateSSO(
    Response,
    null,
    null);
```

The Response object is used to send the authn request to the identity provider via the browser.

The second parameter is the relay state (e.g. target URL) or null if not required.

The third parameter is the partner identity provider's name or null if there's only one configured partner identity provider.

### 5.2.2 SAMLServiceProvider.ReceiveSSO

The ReceiveSSO method receives a SAML response from an identity provider as part of either IdP-initiated SSO or SP-initiated SSO.

For example:

```
SAMLServiceProvider.ReceiveSSO(  
    Request,  
    out isInResponseTo,  
    out partnerIdP,  
    out userName,  
    out attributes,  
    out targetUrl);
```

The Request is used to receive the SAML response.

The isInResponseTo receives the flag indicating whether SAML response is in response to an authn request (i.e. SP-initiated SSO) or not (i.e. IdP-initiated SSO).

The partnerIdP receives the name of the identity provider.

The userName receives the name of the user.

The attributes receives the user's optional attribute names and values.

The targetUrl receives the target service provider URL or null if the default page should be displayed.

### 5.2.3 SAMLServiceProvider.InitiateSLO

The InitiateSLO method sends a logout request to the identity provider as part of SP-initiated SLO.

For example:

```
SAMLServiceProvider.InitiateSLO(  
    Response,  
    null,  
    null);
```

The Response is used to send the logout request to the service provider via the browser.

The second parameter is the logout reason or null if none.

The third parameter is the partner identity provider's name or null if there's only one configured partner identity provider.

### 5.2.4 SAMLServiceProvider.ReceiveSLO

The ReceiveSLO method receives a logout request from a service provider as part of SP-initiated SLO or a logout response from a service provider as part of IdP-initiated SLO.

For example:

```
SAMLServiceProvider.ReceiveSLO(
    Request,
    out isRequest,
    out logoutReason,
    out partnerIdP);
```

The Request is used to receive the logout message.

The isRequest receives the flag indicating whether a logout request or response has been received.

The logoutReason receives the logout reason.

The partnerIdP receives the name of the identity provider that sent the logout message.

### 5.2.5 SAMLServiceProvider.SendSLO

The SendSLO method sends a logout message to the identity provider.

For example:

```
SAMLServiceProvider.SendSLO(
    Response,
    null);
```

The Response is used to send the logout message to the identity provider via the browser.

The second parameter is the error message or null if none.

## 5.3 Configuration Management

The SAML configuration is defined in section 6.

For the majority of use cases, maintaining the SAML configuration in the saml.config configuration file, located in the application directory, is the simplest strategy.

The SAMLConfigFile application setting in web.config may be used to specify an alternative SAML configuration file. For example, this may be useful to distinguish between test and production builds.

```
<appSettings>
  <!-- The relative or absolute path of the SAML configuration file.
       It defaults to saml.config. -->
  <add key="SAMLConfigFile" value="prod-saml.config"/>
</appSettings>
```

SAML configuration files are loaded automatically.

Alternatively, configuration may be specified programmatically if, for example, it's stored in a database rather than the `saml.config` file. Refer to section 6.10 for more details.

## 5.4 *ICertificateManager* Interface

Both local and partner X.509 certificates may be specified by configuration. See section 6 for more details. For most uses cases, this is the preferred method. However, if required, certificates may be managed programmatically through the *ICertificateManager* interface.

The *ICertificateManager* interface permits retrieval of X.509 certificates required as part of the single sign-on process.

The `GetLocalIdentityProviderCertificate` method returns the local identity provider certificate associated with the specified partner service provider.

The `GetLocalServiceProviderCertificate` method returns the local service provider certificate associated with the specified partner identity provider.

The `GetPartnerIdentityProviderCertificate` method returns the specified partner identity provider's certificate.

The `GetPartnerServiceProviderCertificate` method returns the specified partner service provider's certificate.

### 5.4.1 *CertificateManager*

A default implementation, *CertificateManager*, is included which supports X.509 certificates specified by configuration.

### 5.4.2 Custom *ICertificateManager*

If required, a custom *ICertificateManager* may be implemented.

The following example code outlines a custom certificate manager which retrieves certificates that are stored in a database.

```
public class DatabaseCertificateManager : ICertificateManager
{
    public IList<X509Certificate2>
    GetLocalIdentityProviderSignatureCertificates(string configurationID,
    string partnerServiceProviderName)
    {
        // Load the certificate from the database - details not shown.
        byte[] certificateBytes = null;
        return new List<X509Certificate2>()
        {
            new X509Certificate2(certificateBytes)
        };
    }
}
```

## ComponentSpace SAML v2.0 for .NET Developer Guide

```
public IList<X509Certificate2>
GetLocalServiceProviderEncryptionCertificates(string configurationID,
string partnerIdentityProviderName)
{
    // Load the certificate from the database - details not shown.
    byte[] certificateBytes = null;

    return new List<X509Certificate2>()
    {
        new X509Certificate2(certificateBytes)
    };
}

public IList<X509Certificate2>
GetLocalServiceProviderSignatureCertificates(string configurationID,
string partnerIdentityProviderName)
{
    // Load the certificate from the database - details not shown.
    byte[] certificateBytes = null;

    return new List<X509Certificate2>()
    {
        new X509Certificate2(certificateBytes)
    };
}

public IList<X509Certificate2>
GetPartnerIdentityProviderSignatureCertificates(string configurationID,
string partnerIdentityProviderName)
{
    // Load the certificate from the database - details not shown.
    // The partnerIdentityProviderName is used as a key to retrieve
    // the certificate.
    byte[] certificateBytes = null;

    return new List<X509Certificate2>()
    {
        new X509Certificate2(certificateBytes)
    };
}

public IList<X509Certificate2>
GetPartnerServiceProviderEncryptionCertificates(string configurationID,
string partnerServiceProviderName)
{
    // Load the certificate from the database - details not shown.
    // The partnerServiceProviderName is used as a key to retrieve
    // the certificate.
    byte[] certificateBytes = null;

    return new List<X509Certificate2>()
    {
        new X509Certificate2(certificateBytes)
    };
}
```

```

public IList<X509Certificate2>
GetPartnerServiceProviderSignatureCertificates(string configurationID,
string partnerServiceProviderName)
{
    // Load the certificate from the database - details not shown.
    // The partnerServiceProviderName is used as a key to retrieve
    // the certificate.
    byte[] certificateBytes = null;

    return new List<X509Certificate2>()
    {
        new X509Certificate2(certificateBytes)
    };
}
}

```

The following code assume a `DatabaseCertificateManager` class has been implemented and configures this as the certificate manager.

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

## 5.5 IIDCache Interface

The IIDCache interface manages identifiers used as part of the single sign-on process. This includes checking SAML assertion identifiers as part of replay attack detection at the service provider.

### 5.5.1 InMemoryIDCache

A default implementation, `InMemoryIDCache`, is included.

The `InMemoryIDCache` stores identifiers in an in-memory cache which is suitable in a single server configuration but is not suitable in a web farm.

### 5.5.2 DatabaseIDCache

In a web farm, an IIDCache implementation backed by a database, for example, is required.

The `DatabaseIDCache` stores identifiers in a database table which is suitable in a web farm.

The following code configures the database ID cache.

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

The `IDCache` property should be set before any other SAML API calls (eg in the application's `Global.asax`).

The ID cache is stored in a single table, `SAMLIdentifiers`, with the following schema.

	Column Name	Data Type	Allow Nulls
	ID	nvarchar(256)	<input type="checkbox"/>
	ExpirationDateTime	datetime	<input type="checkbox"/>

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 2014 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.

The high-level API MVC and web forms example service providers include a connection string in web.config for the example database.

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 are available to specify the connection string name and table name.

The DatabaseIDCache class includes a DeleteExpired method that removes any expired IDs from the table. This method should be called periodically to maintain the table.

The StartDeletingExpired method may be called to schedule the removal of expired IDs on a regular basis as a background task.

Alternatively, the DeleteExpiredPriorToAdd property may be set to true to have the removal of expired IDs occur prior to adding a new ID.

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

However it is done, care should be taken to ensure this table doesn't grow too large with expired rows.

### **5.5.3 Custom IDCache**

If required, a custom IIDCache may be implemented.

The following code assume a CustomIDCache class has been implemented and configures this as the ID cache.

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

## **5.6 ISSOSessionStore Interface**

The ISSOSessionStore interface manages the storage of SSO session state information used as part of the single sign-on process.

### **5.6.1 HttpSSOSessionStore**

A default implementation, HttpSSOSessionStore, is included.

The HttpSSOSessionStore stores SSO session state information as part of the ASP.NET HTTP session state.

The HttpSSOSessionStore is suitable in a single server configuration using the ASP.NET InProc session mode or in a web farm using the StateServer, SQLServer or equivalent Custom mode.

### **5.6.2 DatabaseSSOSessionStore**

In a web farm, an ISSOSessionStore implementation backed by a database, for example, is required.

The DatabaseSSOSessionStore stores SSO session data in a database table which is suitable in a web farm.

The following code configures the database SSO session store.

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

The SSOSessionStore property should be set before any other SAML API calls (eg in the application's Global.asax).

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

Column Name	Data Type	Allow Nulls
SessionID	nvarchar(64)	<input type="checkbox"/>
UpdateDateTime	datetime	<input type="checkbox"/>
SessionObject	varbinary(MAX)	<input type="checkbox"/>

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 SAMLIdentifiers table in a SQL Server 2014 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`.

The high-level API MVC and web forms example service providers include a connection string in `web.config` for the example database.

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 are available to specify the connection string name and table name.

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.

However it is done, care should be taken to ensure this table doesn't grow too large with expired rows.

### 5.6.3 Custom `ISSOSessionStore`

If required, a custom `ISSOSessionStore` may be implemented.

The following code assume a `CustomSSOSessionStore` class has been implemented and configures this as the SSO session store.

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

## 6 SAML High Level API Configuration

Configuration information associated with the high level API is found within the `saml.config` file in the application's root directory.

The following XML schema defines the `saml.config` syntax.

Any errors in the configuration will be reported when the configuration is loaded.

The utility application, `ValidateConfig.exe`, which is described in section 11.13.1, may be used to validate the configuration.

## ComponentSpace SAML v2.0 for .NET Developer Guide

```
<?xml version="1.0" encoding="utf-8"?>
<!-- This schema defines the SAML configuration syntax. -->
<schema targetNamespace="urn:componentspace:SAML:2.0:configuration"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:saml="urn:componentspace:SAML:2.0:configuration"
  elementFormDefault="qualified">

  <!-- SAML configurations -->
  <element name="SAMLConfigurations"
    type="saml:SAMLConfigurationsType"/>

  <complexType name="SAMLConfigurationsType">
    <sequence>
      <element name="SAMLConfiguration"
        type="saml:SAMLConfigurationType"
        minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="ReloadOnConfigurationChange" type="boolean"
      default="true"/>
    <attribute name="TraceLevel" type="saml:TraceLevelType"/>
    <attribute name="ValidateMessagesAgainstSchema" type="boolean"
      default="false"/>
  </complexType>

  <!-- SAML configuration -->
  <element name="SAMLConfiguration"
    type="saml:SAMLConfigurationType"/>

  <complexType name="SAMLConfigurationType">
    <all>
      <element name="IdentityProvider"
        type="saml:IdentityProviderType" minOccurs="0"/>
      <element name="ServiceProvider" type="saml:ServiceProviderType"
        minOccurs="0"/>
      <element name="PartnerIdentityProviders"
        type="saml:PartnerIdentityProvidersType" minOccurs="0"
        maxOccurs="unbounded"/>
      <element name="PartnerServiceProviders"
        type="saml:PartnerServiceProvidersType" minOccurs="0"
        maxOccurs="unbounded"/>
    </all>
    <attribute name="ReloadOnConfigurationChange" type="boolean"
      default="true"/>
    <attribute name="TraceLevel" type="saml:TraceLevelType"/>
  </complexType>

  <!-- Identity Provider -->
  <complexType name="IdentityProviderType">
    <complexContent>
      <extension base="saml:LocalProviderType">
        <attribute name="SingleSignOnServiceUrl" type="string"/>
      </extension>
    </complexContent>
  </complexType>
```

## ComponentSpace SAML v2.0 for .NET Developer Guide

```
<!-- Service Provider -->
<complexType name="ServiceProviderType">
  <complexContent>
    <extension base="saml:LocalProviderType">
      <attribute name="AssertionConsumerServiceUrl" type="string"/>
    </extension>
  </complexContent>
</complexType>

<!-- Partner Identity Providers -->
<complexType name="PartnerIdentityProvidersType">
  <sequence>
    <element name="PartnerIdentityProvider"
      type="saml:PartnerIdentityProviderType"
      minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>

<!-- Partner Service Providers -->
<complexType name="PartnerServiceProvidersType">
  <sequence>
    <element name="PartnerServiceProvider"
      type="saml:PartnerServiceProviderType"
      minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>

<!-- Partner Identity Provider -->
<complexType name="PartnerIdentityProviderType">
  <complexContent>
    <extension base="saml:PartnerProviderType">
      <attribute name="SingleSignOnServiceUrl" type="string"
        use="required"/>
      <attribute name="SingleSignOnServiceBinding"
        type="saml:SAMLBindingType"
        default="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
Redirect"/>
      <attribute name="SignAuthnRequest" type="boolean"
        default="false"/>
      <attribute name="ForceAuthn" type="boolean"
        default="false"/>
      <attribute name="WantSAMLResponseSigned" type="boolean"
        default="false"/>
      <attribute name="WantAssertionSigned" type="boolean"
        default="false"/>
      <attribute name="WantAssertionEncrypted" type="boolean"
        default="false"/>
      <attribute name="DisableAssertionReplayCheck"
        type="boolean" default="false"/>
      <attribute name="DisableRecipientCheck"
        type="boolean" default="false"/>
      <attribute name="DisableTimePeriodCheck"
        type="boolean" default="false"/>
      <attribute name="DisableAudienceRestrictionCheck"
        type="boolean" default="false"/>
      <attribute name="DisableAuthnContextCheck"
        type="boolean" default="false"/>
    </extension>
  </complexContent>
</complexType>
```

## ComponentSpace SAML v2.0 for .NET Developer Guide

```
<attribute name="OverridePendingAuthnRequest"
  type="boolean" default="false"/>
  <attribute name="ProviderName" type="string"/>
</extension>
</complexContent>
</complexType>

<!-- Partner Service Provider -->
<complexType name="PartnerServiceProviderType">
  <complexContent>
    <extension base="saml:PartnerProviderType">
      <attribute name="AssertionConsumerServiceUrl" type="string"/>
      <attribute name="WantAuthnRequestSigned" type="boolean"
        default="false"/>
      <attribute name="SignSAMLResponse" type="boolean"
        default="false"/>
      <attribute name="SignAssertion" type="boolean"
        default="false"/>
      <attribute name="EncryptAssertion" type="boolean"
        default="false"/>
      <attribute name="AssertionLifeTime" type="string"
        default="00:03:00"/>
    </extension>
  </complexContent>
</complexType>

<!-- Local and partner provider types -->
<complexType name="LocalProviderType" abstract="true">
  <complexContent>
    <extension base="saml:ProviderType"/>
  </complexContent>
</complexType>

<complexType name="PartnerProviderType" abstract="true">
  <complexContent>
    <extension base="saml:ProviderType">
      <attribute name="PartnerCertificateFile" type="string"/>
      <attribute name="PartnerCertificateStoreLocation"
        type="saml:CertificateStoreLocationType"
        default="LocalMachine"/>
      <attribute name="PartnerCertificateSerialNumber"
        type="string"/>
      <attribute name="PartnerCertificateThumbprint" type="string"/>
      <attribute name="PartnerCertificateSubject" type="string"/>
      <attribute name="SingleLogoutServiceUrl" type="string"/>
      <attribute name="SingleLogoutServiceResponseUrl"
        type="string"/>
      <attribute name="SingleLogoutServiceBinding"
        type="saml:SAMLBindingType"
        default="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"/>
      <attribute name="LogoutRequestLifeTime" type="string"
        default="00:03:00"/>
      <attribute name="DisableDestinationCheck" type="boolean"
        default="false"/>
      <attribute name="DisableInboundLogout" type="boolean"
        default="false"/>
    </extension>
  </complexContent>
</complexType>
```

## ComponentSpace SAML v2.0 for .NET Developer Guide

```
<attribute name="DisableOutboundLogout" type="boolean"
  default="false"/>
<attribute name="DisableInResponseToCheck" type="boolean"
  default="false"/>
<attribute name="DisablePendingLogoutCheck" type="boolean"
  default="false"/>
<attribute name="SignLogoutRequest" type="boolean"
  default="false"/>
<attribute name="SignLogoutResponse" type="boolean"
  default="false"/>
<attribute name="WantLogoutRequestSigned" type="boolean"
  default="false"/>
<attribute name="WantLogoutResponseSigned" type="boolean"
  default="false"/>
<attribute name="UseEmbeddedCertificate" type="boolean"
  default="false"/>
<attribute name="IssuerFormat" type="saml:NameIDFormatType"/>
<attribute name="NameIDFormat" type="saml:NameIDFormatType"
  default="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"/>
<attribute name="DigestMethod"
  type="saml:DigestMethodType"
  default="http://www.w3.org/2000/09/xmlldsig#sha1"/>
<attribute name="SignatureMethod"
  type="saml:SignatureMethodType"
  default="http://www.w3.org/2000/09/xmlldsig#rsa-sha1"/>
<attribute name="KeyEncryptionMethod"
  type="saml:KeyEncryptionMethodType"
  default="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
<attribute name="DataEncryptionMethod"
  type="saml:DataEncryptionMethodType"
  default="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
<attribute name="ClockSkew" type="string" default="00:00:00"/>
<attribute name="AuthnContext" type="saml:AuthnContextType"/>
</extension>
</complexContent>

<complexType name="ProviderType" abstract="true">
  <attribute name="Name" type="string" use="required"/>
  <attribute name="LocalCertificateFile" type="string"/>
  <attribute name="LocalCertificatePassword" type="string"/>
  <attribute name="LocalCertificatePasswordKey" type="string"/>
  <attribute name="LocalCertificateStoreLocation"
    type="saml:CertificateStoreLocationType"
    default="LocalMachine"/>
  <attribute name="LocalCertificateSerialNumber" type="string"/>
  <attribute name="LocalCertificateThumbprint" type="string"/>
  <attribute name="LocalCertificateSubject" type="string"/>
</complexType>

<!-- Name ID Formats -->
<simpleType name="NameIDFormatType">
  <restriction base="string">
    <enumeration value="urn:oasis:names:tc:SAML:1.1:nameid-
  format:unspecified"/>
    <enumeration value="urn:oasis:names:tc:SAML:1.1:nameid-
  format:emailAddress"/>
  </restriction>
</simpleType>
```

## ComponentSpace SAML v2.0 for .NET Developer Guide

```
<enumeration value="urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName"/>
  <enumeration value="urn:oasis:names:tc:SAML:1.1:nameid-format:WindowsDomainQualifiedName"/>
  <enumeration value="urn:oasis:names:tc:SAML:2.0:nameid-format:kerberos"/>
  <enumeration value="urn:oasis:names:tc:SAML:2.0:nameid-format:entity"/>
  <enumeration value="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"/>
  <enumeration value="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"/>
</restriction>
</simpleType>

<!-- Authn contexts -->
<simpleType name="AuthnContextType">
  <restriction base="string">
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:InternetProtocol"/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:InternetProtocolPassword"
"/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:Kerberos"/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:MobileOneFactorUnregiste
red"/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:MobileTwoFactorUnregiste
red"/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:MobileOneFactorContract"
/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:MobileTwoFactorContract"
/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:Password"/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTranspo
rt"/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:PreviousSession"/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:X509"/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:PGP"/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:SPKI"/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:XMLDSig"/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:Smartcard"/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:SmartcardPKI"/>
```

## ComponentSpace SAML v2.0 for .NET Developer Guide

```
<enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:SoftwarePKI"/>
  <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:Telephony"/>
    <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:NomadTelephony"/>
      <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:PersonalTelephony"/>
        <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:AuthenticatedTelephony"/
>
          <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:SecureRemotePassword"/>
            <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:TLSClient"/>
              <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:TimeSyncToken"/>
                <enumeration
value="urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified"/>
              </restriction>
            </simpleType>

    <!-- Bindings -->
    <simpleType name="SAMLBindingType">
      <restriction base="string">
        <enumeration value="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
POST"/>
        <enumeration value="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
Redirect"/>
      </restriction>
    </simpleType>

    <!-- Security -->
    <simpleType name="KeyEncryptionMethodType">
      <restriction base="string">
        <enumeration value="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        <enumeration value="http://www.w3.org/2001/04/xmlenc#rsa-oaep-
mgflp"/>
      </restriction>
    </simpleType>

    <simpleType name="DataEncryptionMethodType">
      <restriction base="string">
        <enumeration value="http://www.w3.org/2001/04/xmlenc#tripleDES-
cbc"/>
        <enumeration value="http://www.w3.org/2001/04/xmlenc#aes128-
cbc"/>
        <enumeration value="http://www.w3.org/2001/04/xmlenc#aes192-
cbc"/>
        <enumeration value="http://www.w3.org/2001/04/xmlenc#aes256-
cbc"/>
      </restriction>
    </simpleType>

    <simpleType name="DigestMethodType">
      <restriction base="string">
        <enumeration value="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

```

    <enumeration value="http://www.w3.org/2001/04/xmlenc#sha256"/>
  </restriction>
</simpleType>

<simpleType name="SignatureMethodType">
  <restriction base="string">
    <enumeration value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <enumeration value="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
  </restriction>
</simpleType>

<!-- Trace -->
<simpleType name="TraceLevelType">
  <restriction base="string">
    <enumeration value="Off"/>
    <enumeration value="Verbose"/>
  </restriction>
</simpleType>
</schema>

```

**Figure 7 Saml.config XML Schema**

## 6.1 SAML Configurations

### SAMLConfiguration

The SAMLConfiguration element specifies one or more SAML configurations when configured in a multi-tenanted environment.

### ReloadOnConfigurationChange

The optional ReloadOnConfigurationChange attribute specifies whether the application should be reloaded if the configuration changes. The default is true.

### TraceLevel

The optional TraceLevel attribute specifies the trace level for logging.

Tracing configuration is specified in the application's web.config. Refer to section 15.1 for more details. The trace switch value in the <system.diagnostics> section specifies whether verbose trace is enabled or not.

The trace switch may be overridden by specifying a trace level. For example, if the trace switch value in <system.diagnostics> is Off, the trace level may be set to Verbose to enable trace without modifying web.config.

In most scenarios, it's preferable to modify the trace switch value in <system.diagnostics>.

By default, the trace level is not specified and therefore the trace switch value in <system.diagnostics> applies.

### ValidateMessagesAgainstSchema

The optional `ValidateMessagesAgainstSchema` attribute specifies whether received SAML messages should be validated against the SAML XML schemas. The default is false.

## **6.2 SAML Configuration**

### **IdentityProvider**

The optional `IdentityProvider` element specifies the configuration for the application when acting as an identity provider.

### **ServiceProvider**

The optional `ServiceProvider` element specifies the configuration for the application when acting as a service provider.

### **PartnerIdentityProviders**

The optional `PartnerIdentityProviders` element specifies the configuration for the partner identity providers.

### **PartnerServiceProviders**

The optional `PartnerServiceProviders` element specifies the configuration for the partner service providers.

### **PartnerIdentityProvider**

The optional `PartnerIdentityProvider` element specifies the configuration for a partner identity provider.

### **PartnerServiceProvider**

The optional `PartnerServiceProvider` element specifies the configuration for a partner service provider.

### **ID**

The optional `ID` attribute specifies the SAML configuration identifier in a multi-tenanted environment. If there is a single SAML configuration only, no identifier needs to be specified.

## **6.3 Identity Provider Configuration**

### **SingleSignOnServiceUrl**

The optional `SingleSignOnServiceUrl` attribute specifies the application's single sign-on service URL. SAML authn requests will be received at this URL.

## **6.4 Service Provider Configuration**

### **AssertionConsumerServiceUrl**

The optional `AssertionConsumerServiceUrl` attribute specifies the application's assertion consumer service (ACS) URL. SAML responses will be received at the ACS.

## **6.5 Partner Identity Provider Configuration**

### **SingleSignOnServiceUrl**

The optional SingleSignOnServiceUrl attribute specifies the partner identity provider's single sign-on (SSO) service URL. Authentication requests will be sent to the SSO service. This is only required for SP-initiated SSO.

### **SingleSignOnServiceBinding**

The optional SingleSignOnServiceBinding attribute specifies the transport binding to use when sending authentication requests to the partner identity provider's SSO service. The default is to use the HTTP-Redirect binding.

### **SignAuthnRequest**

The optional SignAuthnRequest attribute specifies whether authentication requests sent to the partner identity provider should be signed. The default is false.

### **ForceAuthn**

The optional ForceAuthn attribute specifies whether to set the force authentication attribute in authentication requests. The default is false.

### **WantSAMLResponseSigned**

The optional WantSAMLResponseSigned attribute specifies whether the SAML response from the partner identity provider should be signed. The default is false.

### **WantAssertionSigned**

The optional WantAssertionSigned attribute specifies whether the SAML assertion from the partner identity provider should be signed. The default is false.

### **WantAssertionEncrypted**

The optional WantAssertionEncrypted attribute specifies whether the SAML assertion from the partner identity provider should be encrypted. The default is false.

### **DisableAssertionReplayCheck**

The optional DisableAssertionReplayCheck attribute specifies whether the check for SAML assertion replay should be made. This attribute should only be set to true in test environments or to work around limitations in the partner identity provider. The default is false.

### **DisableRecipientCheck**

The optional DisableRecipientCheck attribute specifies whether the check SAML recipient URI in the SAML assertion should be checked. This attribute should only be set to true in test environments or to work around limitations in the partner identity provider. The default is false.

### **DisableTimePeriodCheck**

The optional DisableTimePeriodCheck attribute specifies whether the SAML assertion's not-before/not-on-or-after time period should be checked. This attribute

should only be set to true in test environments or to work around limitations in the partner identity provider. The default is false.

#### **DisableAudienceRestrictionCheck**

The optional `DisableAudienceRestrictionCheck` attribute specifies whether the audience restriction condition in the SAML assertion should be checked. This attribute should only be set to true in test environments or to work around limitations in the partner identity provider. The default is false.

#### **DisableAuthnContextCheck**

The optional `DisableAuthnContextCheck` attribute specifies whether the authentication context in the authentication statement in the SAML assertion should be checked against the `AuthnContext` configuration attribute. This attribute should only be set to true in test environments or to work around limitations in the partner identity provider. The default is false.

#### **OverridePendingAuthnRequest**

The optional `OverridePendingAuthnRequest` attribute specifies whether a pending authentication request may be overridden and an IdP-initiated SAML response received. Setting this flag to true supports an SP-initiated SSO flow being supplanted by an IdP-initiated SSO. The default is false.

#### **ProviderName**

The optional `ProviderName` attribute specifies the provider name to include in authentication requests sent to the partner identity provider. The default is none.

## **6.6 Partner Service Provider Configuration**

#### **AssertionConsumerServiceUrl**

The optional `AssertionConsumerServiceUrl` attribute specifies the partner service provider's assertion consumer service (ACS) URL. SAML responses will be sent to the ACS.

An `AssertionConsumerServiceUrl` must be configured for IdP-initiated SSO.

For SP-initiated SSO, the assertion consumer service URL included in the authn request from the service provider will be used. If no assertion consumer service URL is included in the authn request then the `AssertionConsumerServiceUrl` must be configured for SP-initiated SSO.

#### **WantAuthnRequestSigned**

The optional `WantAuthnRequestSigned` attribute specifies whether the authentication request from the partner service provider should be signed. The default is false.

#### **SignSAMLResponse**

The optional `SignSAMLResponse` attribute specifies whether SAML responses sent to the partner service provider should be signed. The default is false.

### **SignAssertion**

The optional SignAssertion attribute specifies whether SAML assertions sent to the partner service provider should be signed. The default is false.

### **EncryptAssertion**

The optional EncryptAssertion attribute specifies whether SAML assertions sent to the partner service provider should be encrypted. The default is false.

### **AssertionLifeTime**

The optional AssertionLifeTime attribute specifies the NotBefore/NotOnOrAfter time interval for the SAML assertion. The format is hh:mm:ss. The default is 3 minutes.

## **6.7 Partner Provider Configuration**

### **PartnerCertificateFile**

The optional PartnerCertificateFile attribute specifies the X.509 certificate file for this provider. The certificate file name may be an absolute path or a path relative to the application folder.

### **PartnerCertificateStoreLocation**

The optional PartnerCertificateStoreLocation attribute specifies the X.509 certificate store (LocalMachine or CurrentUser). The default is local machine.

### **PartnerCertificateSerialNumber**

The optional PartnerCertificateSerialNumber attribute specifies the X.509 certificate by serial number for this provider. The certificate is retrieved from the local computer's X.509 certificate store.

### **PartnerCertificateThumbprint**

The optional PartnerCertificateThumbprint attribute specifies the X.509 certificate by thumbprint for this provider. The certificate is retrieved from the local computer's X.509 certificate store.

### **PartnerCertificateSubject**

The optional PartnerCertificateSubject attribute specifies the X.509 certificate by subject name for this provider. The certificate is retrieved from the local computer's X.509 certificate store.

### **SingleLogoutServiceUrl**

The SingleLogoutServiceUrl attribute specifies the partner provider's single logout (SLO) service URL. Logout requests will be sent to the SLO service.

### **SingleLogoutServiceResponseUrl**

The SingleLogoutServiceResponseUrl attribute specifies the partner provider's single logout (SLO) service response URL. Logout responses will be sent to the

SLO response service. If not specified then logout responses are sent to the same URL as logout requests.

### **SingleLogoutServiceBinding**

The optional SingleLogoutServiceBinding attribute specifies the transport binding to use when sending logout messages to the partner provider's SLO service. The default is to use the HTTP-Redirect binding.

### **LogoutRequestLifeTime**

The optional LogoutRequestLifeTime attribute specifies the NotOnOrAfter time interval for the logout request. The format is hh:mm:ss. The default is 3 minutes.

### **DisableDestinationCheck**

The optional DisableDestinationCheck attribute specifies whether the SAML message's Destination should be checked. This attribute should only be set to true in test environments or to work around limitations in the partner provider. The default is false.

### **DisableInboundLogout**

The optional DisableInboundLogout attribute specifies whether logout requests sent by the partner provider are not supported. The default is false.

### **DisableOutboundLogout**

The optional DisableOutboundLogout attribute specifies whether logout requests sent to the partner provider are not supported. The default is false.

### **DisableInResponseToCheck**

The optional DisableInResponseToCheck attribute specifies whether the SAML message's InResponseTo should be checked. This attribute should only be set to true in test environments or to work around limitations in the partner provider. The default is false.

### **DisablePendingLogoutCheck**

The optional DisablePendingLogoutCheck attribute specifies whether a check for a pending SAML response should be performed. This attribute should only be set to true in test environments or to work around limitations in the partner provider. The default is false.

### **SignLogoutRequest**

The optional SignLogoutRequest attribute specifies whether logout requests sent to the partner provider should be signed. The default is false.

### **SignLogoutResponse**

The optional SignLogoutResponse attribute specifies whether logout responses sent to the partner provider should be signed. The default is false.

### **WantLogoutRequestSigned**

The optional `WantLogoutRequestSigned` attribute specifies whether the logout request from the partner provider should be signed. The default is false.

### **WantLogoutResponseSigned**

The optional `WantLogoutResponseSigned` attribute specifies whether the logout response from the partner provider should be signed. The default is false.

### **UseEmbeddedCertificate**

The optional `UseEmbeddedCertificate` attribute specifies whether the certificate embedded in the XML signature should be used when verifying the signature. If false then a configured certificate retrieved from the certificate manager is used. The default is false.

### **IssuerFormat**

The optional `IssuerFormat` attribute specifies the issuer format to include in SAML messages and assertions sent to the partner provider.

The default is to not include a format attribute with the issuer.

Refer to section 6.9.1 for a list of name ID formats.

### **NameIDFormat**

The optional `NameIDFormat` attribute specifies the name identifier format to include in SAML assertions sent to the partner service provider or in authn requests sent to the partner identity provider.

The default is `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`.

Refer to section 6.9.1 for a list of name ID formats.

### **DigestMethod**

The optional `DigestMethod` attribute specifies the XML signature digest method.

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

Refer to section 6.9.6 for a list of digest methods.

### **SignatureMethod**

The optional `SignatureMethod` attribute specifies the XML signature method.

The default is `"http://www.w3.org/2000/09/xmlsig#rsa-sha1"`.

Refer to section 6.9.7 for a list of signature methods.

### **KeyEncryptionMethod**

The optional `KeyEncryptionMethod` attribute specifies the XML encryption key encryption method.

The default is `http://www.w3.org/2001/04/xmlenc#rsa-1_5`.

Refer to section 6.9.4 for a list of key encryption methods.

### **DataEncryptionMethod**

The optional `DataEncryptionMethod` attribute specifies the XML encryption data encryption method.

The default is <http://www.w3.org/2001/04/xmlenc#aes128-cbc>.

Refer to section 6.9.5 for a list of data encryption methods.

### **ClockSkew**

The optional `ClockSkew` attribute specifies the time span to allow for differences between local and partner computer clocks when checking time intervals. The default is no clock skew.

### **AuthnContext**

For a partner identity provider, the optional `AuthnContext` attribute specifies the requested authentication context to include in authentication requests and the expected authentication context returned in SAML assertions.

For a partner service provider, the optional `AuthnContext` attribute specifies the authentication context to include in SAML assertions.

The default is none.

Refer to section 6.9.2 for a list of authentication contexts.

## **6.8 Provider Configuration**

### **Name**

The `Name` attribute specifies the name of the provider.

### **LocalCertificateFile**

The optional `LocalCertificateFile` attribute specifies the X.509 certificate file for this provider. The certificate file name may be an absolute path or a path relative to the application folder.

### **LocalCertificatePassword**

The optional `LocalCertificatePassword` attribute specifies the password associated with the X.509 certificate file for this provider.

Certificate files (\*.pfx) that include the private key should be protected by a password.

Certificate files (\*.cer) that do not include a private key are not password protected.

The certificate password must be kept secure. In a test environment using a test certificate, specifying the password using the `LocalCertificatePassword` attribute is acceptable.

For a production certificate, the password should be stored encrypted in `web.config`. Refer to the `LocalCertificatePasswordKey` attribute for more details.

### **LocalCertificatePasswordKey**

The optional `LocalCertificatePasswordKey` attribute specifies the `web.config`'s `appSettings` key for the certificate file password.

For example, if the LocalCertificatePasswordKey attribute value is localCertificatePassword, then under the web.config's appSettings section, an entry with the key name localCertificatePassword is expected and the entry value is used as the password.

By encrypting the appSettings section using the aspnet\_regiis utility, the certificate file password is secured.

### **LocalCertificateStoreLocation**

The optional LocalCertificateStoreLocation attribute specifies the X.509 certificate store (LocalMachine or CurrentUser). The default is local machine.

### **LocalCertificateSerialNumber**

The optional LocalCertificateSerialNumber attribute specifies the X.509 certificate by serial number for this provider. The certificate is retrieved from the local computer's X.509 certificate store.

### **LocalCertificateThumbprint**

The optional LocalCertificateThumbprint attribute specifies the X.509 certificate by thumbprint for this provider. The certificate is retrieved from the local computer's X.509 certificate store.

### **LocalCertificateSubject**

The optional LocalCertificateSubject attribute specifies the X.509 certificate by subject name for this provider. The certificate is retrieved from the local computer's X.509 certificate store.

## **6.9 Miscellaneous Configuration**

### **6.9.1 Name ID Format Types**

The supported name ID format types are:

- urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified
- urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress
- urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName
- urn:oasis:names:tc:SAML:1.1:nameid-format:WindowsDomainQualifiedName
- urn:oasis:names:tc:SAML:2.0:nameid-format:Kerberos
- urn:oasis:names:tc:SAML:2.0:nameid-format:entity
- urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
- urn:oasis:names:tc:SAML:2.0:nameid-format:transient

### **6.9.2 Authentication Contexts**

The supported authentication contexts are:

- urn:oasis:names:tc:SAML:2.0:ac:classes:InternetProtocol
- urn:oasis:names:tc:SAML:2.0:ac:classes:InternetProtocolPassword

- urn:oasis:names:tc:SAML:2.0:ac:classes:Kerberos
- urn:oasis:names:tc:SAML:2.0:ac:classes:MobileOneFactorUnregistered
- urn:oasis:names:tc:SAML:2.0:ac:classes:MobileTwoFactorUnregistered
- urn:oasis:names:tc:SAML:2.0:ac:classes:MobileOneFactorContract
- urn:oasis:names:tc:SAML:2.0:ac:classes:MobileTwoFactorContract
- urn:oasis:names:tc:SAML:2.0:ac:classes>Password
- urn:oasis:names:tc:SAML:2.0:ac:classes>PasswordProtectedTransport
- urn:oasis:names:tc:SAML:2.0:ac:classes:PreviousSession
- urn:oasis:names:tc:SAML:2.0:ac:classes:X509
- urn:oasis:names:tc:SAML:2.0:ac:classes:PGP
- urn:oasis:names:tc:SAML:2.0:ac:classes:SPKI
- urn:oasis:names:tc:SAML:2.0:ac:classes:XMLDSig
- urn:oasis:names:tc:SAML:2.0:ac:classes:Smartcard
- urn:oasis:names:tc:SAML:2.0:ac:classes:SmartcardPKI
- urn:oasis:names:tc:SAML:2.0:ac:classes:SoftwarePKI
- urn:oasis:names:tc:SAML:2.0:ac:classes:Telephony
- urn:oasis:names:tc:SAML:2.0:ac:classes:NomadTelephony
- urn:oasis:names:tc:SAML:2.0:ac:classes:PersonalTelephony
- urn:oasis:names:tc:SAML:2.0:ac:classes:AuthenticatedTelephony
- urn:oasis:names:tc:SAML:2.0:ac:classes:SecureRemotePassword
- urn:oasis:names:tc:SAML:2.0:ac:classes:TLSClient
- urn:oasis:names:tc:SAML:2.0:ac:classes:TimeSyncToken
- urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified

### 6.9.3 Binding Types

The supported binding types are:

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

### 6.9.4 Key Encryption Methods

The supported key encryption methods are:

- [http://www.w3.org/2001/04/xmlenc#rsa-1\\_5](http://www.w3.org/2001/04/xmlenc#rsa-1_5)
- <http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p>

### 6.9.5 Data Encryption Methods

The supported data encryption methods are:

- <http://www.w3.org/2001/04/xmlenc#tripleDES-cbc>
- <http://www.w3.org/2001/04/xmlenc#aes128-cbc>
- <http://www.w3.org/2001/04/xmlenc#aes192-cbc>

- <http://www.w3.org/2001/04/xmlenc#aes256-cbc>

### 6.9.6 Digest Methods

The supported digest methods are:

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

See section 16.3 for SHA-256, SHA-384 and SHA-512 support requirements.

### 6.9.7 Signature Methods

The supported signature methods are:

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

See section 16.3 for SHA-256, SHA-384 and SHA-512 support requirements.

## 6.10 Specifying Configuration Programmatically

For the majority of use cases, maintaining the SAML configuration in the `saml.config` configuration file is the simplest strategy.

However, there may be circumstances where configuration must be stored elsewhere (e.g. in a database).

Rather than defining configuration in the `saml.config` configuration file, the configuration may be specified programmatically. A good place to do this is in the `Global.Application_Start` method.

For example, the following code configures the local service provider and one partner identity provider.

```
SAMLConfiguration samlConfiguration = new SAMLConfiguration();

samlConfiguration.LocalServiceProviderConfiguration = new
LocalServiceProviderConfiguration() {
    Name = "urn:componentspace:ExampleServiceProvider",
    AssertionConsumerServiceUrl =
"~/SAML/AssertionConsumerService.aspx",
    LocalCertificateFile = "sp.pfx",
    LocalCertificatePassword = "password"
};

samlConfiguration.AddPartnerIdentityProvider(
```

## ComponentSpace SAML v2.0 for .NET Developer Guide

```
new PartnerIdentityProviderConfiguration() {
    Name = "urn:componentspace:ExampleIdentityProvider",
    SignAuthnRequest = false,
    WantSAMLResponseSigned = true,
    WantAssertionSigned = false,
    WantAssertionEncrypted = false,
    SingleSignOnServiceUrl =
"http://localhost/ExampleIdentityProvider/SAML/SSOService.aspx",
    SingleLogoutServiceUrl =
"http://localhost/ExampleIdentityProvider/SAML/SLOService.aspx",
    PartnerCertificateFile = "idp.cer"
});

SAMLController.Configuration = samlConfiguration;
```

And the following code configures the local identity provider and one partner service provider.

```
SAMLConfiguration samlConfiguration = new SAMLConfiguration();

samlConfiguration.LocalIdentityProviderConfiguration =
    new LocalIdentityProviderConfiguration() {
        Name = "urn:componentspace:ExampleIdentityProvider",
        LocalCertificateFile = "idp.pfx",
        LocalCertificatePassword = "password"
    };

samlConfiguration.AddPartnerServiceProvider(
    new PartnerServiceProviderConfiguration() {
        Name = "urn:componentspace:ExampleServiceProvider",
        WantAuthnRequestSigned = false,
        SignSAMLResponse = true,
        SignAssertion = false,
        EncryptAssertion = false,
        AssertionConsumerServiceUrl =
"http://localhost/ExampleServiceProvider/SAML/AssertionConsumerService.
aspx",
        SingleLogoutServiceUrl =
"http://localhost/ExampleServiceProvider/SAML/SLOService.aspx",
        PartnerCertificateFile = "sp.cer"
    });

SAMLController.Configuration = samlConfiguration;
```

## 7 SAML High Level API Certificate Configuration

X.509 certificates are used for XML signatures (see section 3.5.2) and XML encryption (see section 3.5.3).

Certificates may be specified either programmatically (see section 5.2.3) or through configuration (see section 6). For most use cases it's simpler to specify certificates through configuration.

A certificate may be stored in a file or in the Windows certificate store.

The following sections provide instructions of configuring X.509 certificates.

### **7.1 Local Provider Certificate File**

The following configuration entry specifies the local provider's certificate is contained in the file idp.pfx which is protected by password.

```
<IdentityProvider Name="ExampleIdentityProvider"
  LocalCertificateFile="idp.pfx"
  LocalCertificatePassword="password"/>
```

The certificate file path is not an absolute path and therefore is assumed to be relative to the application's folder.

The following configuration entry specifies an absolute file path.

```
<IdentityProvider Name="ExampleIdentityProvider"
  LocalCertificateFile="C:\\Certificates\\idp.pfx"
  LocalCertificatePassword="password"/>
```

### **7.2 Partner Provider Certificate File**

The following configuration entry specifies the partner provider's certificate is contained in the file sp.cer.

```
<PartnerServiceProvider Name="ExampleServiceProvider"
  PartnerCertificateFile="sp.cer"/>
```

The certificate file path is not an absolute path and therefore is assumed to be relative to the application's folder.

No password is required with a partner certificate file as the file does not contain the private key.

The following configuration entry specifies an absolute file path.

```
<PartnerServiceProvider Name=" ExampleServiceProvider"
  PartnerCertificateFile=
  "C:\\Certificates\\sp.cer"/>
```

### **7.3 Encrypting the Certificate File Password**

When using production certificate files the password should be encrypted. To do this, the ability to encrypt sections of web.config is employed.

The following configuration entry specifies the local provider's certificate is contained in the file idp.pfx which is protected by an encrypted password.

```
<IdentityProvider Name="ExampleIdentityProvider"
  LocalCertificateFile="idp.pfx"
  LocalCertificatePasswordKey=
  "certificateFilePassword"/>
```

The application's web.config appSettings section must include the certificate password key.

```
<appSettings>  
  <add key="certificateFilePassword" value="password"/>  
</appSettings>
```

The .NET framework's `aspnet_regiis` utility application is used to encrypt the `appSettings` section. You must be run this utility as an administrator.

```
aspnet_regiis -pef appSettings  
C:\inetpub\wwwroot\ExampleIdentityProvider
```

The `aspnet_regiis` utility also is used to decrypt the section.

```
aspnet_regiis -pdf appSettings  
C:\inetpub\wwwroot\ExampleIdentityProvider
```

The IIS account under which the application runs must be granted permission to decrypt `web.config`.

```
aspnet_regiis -pa NetFrameworkConfigurationKey "IIS_IUSRS"
```

## **7.4 Managing the Windows Certificate Store**

The Microsoft Management Console may be used to install and manage certificates in the Windows certificate store.

### **7.4.1 Running the MMC Certificates Snap-in**

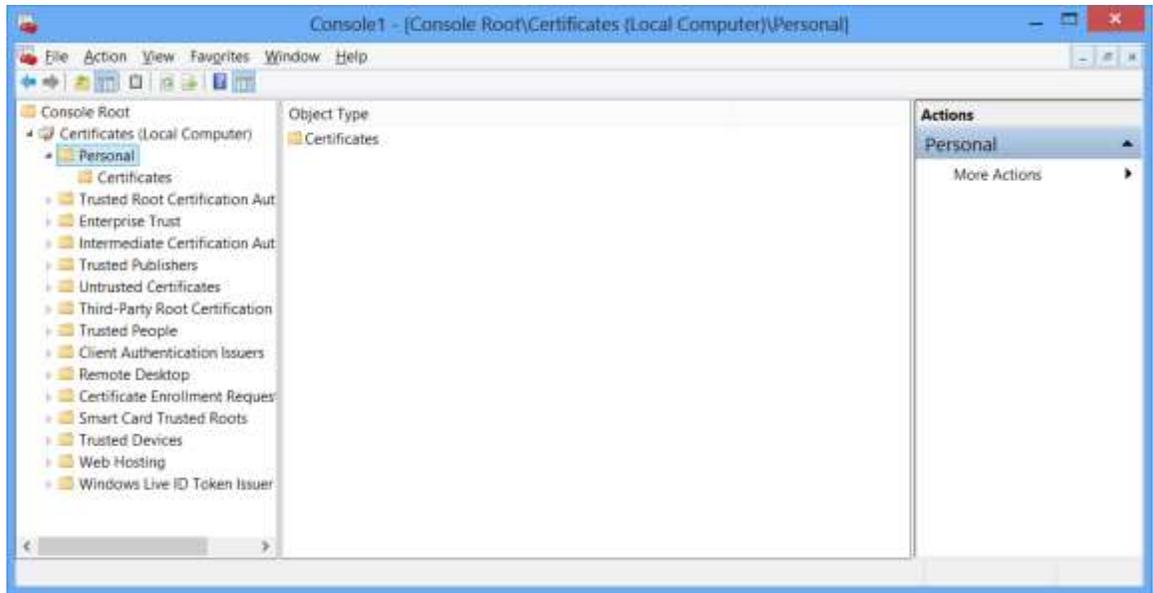
To run the MMC Certificates snap-in:

1. Run the Microsoft Management Console (MMC) as an administrator.
2. From the menu, click File > Add/Remove Snap-in...
3. Select Certificates from the list of available snap-ins.
4. Specify that the computer account for the local computer is to be managed.

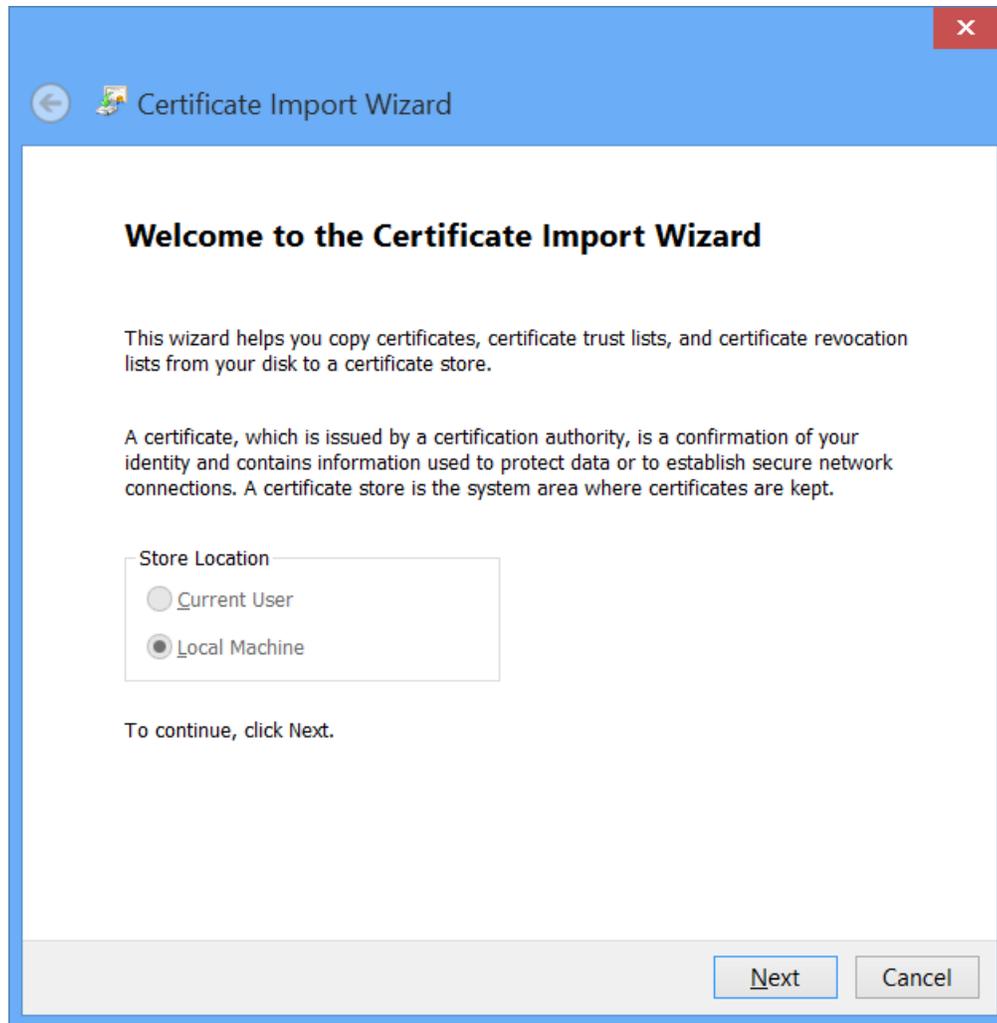
### **7.4.2 Importing a PFX File**

To import a certificate and private key contained in a pfx file into the Windows certificate store:

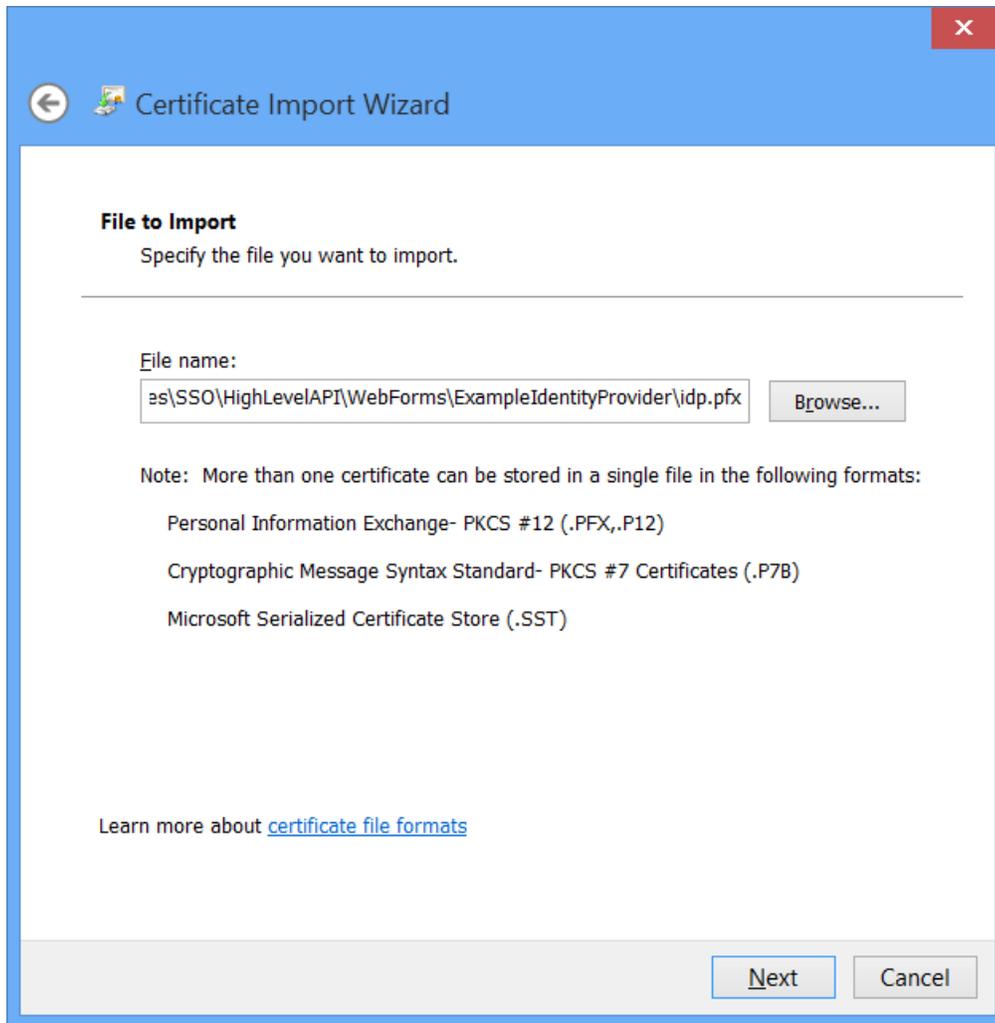
1. Select the Personal folder in the certificates tree.



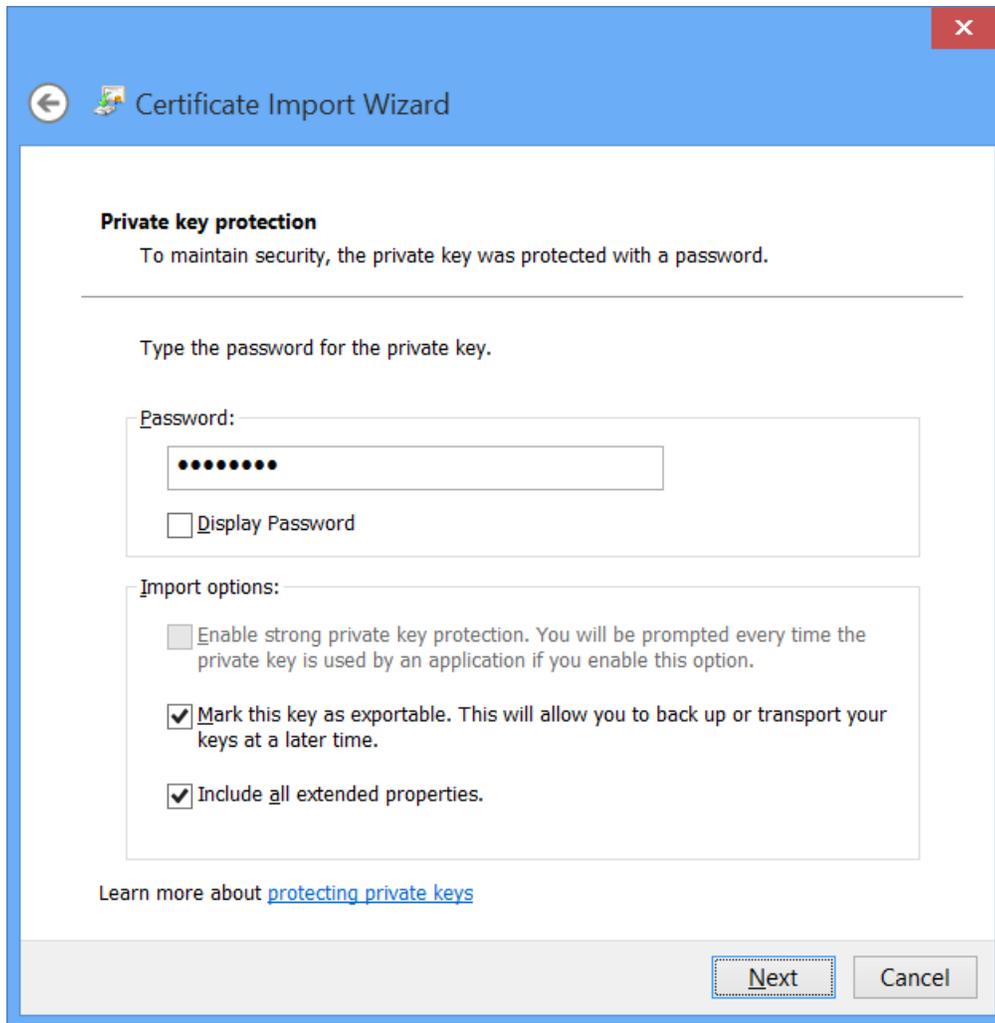
2. From the menu, click Action > All Tasks > Import...



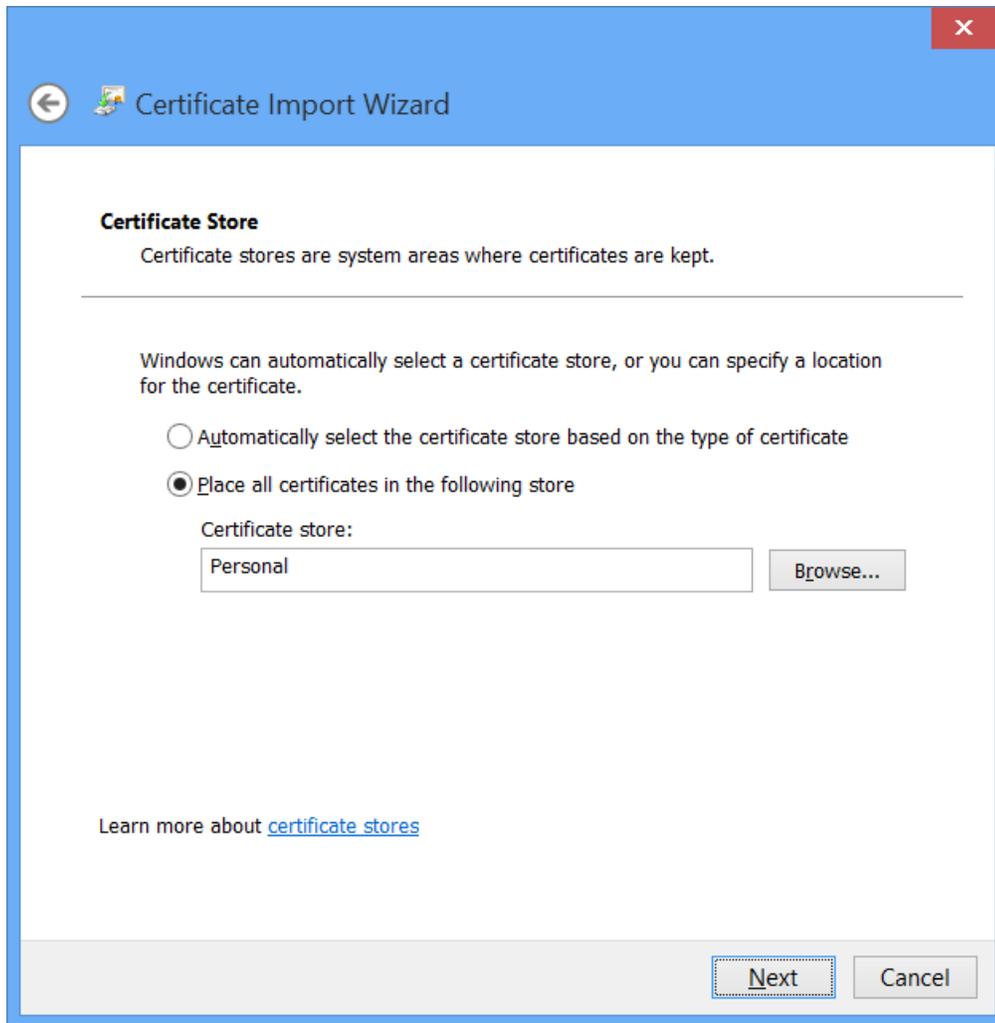
3. Browse to the pfx file to import.



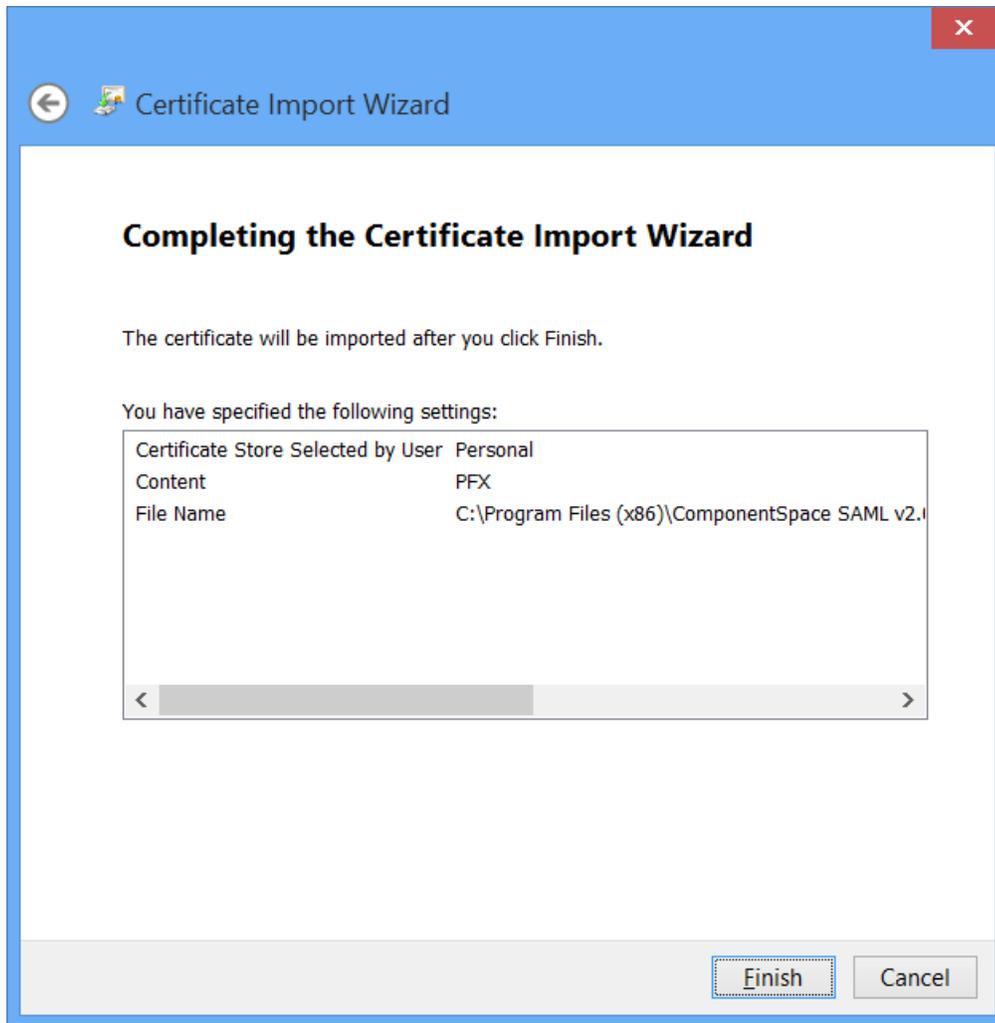
4. Supply the password and optionally check the check box to mark the key as exportable.



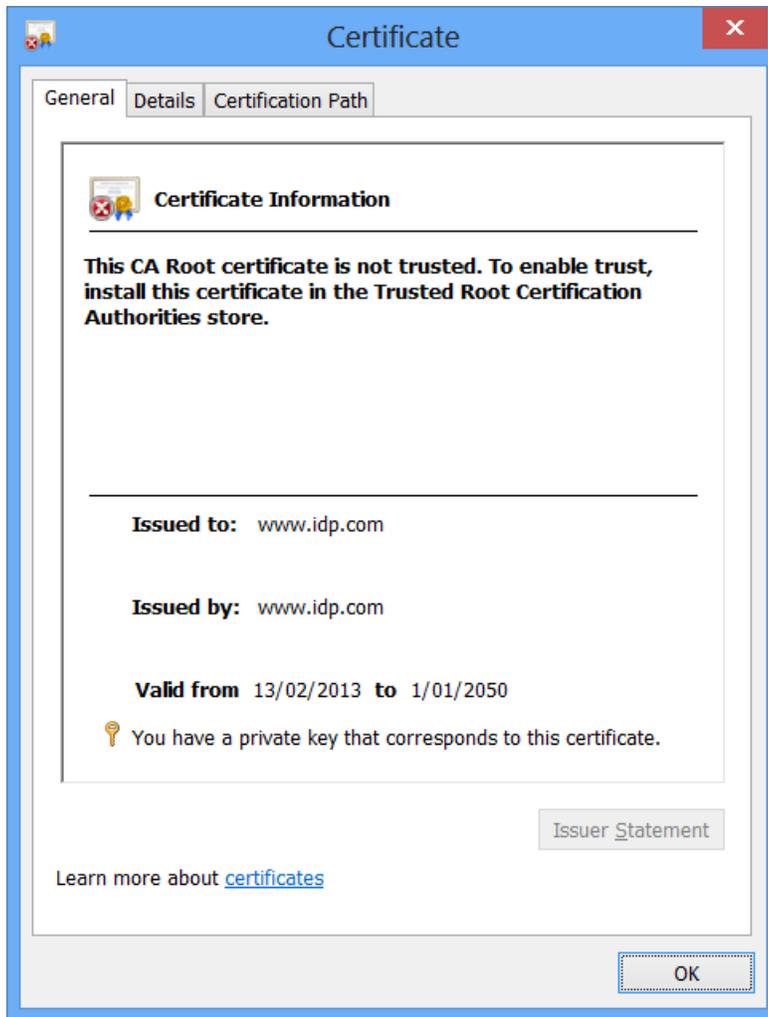
5. Place the certificate in the Personal certificate store.



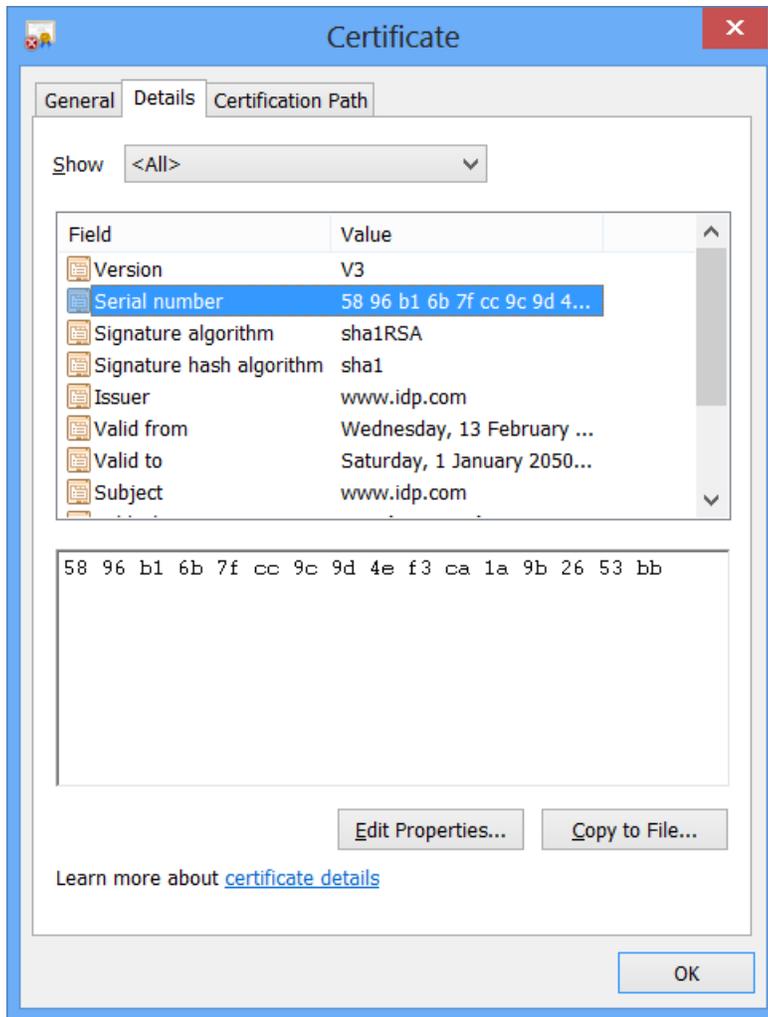
6. Click Finish to complete the import.



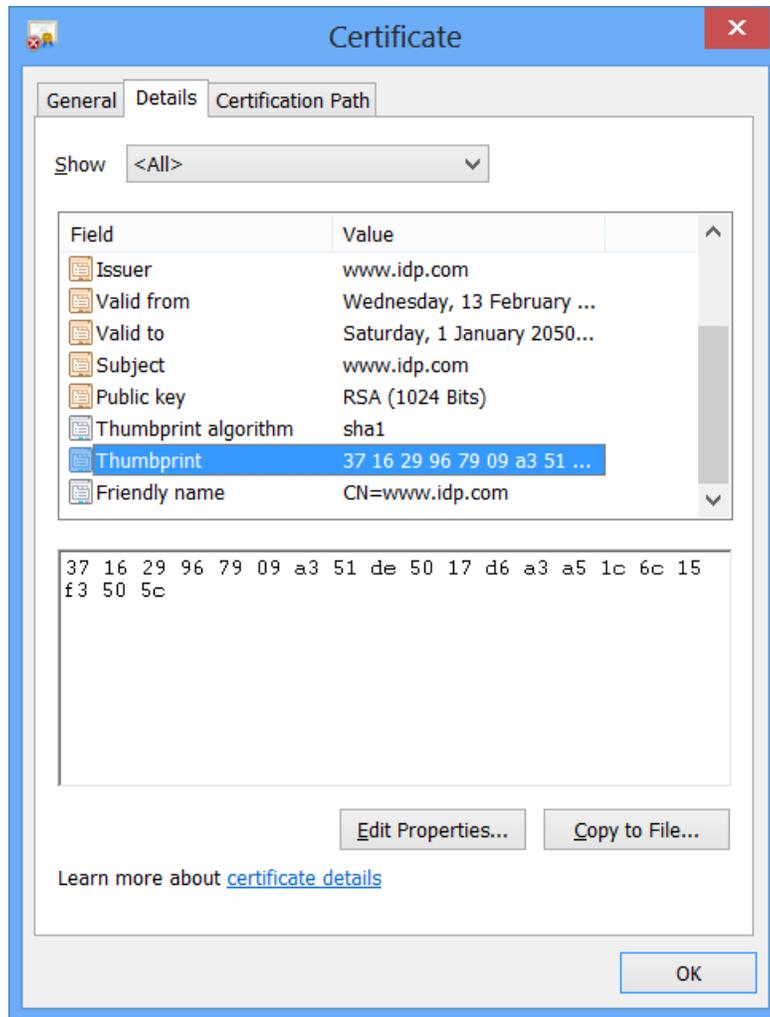
7. Confirm the certificate is listed and open it to review its details.



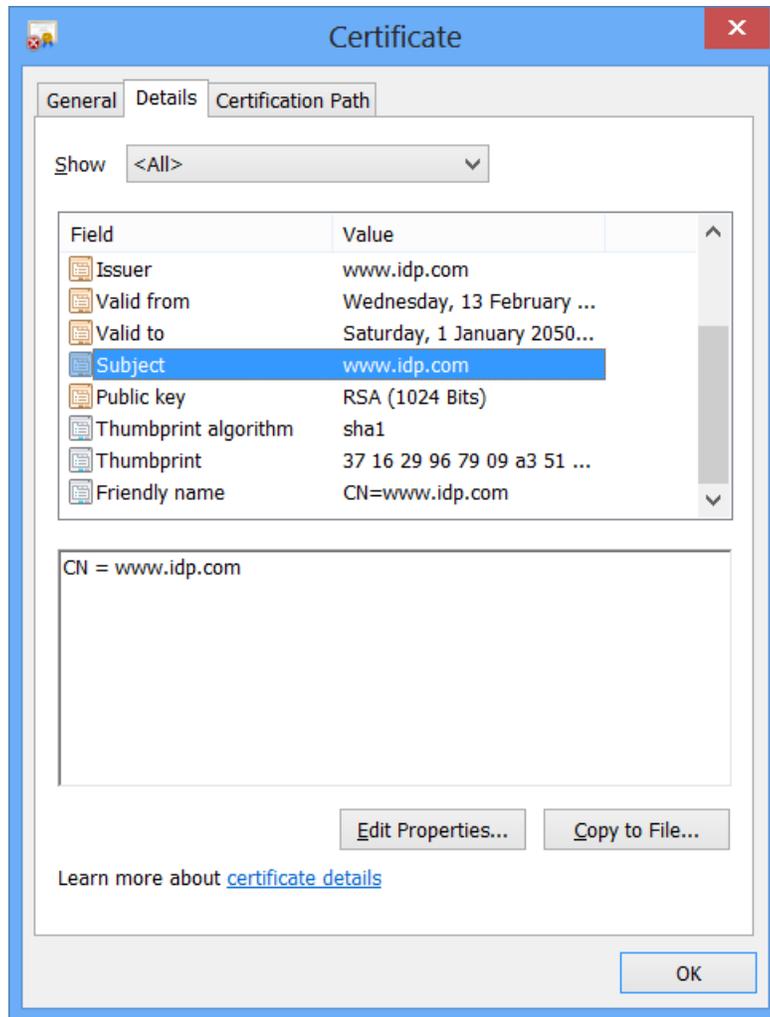
8. Note the certificate's serial number.



9. Note the certificate's thumbprint.



10. Note the certificate's subject name.

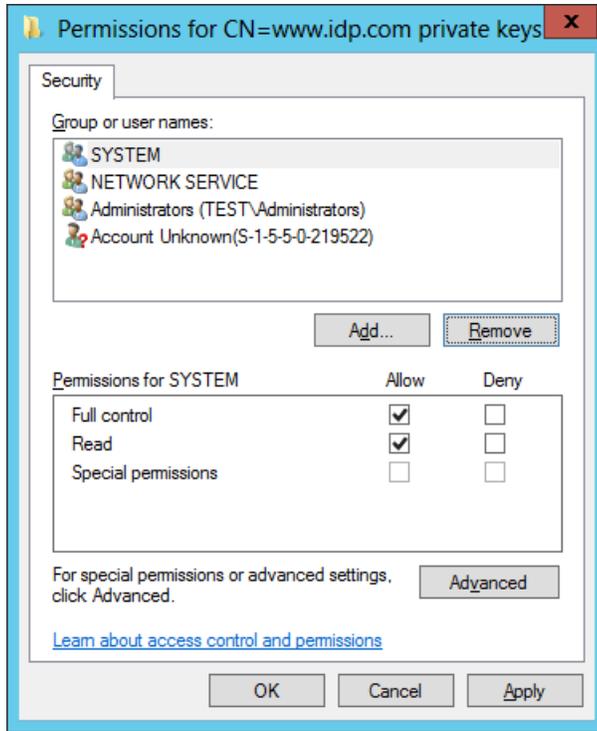


### 7.4.3 Private Key Security

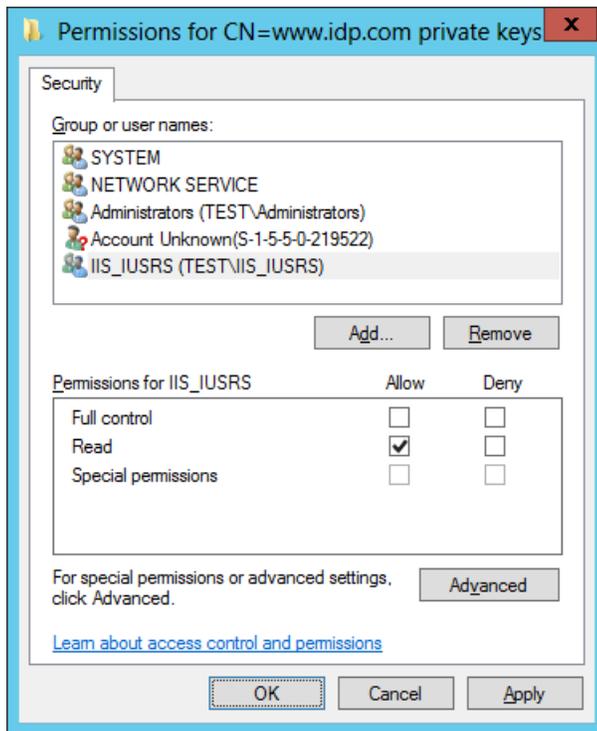
If your application will be accessing the private key (e.g. signature generation or decryption) then the account under which it runs must have read access to the private key.

To add read permission for the private key:

1. Right click on the certificate to bring up the context menu and select All Tasks > Manage Private Keys...



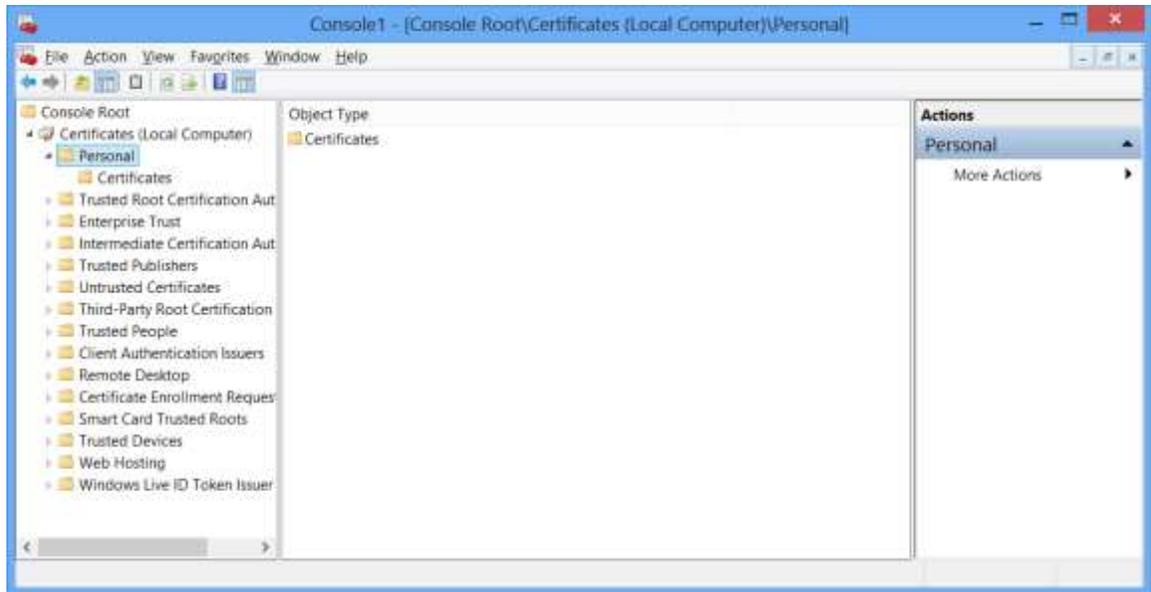
2. Add permissions for the application account. For example, give the IIS\_IUSRS group read permission. The user or group to permit is dependent on the version of IIS and its configuration.



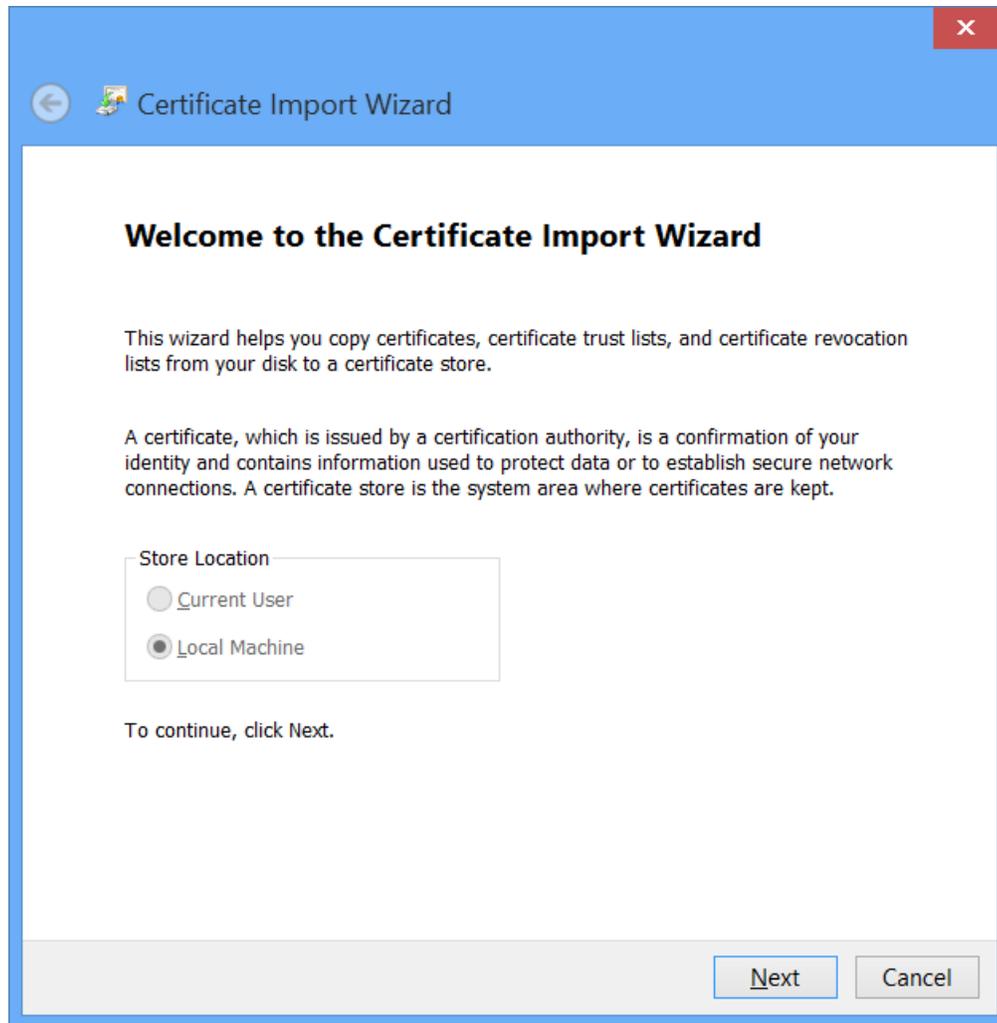
### 7.4.4 Importing a CER File

To import a certificate contained in a cer file into the Windows certificate store:

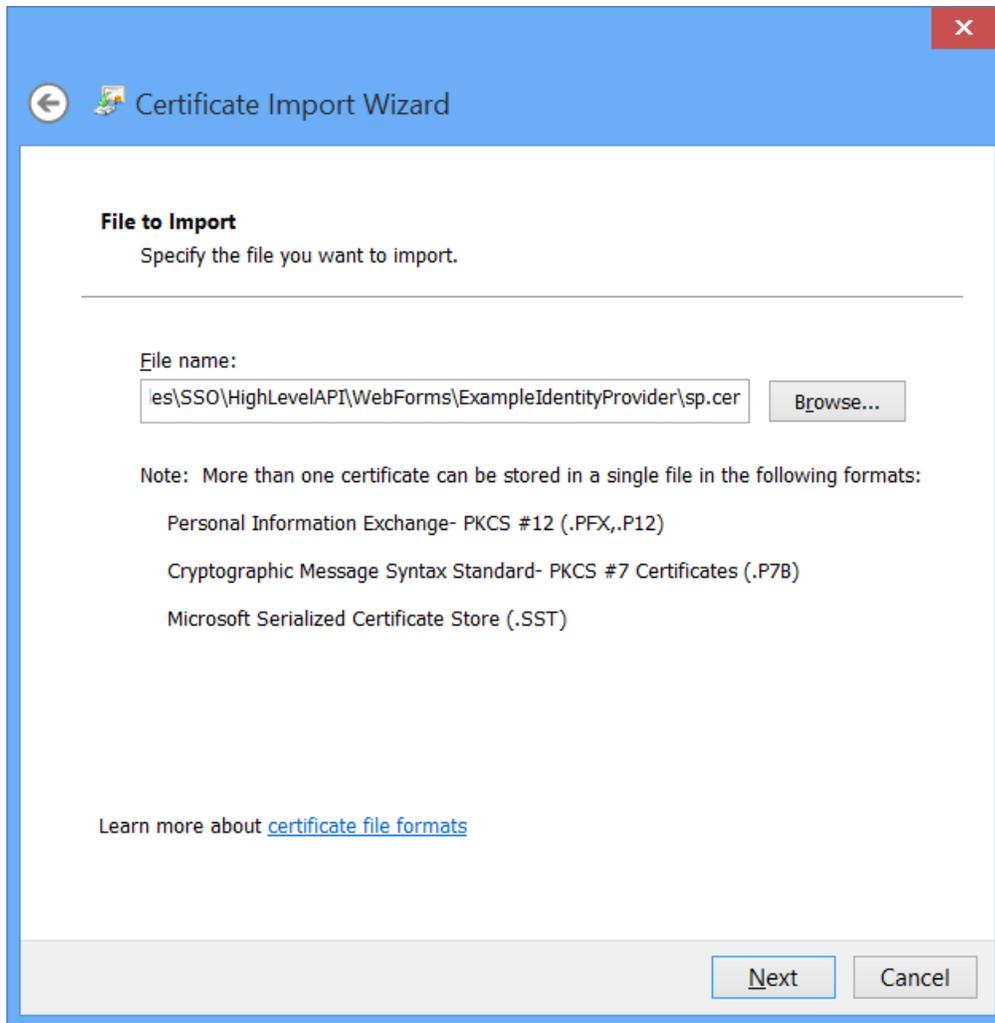
1. Select the Personal folder in the certificates tree.



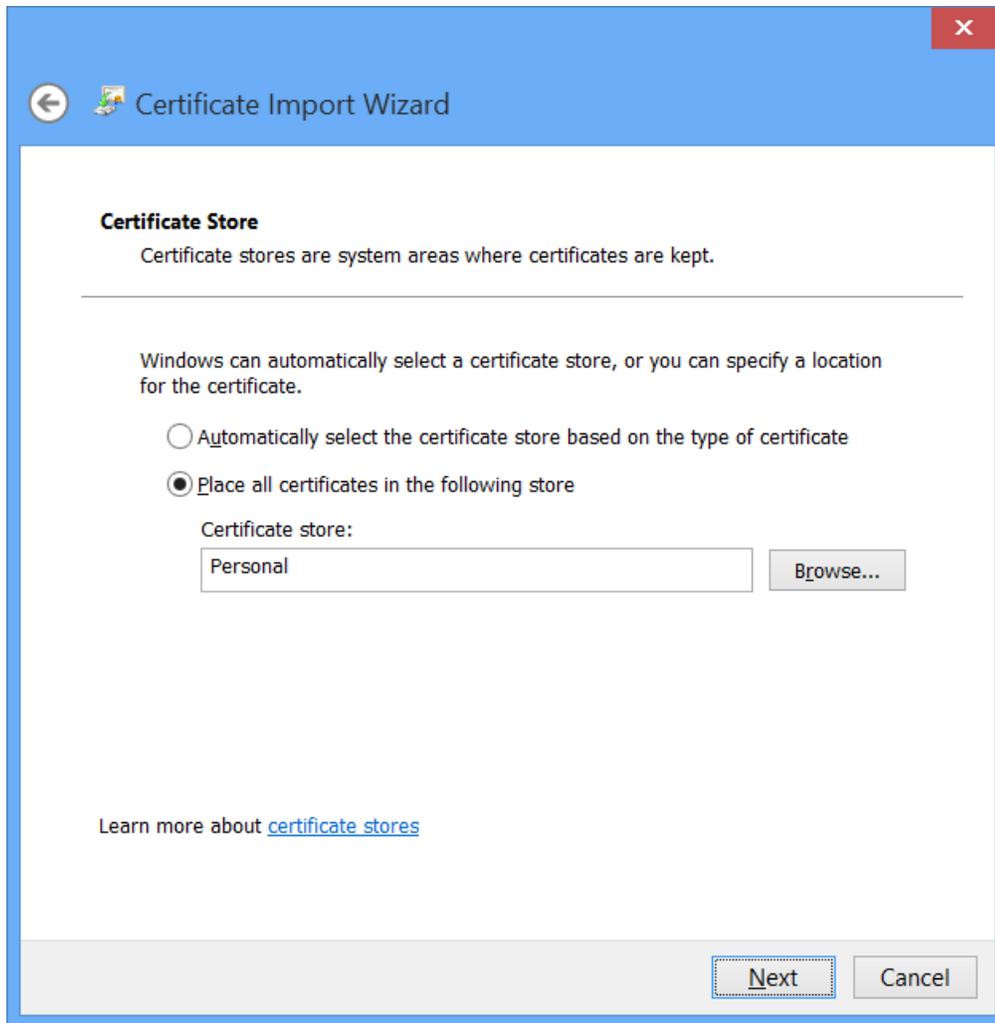
2. From the menu, click Action > All Tasks > Import...



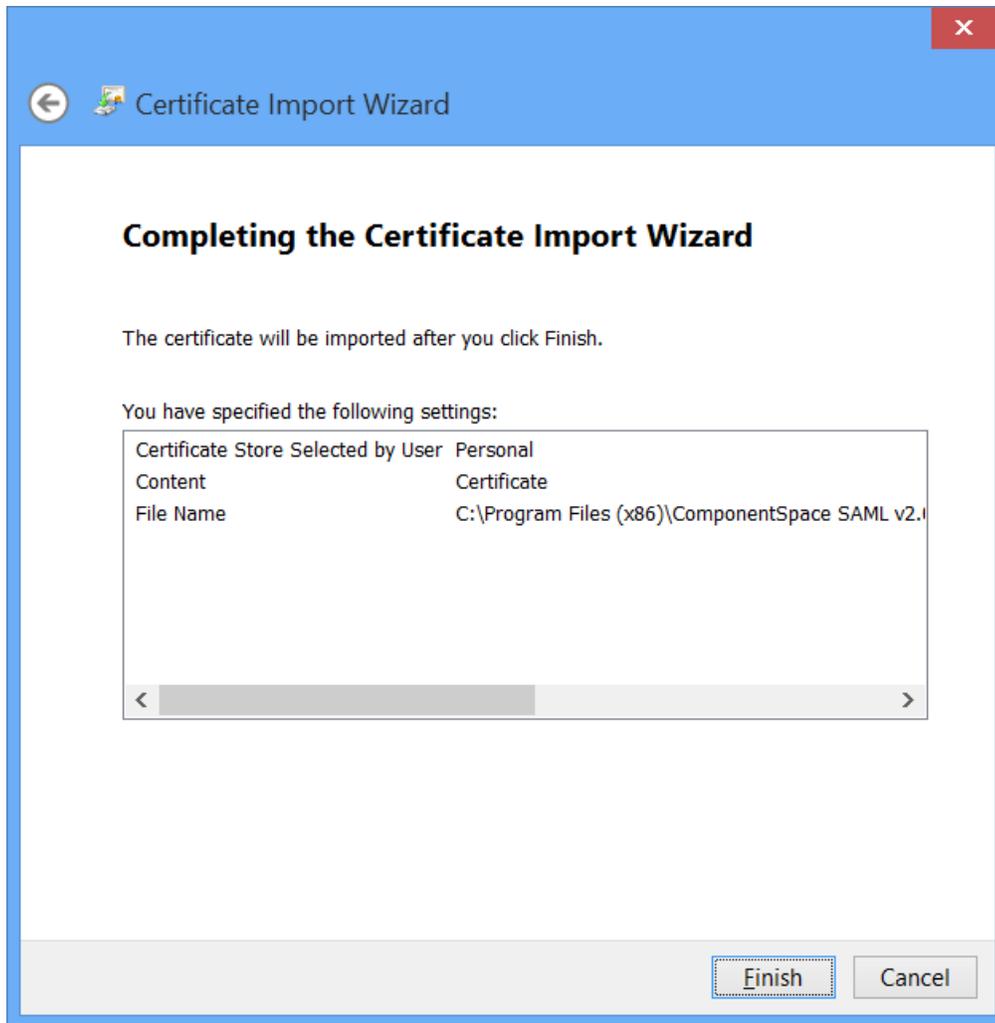
3. Browse to the cer file to import.



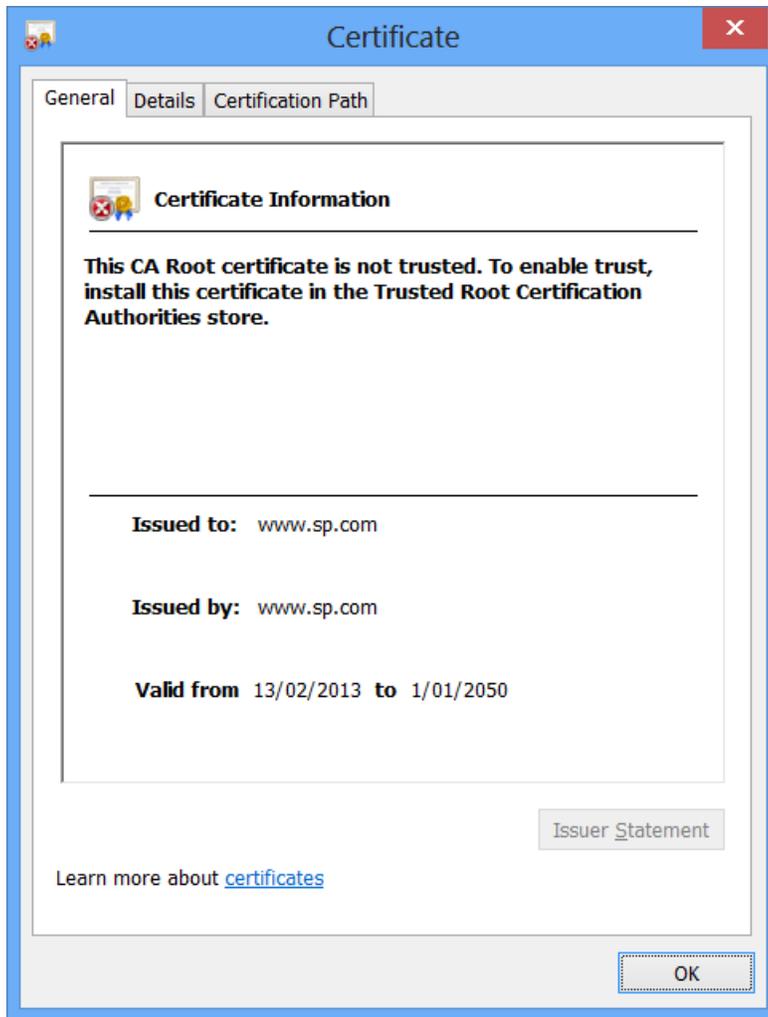
4. Place the certificate in the Personal certificate store.



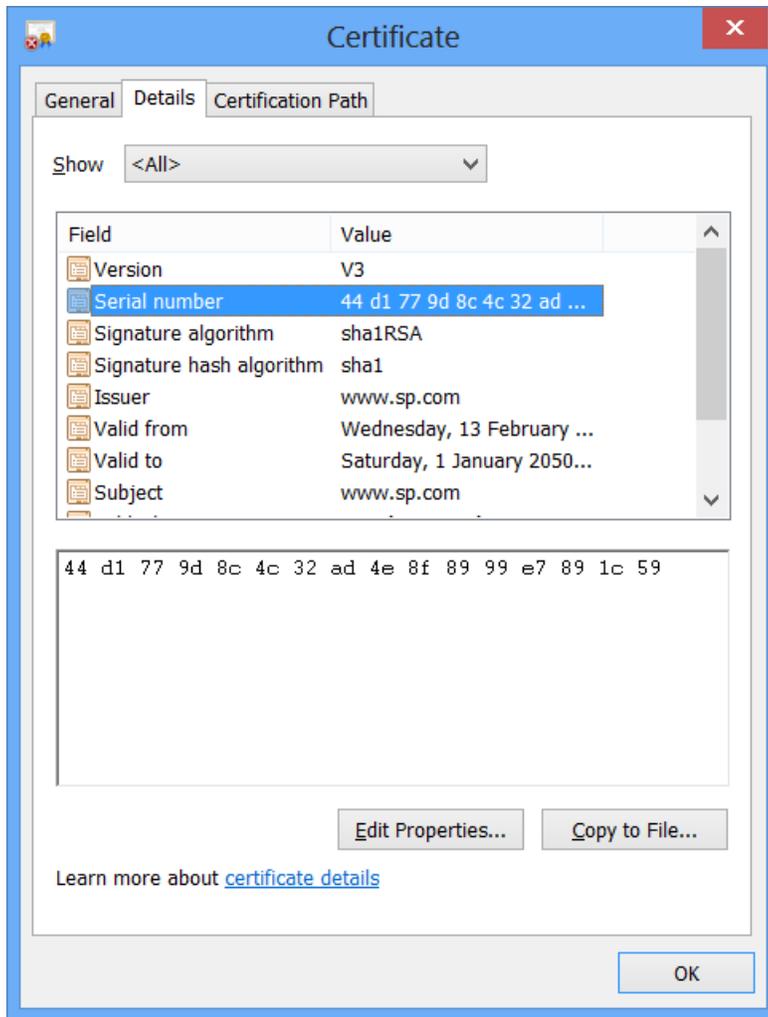
5. Click Finish to complete the import.



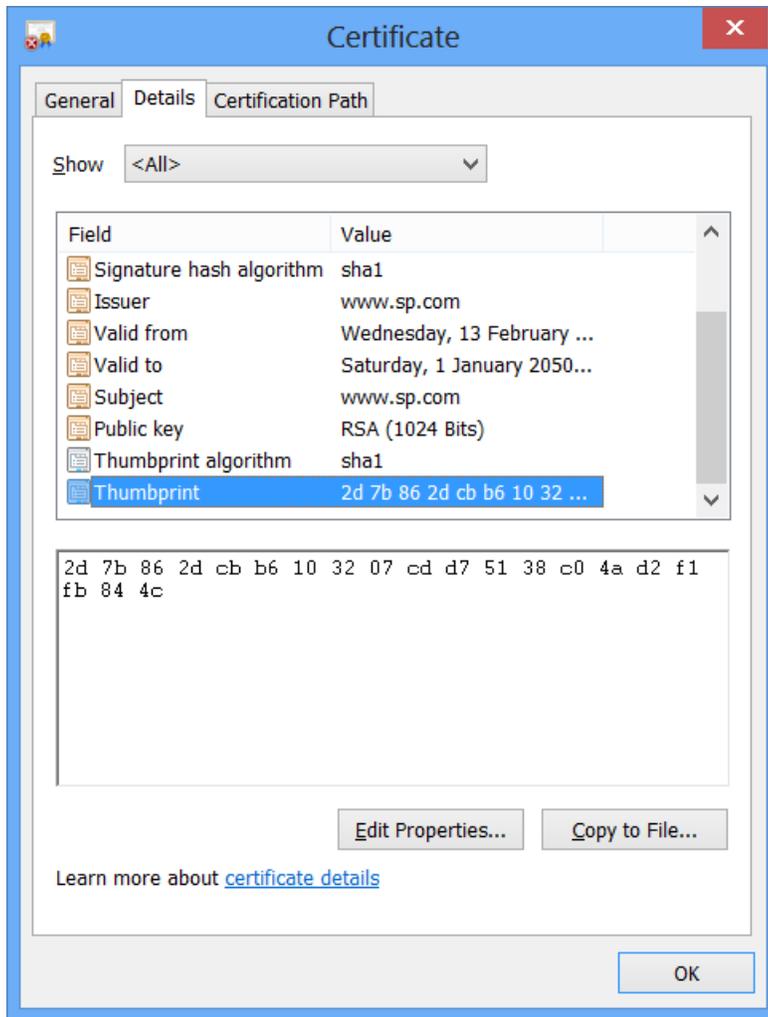
6. Confirm the certificate is listed and open it to review its details.



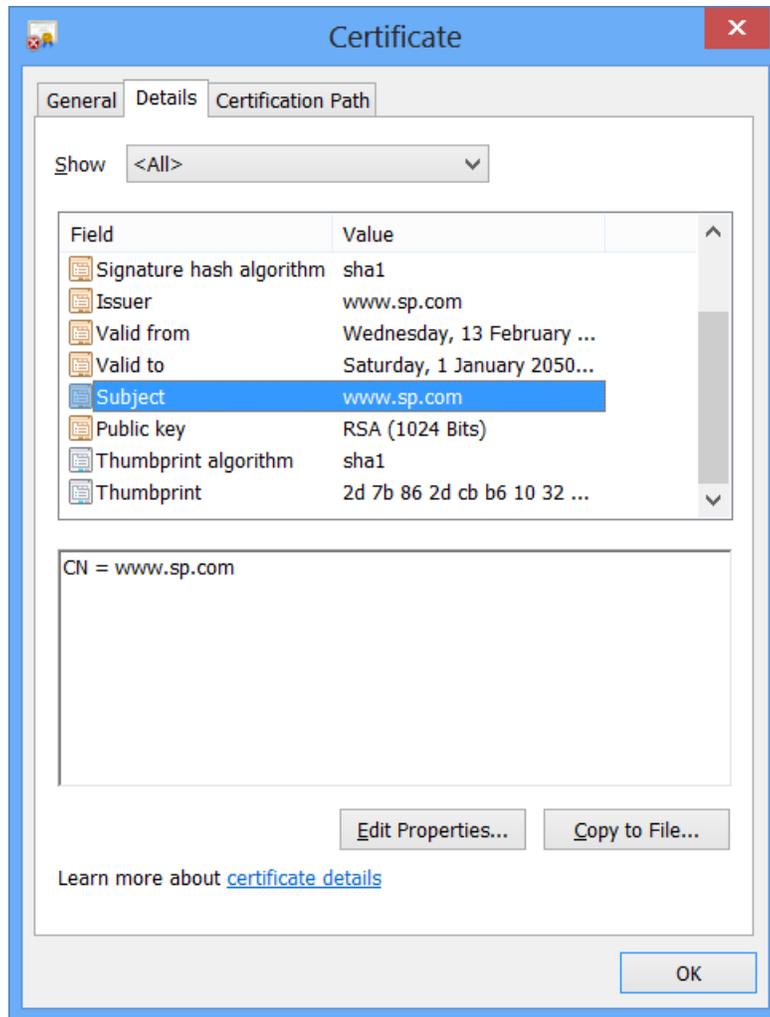
7. Note the certificate's serial number.



8. Note the certificate's thumbprint.



9. Note the certificate's subject name.



## 7.5 Local Provider Certificate Store

Refer to sections 7.4.1, 7.4.2 and 7.4.3 for instructions on importing a PFX file into the Windows certificate store.

The following configuration entry specifies the local provider's certificate is contained in the Windows certificate store and is identified by the certificate's serial number.

Serial numbers optionally may include separating space characters for readability.

If copying/pasting from the Windows certificate store, invisible Unicode characters may be included. It's best to first paste to Notepad etc. to avoid these issues.

```
<IdentityProvider Name="ExampleIdentityProvider"
  LocalCertificateSerialNumber=
    "5896b16b7fcc9c9d4ef3ca1a9b2653bb"/>
```

The following configuration entry specifies the local provider's certificate is contained in the Windows certificate store and is identified by the certificate's thumbprint.

Thumbprints optionally may include separating space characters for readability.

If copying/pasting from the Windows certificate store, invisible Unicode characters may be included. It's best to first paste to Notepad etc. to avoid these issues.

```
<IdentityProvider Name="ExampleIdentityProvider"
  LocalCertificateThumbprint=
    "371629967909a351de5017d6a3a51c6c15f3505c"/>
```

The following configuration entry specifies the local provider's certificate is contained in the Windows certificate store and is identified by the certificate subject's distinguished name.

```
<IdentityProvider Name="ExampleIdentityProvider"
  LocalCertificateSubject="CN=www.idp.com"/>
```

Alternatively, some subset of the subject's distinguished name that uniquely identifies the certificate may be used.

```
<IdentityProvider Name="ExampleIdentityProvider"
  LocalCertificateSubject="www.idp.com"/>
```

## 7.6 Partner Provider Certificate Store

Refer to sections 7.4.1 and 7.4.27.4.4 for instructions on importing a CER file into the Windows certificate store.

The following configuration entry specifies the partner provider's certificate is contained in the Windows certificate store and is identified by the certificate's serial number.

```
<PartnerServiceProvider Name="ExampleServiceProvider"
  PartnerCertificateSerialNumber=
    "44d1779d8c4c32ad4e8f8999e7891c59"/>
```

The following configuration entry specifies the partner provider's certificate is contained in the Windows certificate store and is identified by the certificate's thumbprint.

```
<PartnerServiceProvider Name="ExampleServiceProvider"
  PartnerCertificateThumbprint=
    "2d7b862dcbb6103207cdd75138c04ad2f1fb844c"/>
```

The following configuration entry specifies the partner provider's certificate is contained in the Windows certificate store and is identified by the certificate subject's distinguished name.

```
<PartnerServiceProvider Name="ExampleServiceProvider"
  PartnerCertificateSubject="CN=www.sp.com"/>
```

Alternatively, some subset of the subject's distinguished name that uniquely identifies the certificate may be used.

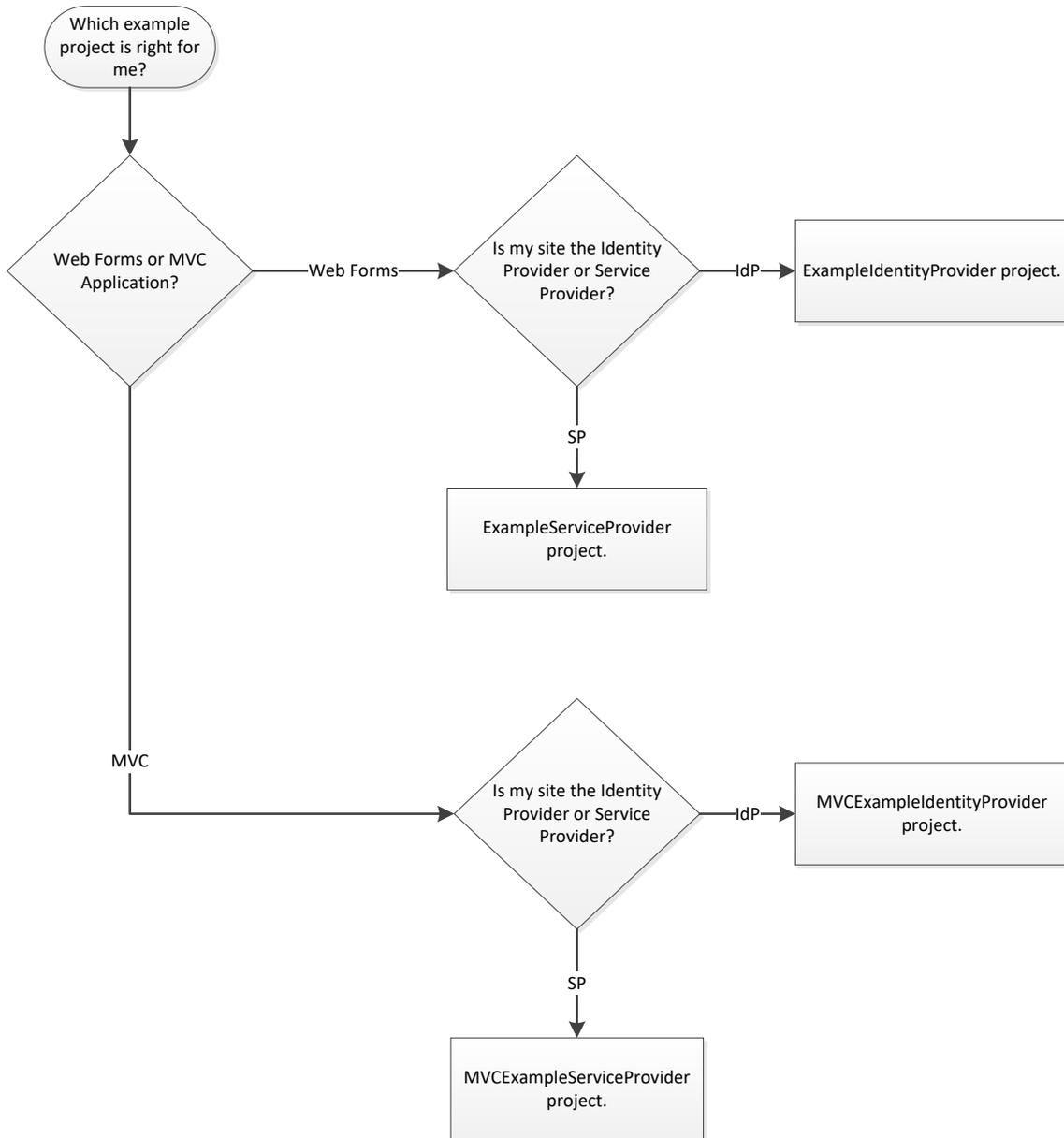
```
<PartnerServiceProvider Name="ExampleServiceProvider"
  PartnerCertificateSubject="www.sp.com "/>
```

## 8 Selecting the Most Applicable Example

For the majority of use cases, it's recommended the high level APIs are used as these provide the greatest ease of use.

## 8.1 High Level APIs

Use the following flow chart to determine the most suitable high level API example project to review.



**Figure 8 High Level API Project Selection Flow Chart**

If your application is an ASP.NET web forms application then refer to the ExampleIdentityProvider or ExampleServiceProvider project depending on whether your site is the identity provider or service provider. See section 10.1 for more information.

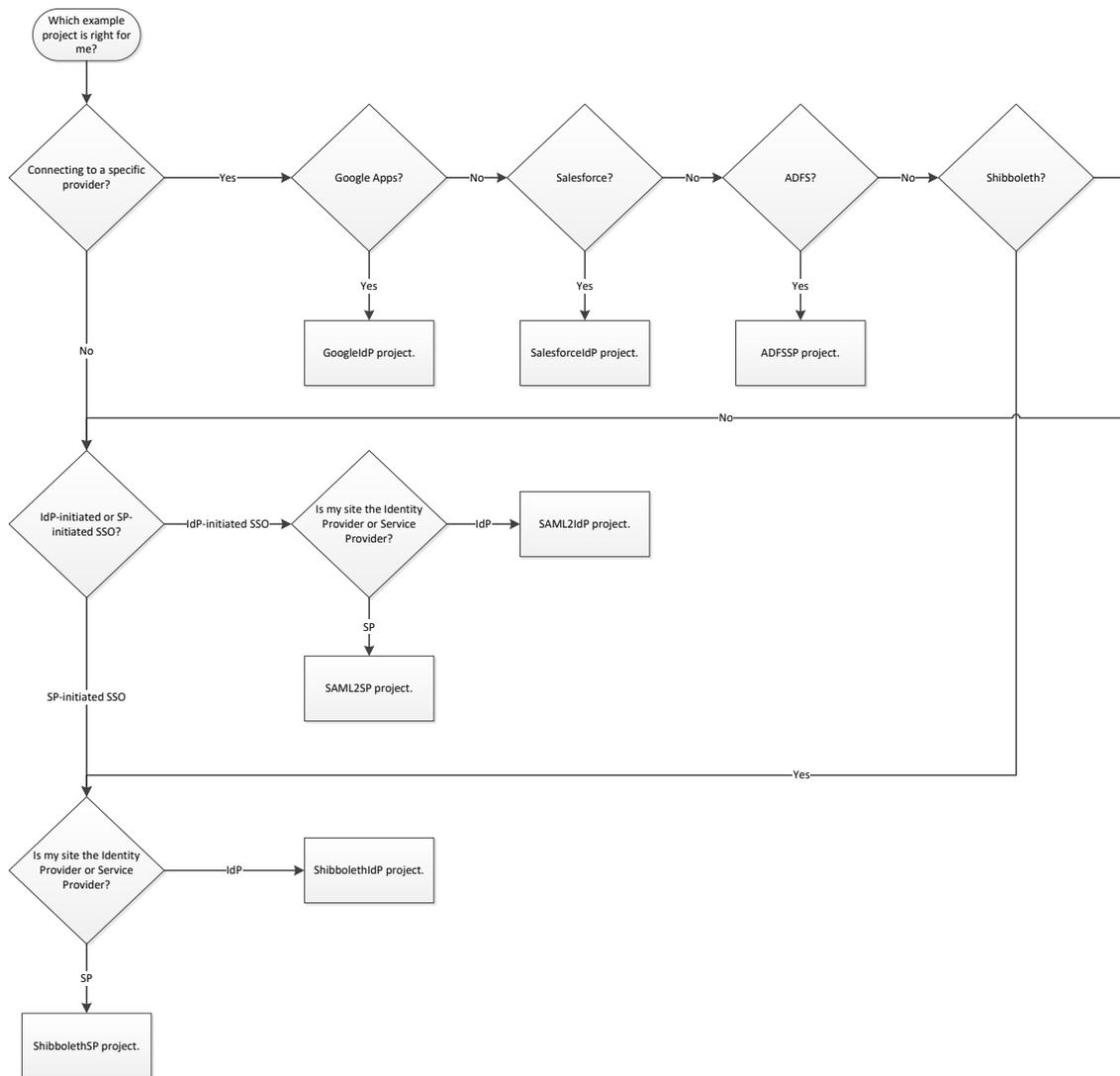
If your application is an ASP.NET MVC application then refer to the MvcExampleIdentityProvider or MvcExampleServiceProvider project depending on

whether your site is the identity provider or service provider. See section 10.1.7 for more information.

## 8.2 Low Level APIs

Use the following flow chart to determine the most suitable low level API example project to review.

Note that the high level API examples include support for Google Apps, Salesforce, ADFS and Shibboleth, as well as IdP-initiated and SP-initiated SSO.



**Figure 9 Low Level API Project Selection Flow Chart**

If you're connecting to a specific provider such as Google Apps etc then refer to the corresponding example project. If the provider is not listed then use one of the generic projects or contact us for assistance.

For IdP-initiated SSO, refer to the SAML2IdP or SAML2SP project depending on whether your site is the identity provider or service provider.  
For SP-initiated SSO, refer to the ShibbolethIdP or ShibbolethSP project depending on whether your site is the identity provider or service provider.

The ShibbolethIdP and ShibbolethSP projects have been tested against the Shibboleth test servers at <https://www.testshib.org/testshib-two/>.

## 9 Building the Example Applications

Solution files for Visual Studio 2005, 2008, 2010, 2012 and 2013 may be found in the root folder (e.g. C:\Program Files (x86)\ComponentSpace SAML v2.0 for .NET).

- SAMLExamplesVS2013.sln – Visual Studio 2013
- SAMLExamplesVS2012.sln – Visual Studio 2012
- SAMLExamplesVS2010.sln – Visual Studio 2010
- SAMLExamplesVS2008.sln – Visual Studio 2008
- SAMLExamplesVS2005.sln – Visual Studio 2005

Select the appropriate solution file to build the example projects.

The Visual Studio 2013 solution includes publish definitions for publishing to the default web site on the local host.

The projects should build and run without error.

## 10 Example Applications – High Level APIs

The class library ships with a number of example applications. They are a good way to become familiar with the SAML v2.0 web browser SSO profile and using the class library. You may use the examples solution to build these projects.

The example applications must be built and published prior to their use.

The following sections describe the installation and execution of these example applications.

The example web forms applications described in section 10.1 demonstrate IdP-initiated and SP-initiated SSO. These applications are written in C#.

The example MVC applications described in sections 10.1.7 demonstrate IdP-initiated and SP-initiated SSO. These applications are written in C#.

### 10.1 Web Forms Identity Provider and Service Provider

The ExampleIdentityProvider and ExampleServiceProvider web applications demonstrate IdP-initiated and SP-initiated single sign-on.

#### 10.1.1 Installing the Web Forms Identity Provider

1. Using Visual Studio, build and publish the web application.

2. Open the Internet Information Services management console.
3. Under the default web site for the local computer, create an application with an alias of ExampleIdentityProvider.
4. For the physical path, browse to the directory where ExampleIdentityProvider was built and published.
5. Ensure the web application has been successfully installed by browsing to <http://localhost/ExampleIdentityProvider>.

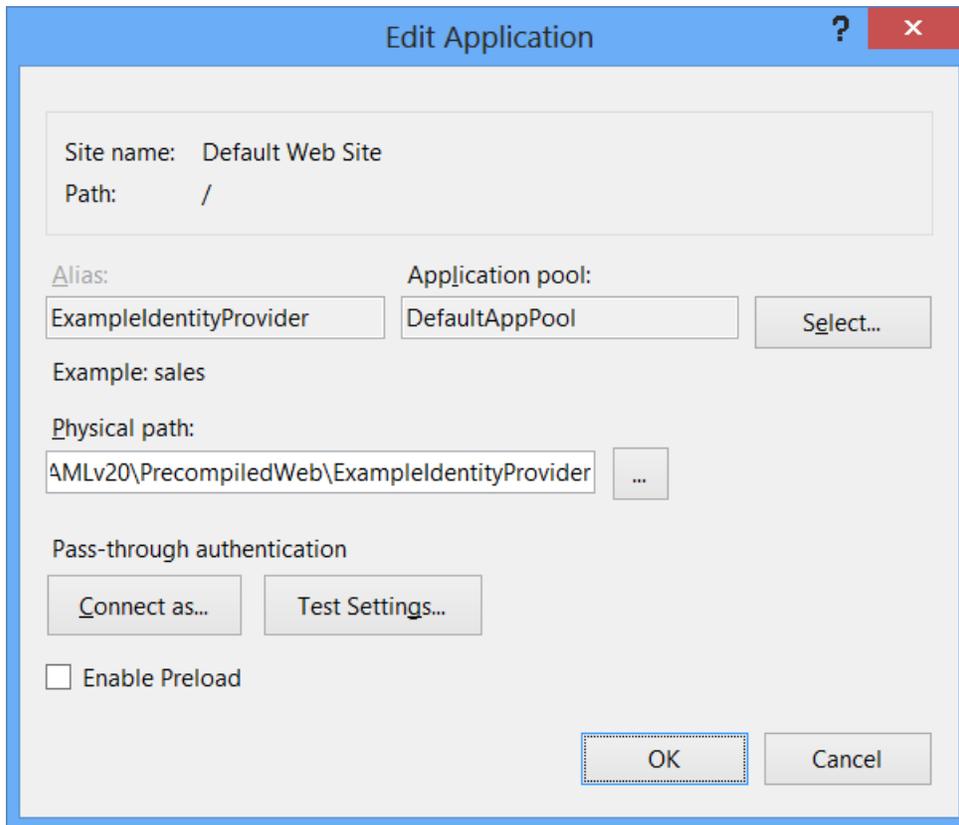


Figure 10 ExampleIdentityProvider Installation

### 10.1.2 Installing the Web Forms Service Provider

1. Using Visual Studio, build and publish the web application.
2. Open the Internet Information Services management console.
3. Under the default web site for the local computer, create an application with an alias of ExampleServiceProvider.
4. For the physical path, browse to the directory where ExampleServiceProvider was built and published.
5. Ensure the web application has been successfully installed by browsing to <http://localhost/ExampleServiceProvider>.

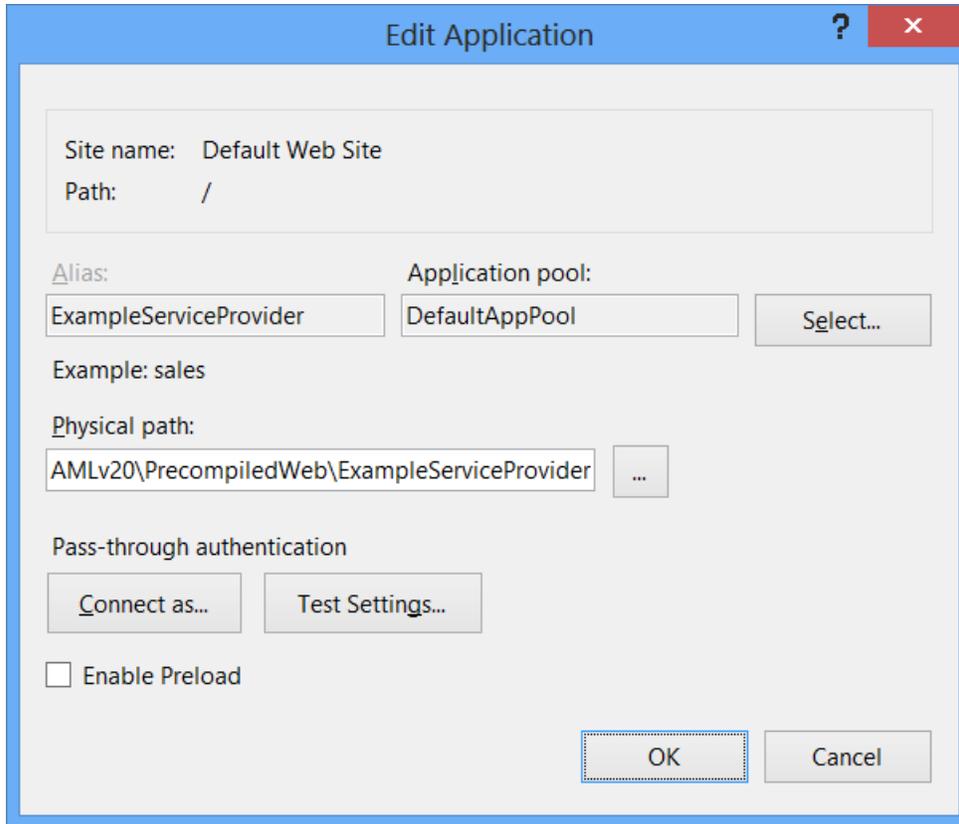


Figure 11 ExampleServiceProvider Installation

### 10.1.3 Configuring the Web Forms Identity Provider

The identity provider configuration is contained within its web.config file's <appSettings> section and the saml.config file.

The saml.config includes the local identity provider configuration as well as partner service provider configuration.

The web.config's *PartnerSP* setting specifies the partner service provider for IdP-initiated SSO.

The default configuration supports single sign-on with the ExampleServiceProvider.

### 10.1.4 Configuring the Web Forms Service Provider

The service provider configuration is contained within its web.config file's <appSettings> section and the saml.config file.

The saml.config includes the local service provider configuration as well as partner identity provider configuration.

The web.config's *PartnerIdP* setting specifies the partner identity provider for SP-initiated SSO.

The default configuration supports single sign-on with the ExampleIdentityProvider.

### 10.1.5 IdP-Initiated SSO from the Web Forms Identity Provider

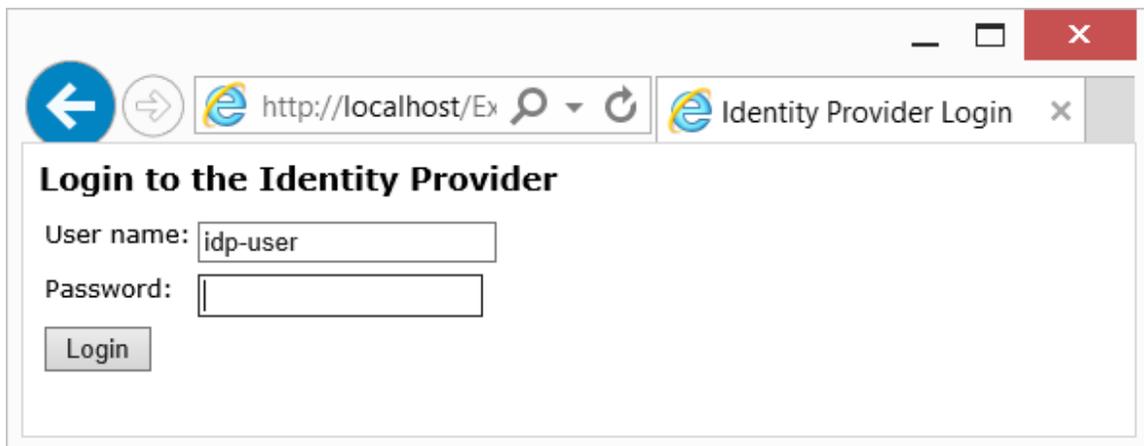
In this example, the user starts at the identity provider site and is attempting to access a protected resource on the service provider. Rather than performing a local login at the service provider, SSO is initiated with a local login occurring at the identity provider and the asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the service provider.

1. Browse to `http://localhost/ExampleIdentityProvider`.
2. You should be presented with the form shown in Figure 12.

If you are not then you must already have logged in at the identity provider. To force a login, close the browser and start again.

3. Login using the user name *idp-user* and a password of *password*.
4. You should then be presented with the identity provider's default page (see Figure 13).
5. Click the link to single sign-on to the service provider.
6. You should then be presented with the service provider's default page (see Figure 14).

This means you've successfully completed a SAML v2.0 SSO and are logged in at the service provider with your identity provider user name.



The screenshot shows a web browser window with the following details:

- Address bar: `http://localhost/ExampleIdentityProvider`
- Page title: Identity Provider Login
- Form title: Login to the Identity Provider
- User name field: `idp-user`
- Password field: (empty)
- Login button: Login

Figure 12 Web Forms Example Identity Provider Login Page

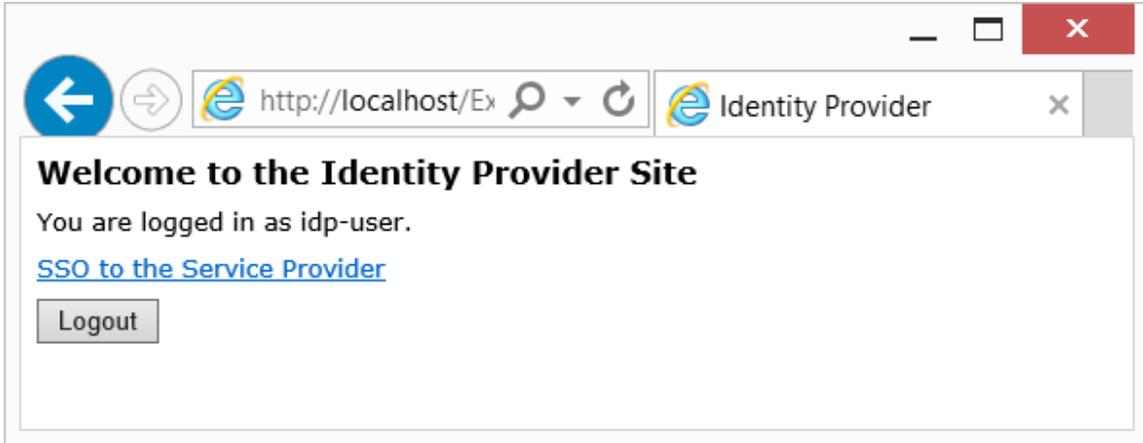


Figure 13 Web Forms Example Identity Provider Default Page

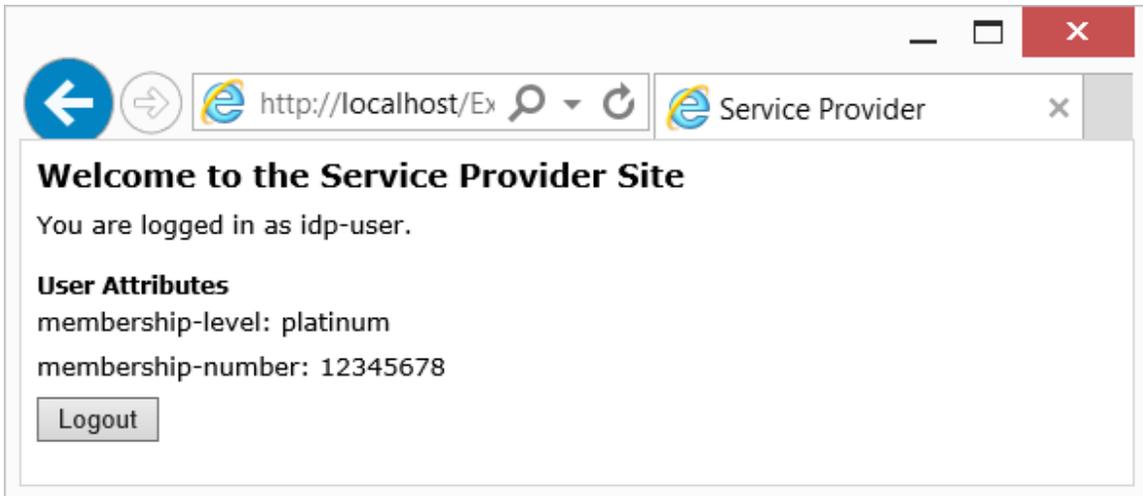


Figure 14 Web Forms Example Service Provider Home Page

### 10.1.6 SP-Initiated SSO from the Web Forms Service Provider

In this example, the user starts at the service provider site and is attempting to access a protected resource on the service provider. Rather than performing a local login at the service provider, SSO is initiated with a local login occurring at the identity provider and the asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the service provider.

1. Browse to <http://localhost/ExampleServiceProvider>.
2. You should then be presented with the service provider's login page.
3. Click the link to single sign-on to the identity provider.
4. You should be presented with the form shown in Figure 16.

If you are not then you must already have logged in at the identity provider. To force a login, close the browser and start again.

5. Login using the user name *idp-user* and a password of *password*.
6. You should then be presented with the service provider's default page (see Figure 17).

This means you've successfully completed a SAML v2.0 SSO and are logged in at the service provider with your identity provider user name.

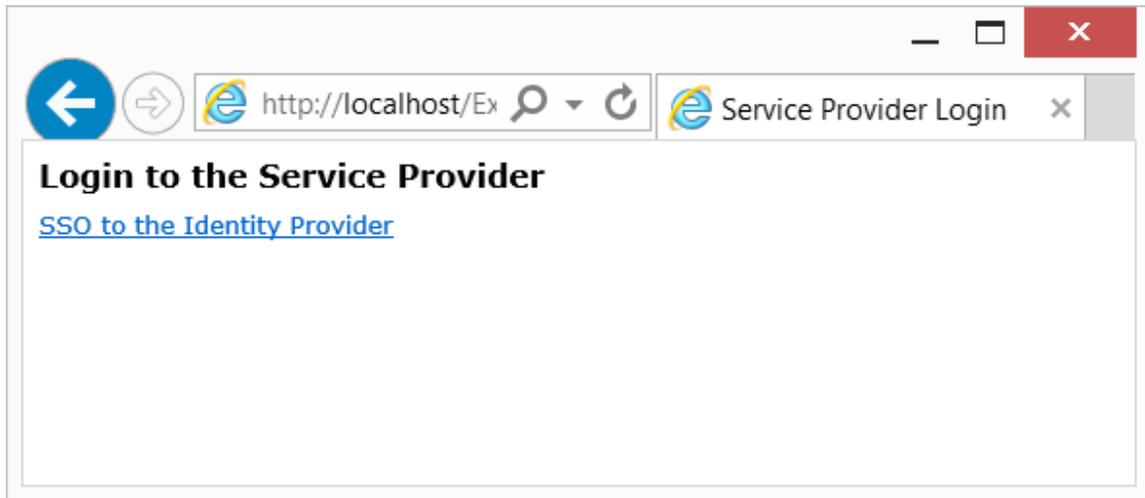


Figure 15 Web Forms Example Service Provider Login Page

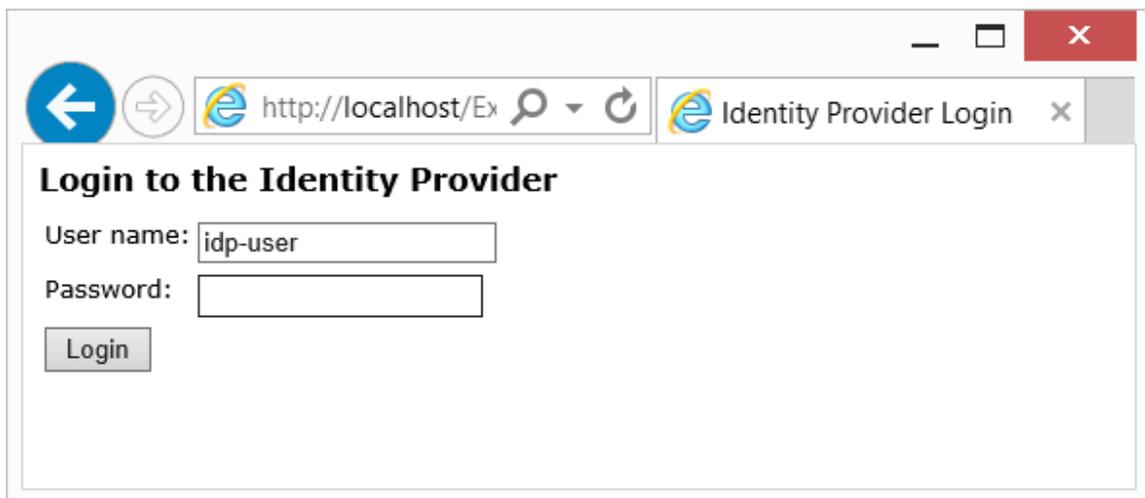


Figure 16 Web Forms Example Identity Provider Login Page

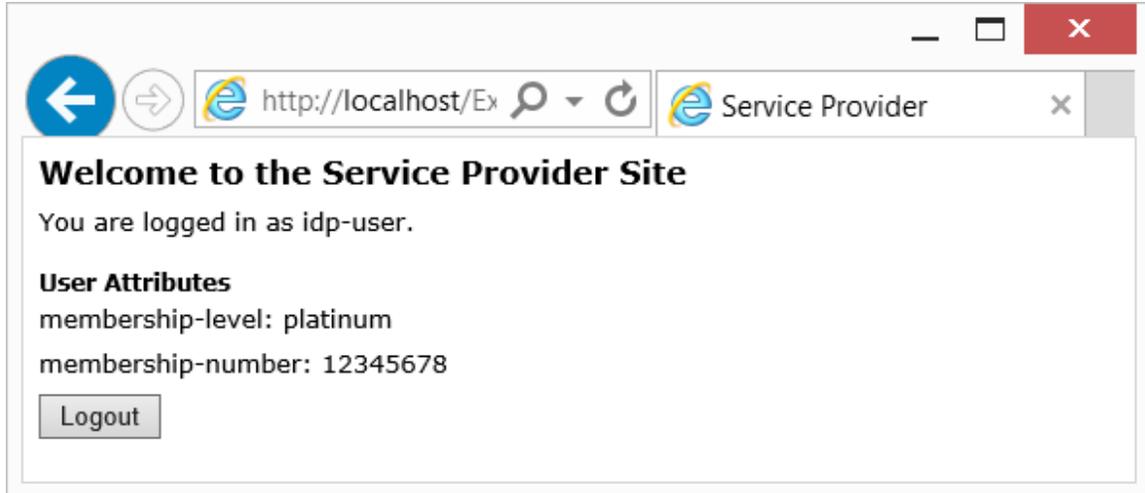


Figure 17 Web Forms Example Service Provider Home Page

### 10.1.7 Code Walkthrough - IdP-Initiated SSO

The following section follows the flow of IdP-initiated SSO between the ExampleIdentityProvider and ExampleServiceProvider.

1. The user clicks the link on the ExampleIdentityProvider's Default.aspx page and the SAMLIdentityProvider.InitiateSSO method is called to initiate SSO.
2. A SAML response containing a SAML assertion is constructed and sent to the service provider's assertion consumer service URL.
3. The ExampleServiceProvider's SAML/AssertionConsumerService.aspx page calls the SAMLServiceProvider.ReceiveSSO method to receive and process the SAML response.
4. The user is logged in automatically at the ExampleServiceProvider.

### 10.1.8 Code Walkthrough - SP-Initiated SSO

The following section follows the flow of SP-initiated SSO between the ExampleIdentityProvider and ExampleServiceProvider.

1. The user browses to the ExampleServiceProvider's Default.aspx page and the SAMLServiceProvider.InitiateSSO method is called to initiate SSO.
2. An authentication request is constructed and sent to the identity provider's SSO service URL.
3. The ExampleIdentityProvider's SAML/SSOService.aspx page calls the SAMLIdentityProvider.ReceiveSSO method to receive and process the authentication request.
4. The user is prompted to login at the ExampleIdentityProvider if not already logged in.

5. The ExampleIdentityProvider's SAML/SSOService.aspx page calls the SAMLIdentityProvider.SendSSO method.
6. A SAML response containing a SAML assertion is constructed and sent to the service provider's assertion consumer service URL.
7. The ExampleServiceProvider's SAML/AssertionConsumerService.aspx page calls the SAMLServiceProvider.ReceiveSSO method to receive and process the SAML response.
8. The user is logged in automatically at the ExampleServiceProvider.

## **10.2 MVC Identity Provider and Service Provider**

The MvcExampleIdentityProvider and MvcExampleServiceProvider web applications demonstrate IdP-initiated and SP-initiated single sign-on.

The applications use MVC4 but the SAML v2.0 component may be used with earlier versions of MVC.

The MVC examples require Visual Studio 2012 or later, or Visual Studio 2010 with the MVC4 upgrade.

### **10.2.1 Installing the MVC Identity Provider**

1. Using Visual Studio, build and publish the web application.
2. Open the Internet Information Services management console.
3. Under the default web site for the local computer, create an application with an alias of MvcExampleIdentityProvider.
4. For the physical path, browse to the directory where MvcExampleIdentityProvider was built and published.
5. Ensure the web application has been successfully installed by browsing to <http://localhost/MvcExampleIdentityProvider>.

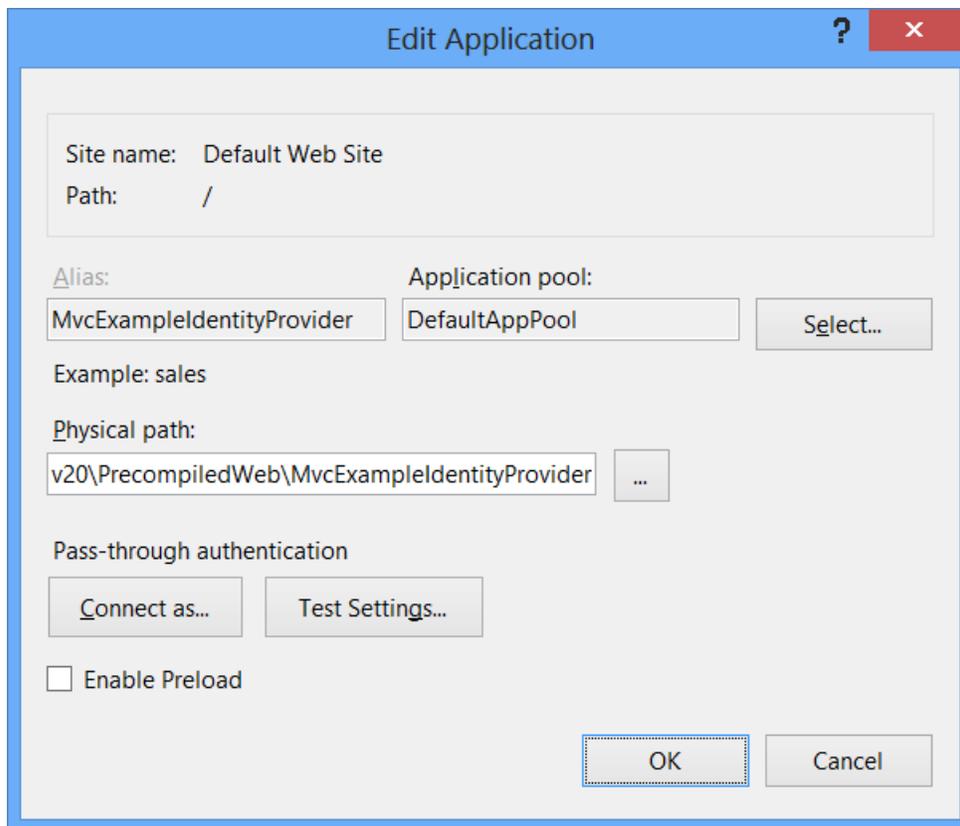


Figure 18 MvcExampleIdentityProvider Installation

### 10.2.2 Installing the MVC Service Provider

1. Using Visual Studio, build and publish the web application.
2. Open the Internet Information Services management console.
3. Under the default web site for the local computer, create an application with an alias of MvcExampleServiceProvider.
4. For the physical path, browse to the directory where MvcExampleServiceProvider was built and published.
5. Ensure the web application has been successfully installed by browsing to <http://localhost/MvcExampleServiceProvider>.

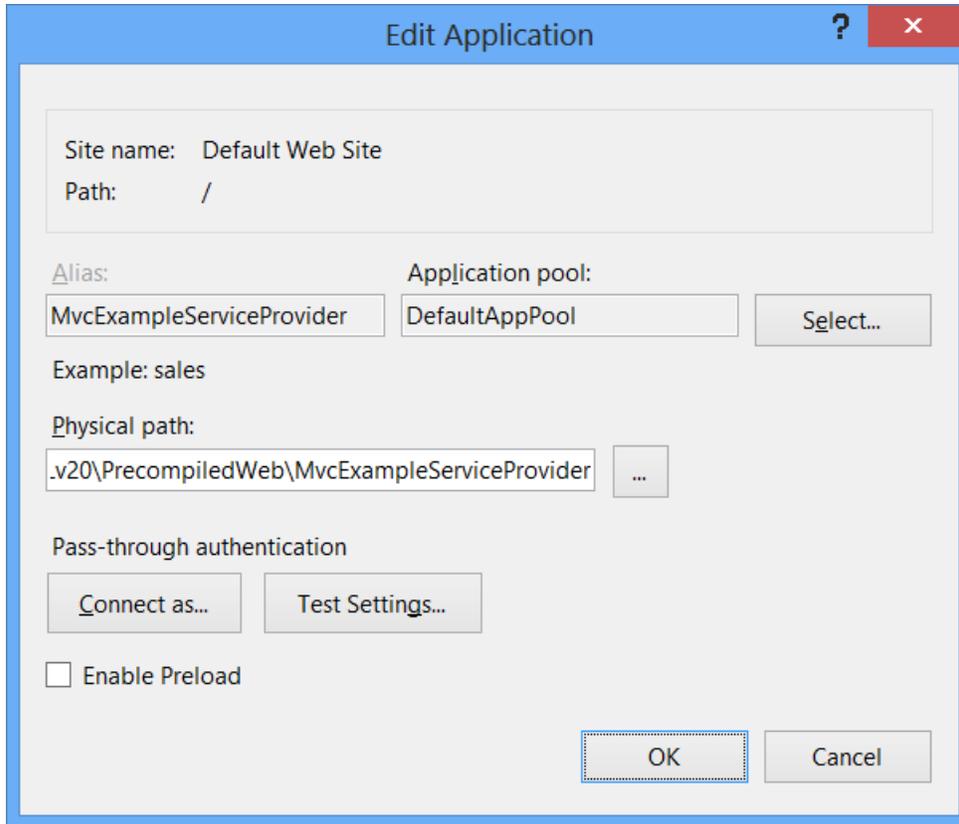


Figure 19 MvcExampleServiceProvider Installation

### 10.2.3 Configuring the MVC Identity Provider

The identity provider configuration is contained within its web.config file's <appSettings> section and the saml.config file.

The saml.config includes the local identity provider configuration as well as partner service provider configuration.

The web.config's *PartnerSP* setting specifies the partner service provider for IdP-initiated SSO.

The default configuration supports single sign-on with the MvcExampleServiceProvider.

### 10.2.4 Configuring the MVC Service Provider

The service provider configuration is contained within its web.config file's <appSettings> section and the saml.config file.

The saml.config includes the local service provider configuration as well as partner identity provider configuration.

The web.config's *PartnerIdP* setting specifies the partner identity provider for SP-initiated SSO.

The default configuration supports single sign-on with the MvcExampleIdentityProvider.

## 10.2.5 IdP-Initiated SSO from the MVC Identity Provider

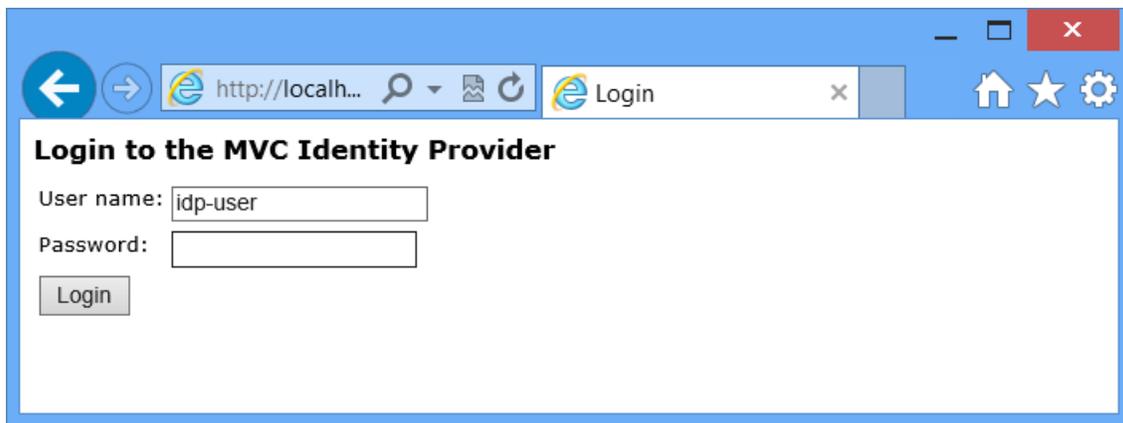
In this example, the user starts at the identity provider site and is attempting to access a protected resource on the service provider. Rather than performing a local login at the service provider, SSO is initiated with a local login occurring at the identity provider and the asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the service provider.

1. Browse to `http://localhost/MvcExampleIdentityProvider`.
2. You should be presented with the form shown in Figure 20.

If you are not then you must already have logged in at the identity provider. To force a login, close the browser and start again.

3. Login using the user name *idp-user* and a password of *password*.
4. You should then be presented with the identity provider's default page (see Figure 21).
5. Click the link to single sign-on to the service provider.
6. You should then be presented with the service provider's default page (see Figure 22).

This means you've successfully completed a SAML v2.0 SSO and are logged in at the service provider with your identity provider user name.



The screenshot shows a web browser window with the address bar containing `http://localhost/...` and a tab titled "Login". The main content area displays the heading "Login to the MVC Identity Provider" and a login form. The form includes a "User name:" field with the text "idp-user" entered, a "Password:" field which is currently empty, and a "Login" button positioned below the password field.

Figure 20 MVC Example Identity Provider Login Page

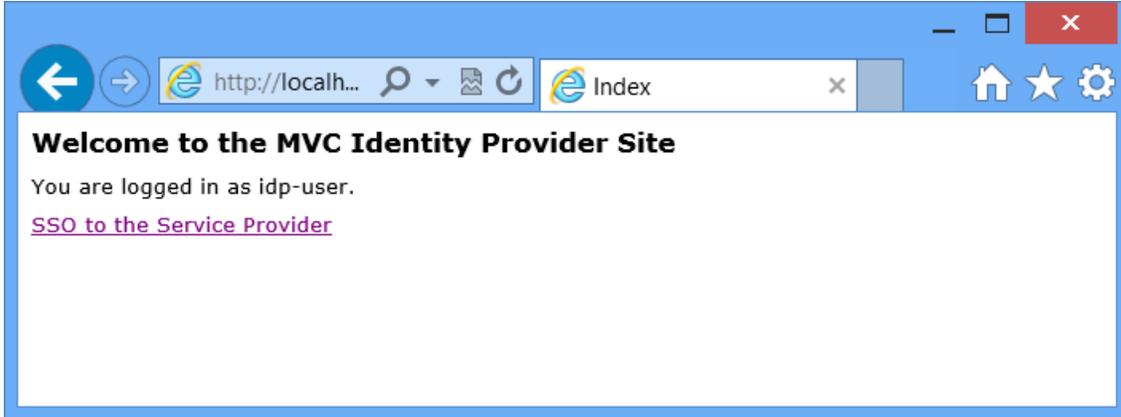


Figure 21 MVC Example Identity Provider Default Page

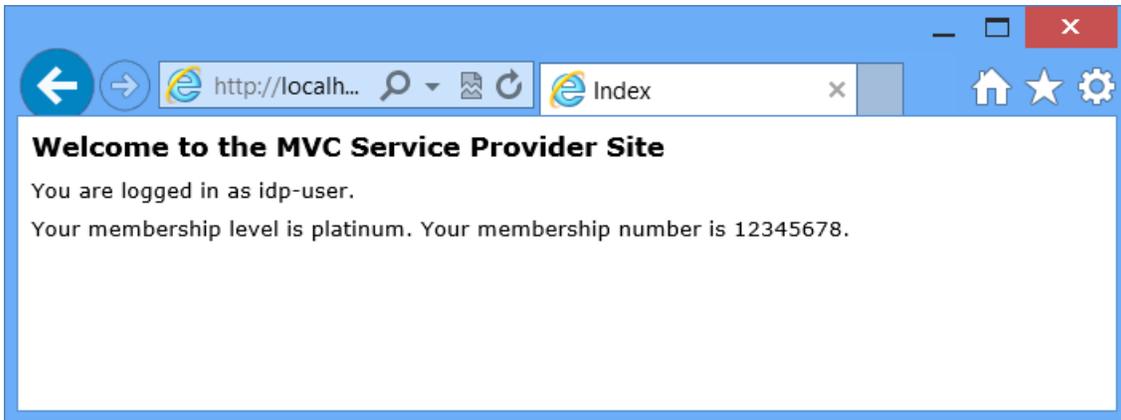


Figure 22 MVC Example Service Provider Home Page

### 10.2.6 SP-Initiated SSO from the MVC Service Provider

In this example, the user starts at the service provider site and is attempting to access a protected resource on the service provider. Rather than performing a local login at the service provider, SSO is initiated with a local login occurring at the identity provider and the asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the service provider.

1. Browse to <http://localhost/MvcExampleServiceProvider>.
2. You should then be presented with the service provider's login.
3. Click the link to single sign-on to the identity provider.
4. You should be presented with the form shown in Figure 23.

If you are not then you must already have logged in at the identity provider. To force a login, close the browser and start again.

5. Login using the user name *idp-user* and a password of *password*.

6. You should then be presented with the service provider's default page (see Figure 24).

This means you've successfully completed a SAML v2.0 SSO and are logged in at the service provider with your identity provider user name.

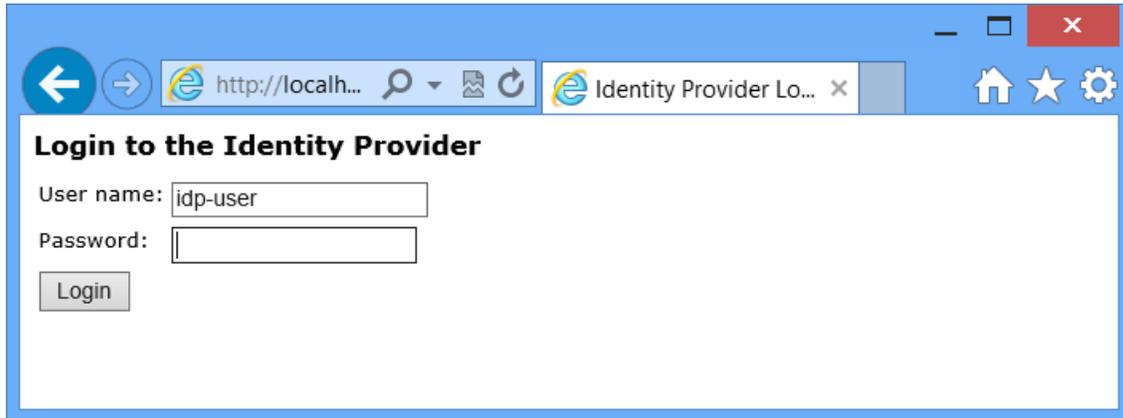


Figure 23 MVC Example Identity Provider Login Page

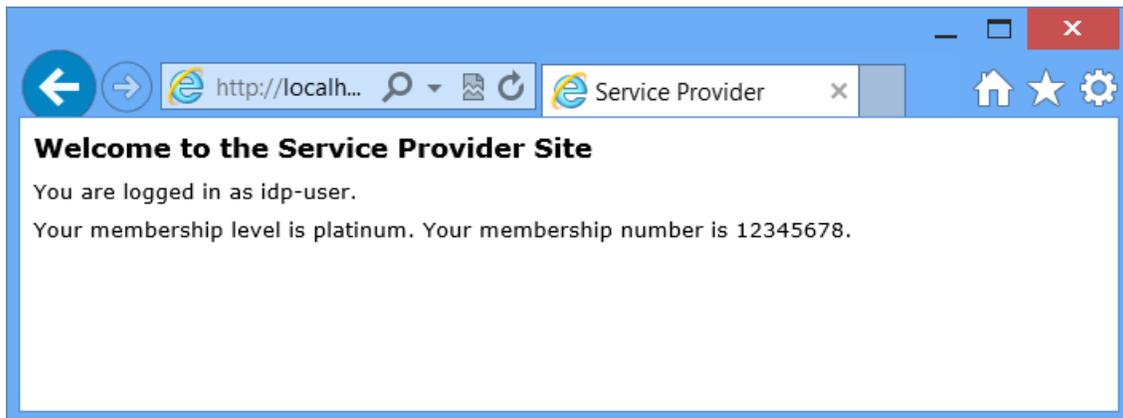


Figure 24 MVC Example Service Provider Home Page

### 10.2.7 Code Walkthrough - IdP-Initiated SSO

The following section follows the flow of IdP-initiated SSO between the `MvcExampleIdentityProvider` and `MvcExampleServiceProvider`.

1. The user clicks the link on the `MvcExampleIdentityProvider`'s home page and the `HomeController` calls the `SAMLIdentityProvider.InitiateSSO` method to initiate SSO.
2. A SAML response containing a SAML assertion is constructed and sent to the service provider's assertion consumer service URL.

3. The `MvcExampleServiceProvider`'s `SAMLController` calls the `SAMLServiceProvider.ReceiveSSO` method to receive and process the SAML response.
4. The user is logged in automatically at the `MvcExampleServiceProvider`.

### **10.2.8 Code Walkthrough - SP-Initiated SSO**

The following section follows the flow of SP-initiated SSO between the `MvcExampleIdentityProvider` and `MvcExampleServiceProvider`.

1. The user browses to the `MvcExampleServiceProvider`'s home page and the `AccountController` calls the `SAMLServiceProvider.InitiateSSO` method to initiate SSO.
2. An authentication request is constructed and sent to the identity provider's SSO service URL.
3. The `MvcExampleIdentityProvider`'s `SAMLController` calls the `SAMLIdentityProvider.ReceiveSSO` method to receive and process the authentication request.
4. The user is prompted to login at the `MvcExampleIdentityProvider` if not already logged in.
5. The `MvcExampleIdentityProvider`'s `SAMLController` calls the `SAMLIdentityProvider.SendSSO` method.
6. A SAML response containing a SAML assertion is constructed and sent to the service provider's assertion consumer service URL.
7. The `MvcExampleServiceProvider`'s `SAMLController` calls the `SAMLServiceProvider.ReceiveSSO` method to receive and process the SAML response.
8. The user is logged in automatically at the `MvcExampleServiceProvider`.

## ***10.3 OWIN Identity Provider and Service Provider***

The `OwinExampleIdentityProvider` and `OwinExampleServiceProvider` web applications demonstrate IdP-initiated and SP-initiated single sign-on.

The applications use OWIN and ASP.NET Identity v2.0.

The MVC examples require Visual Studio 2015 or later.

### **10.3.1 Installing the OWIN Identity Provider**

1. Using Visual Studio, build and publish the web application.
2. Open the Internet Information Services management console.
3. Under the default web site for the local computer, create an application with an alias of `OwinExampleIdentityProvider`.
4. For the physical path, browse to the directory where `OwinExampleIdentityProvider` was built and published.

5. Ensure the web application has been successfully installed by browsing to <http://localhost/OwinExampleIdentityProvider>.

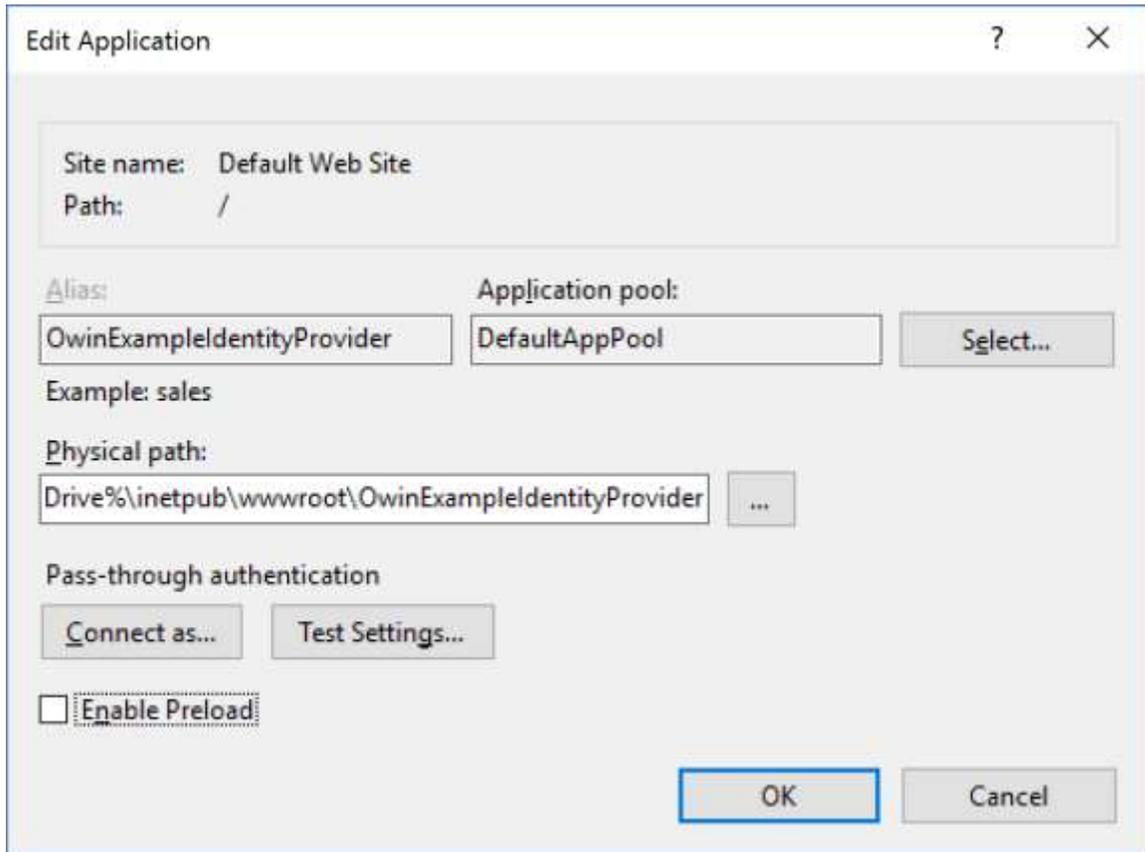


Figure 25 OwinExampleIdentityProvider Installation

### 10.3.2 Installing the OWIN Service Provider

1. Using Visual Studio, build and publish the web application.
2. Open the Internet Information Services management console.
3. Under the default web site for the local computer, create an application with an alias of OwinExampleServiceProvider.
4. For the physical path, browse to the directory where OwinExampleServiceProvider was built and published.
5. Ensure the web application has been successfully installed by browsing to <http://localhost/OwinExampleServiceProvider>.

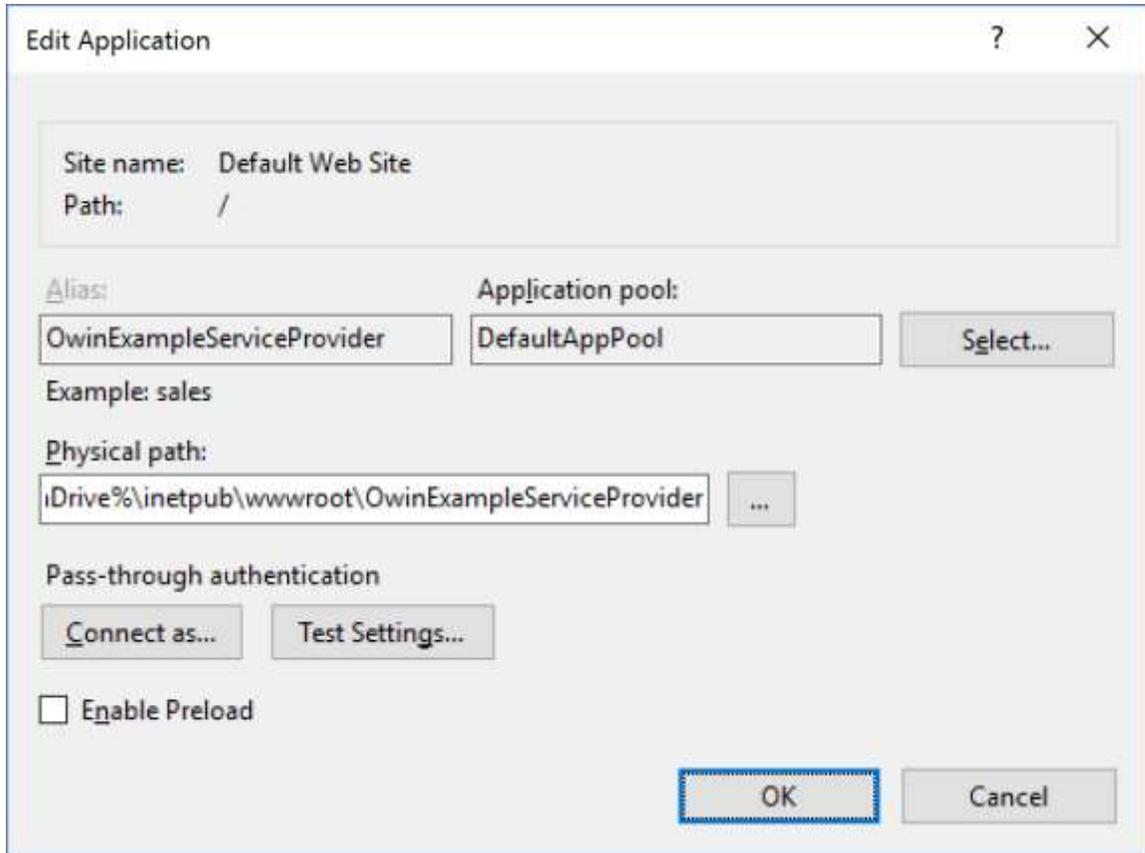


Figure 26 OwinExampleServiceProvider Installation

### 10.3.3 Configuring the OWIN Identity Provider

The identity provider configuration is contained within its web.config file's <appSettings> section and the saml.config file.

The saml.config includes the local identity provider configuration as well as partner service provider configuration.

The web.config's *PartnerSP* setting specifies the partner service provider for IdP-initiated SSO.

The default configuration supports single sign-on with the OwinExampleServiceProvider.

### 10.3.4 Configuring the OWIN Service Provider

The service provider configuration is contained within its web.config file's <appSettings> section and the saml.config file.

The saml.config includes the local service provider configuration as well as partner identity provider configuration.

The web.config's *PartnerIdP* setting specifies the partner identity provider for SP-initiated SSO.

The default configuration supports single sign-on with the OwinExampleIdentityProvider.

### 10.3.5 IdP-Initiated SSO from the OWIN Identity Provider

In this example, the user starts at the identity provider site and is attempting to access a protected resource on the service provider. Rather than performing a local login at the service provider, SSO is initiated with a local login occurring at the identity provider and the asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the service provider.

1. Browse to <http://localhost/OwinExampleIdentityProvider>.
2. If you haven't registered a user account previously, click the Register link to register a new user in the local database. Otherwise, click the Login link.
3. Once logged in, click the link to single sign-on to the service provider (see Figure 27).
4. You should then be presented with the service provider's home page (see Figure 28).

This means you've successfully completed a SAML v2.0 SSO and are logged in at the service provider with your identity provider user name.

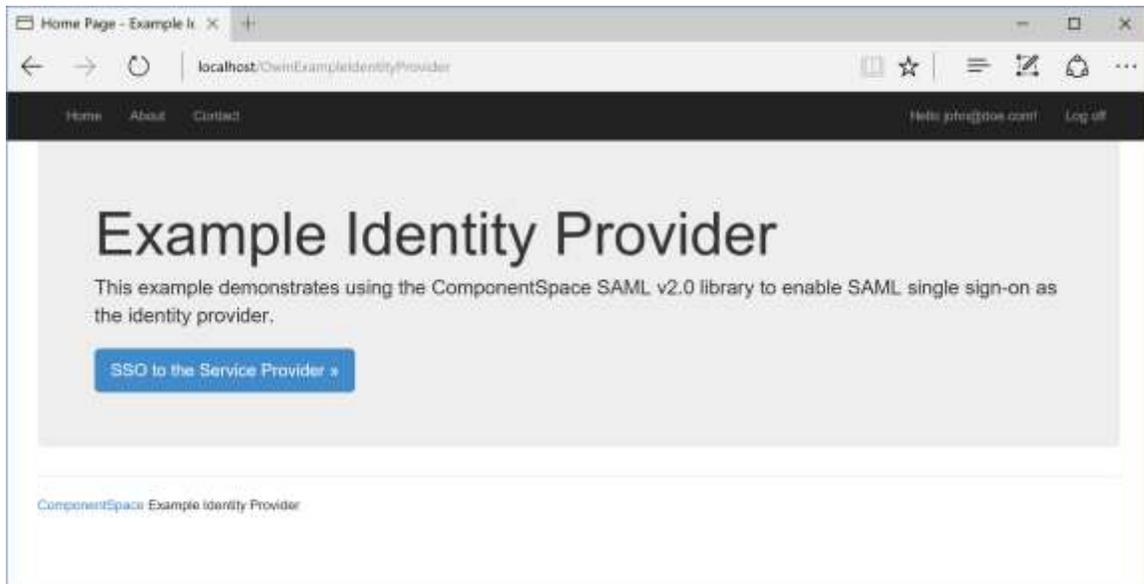


Figure 27 OWIN Example Identity Provider Home Page

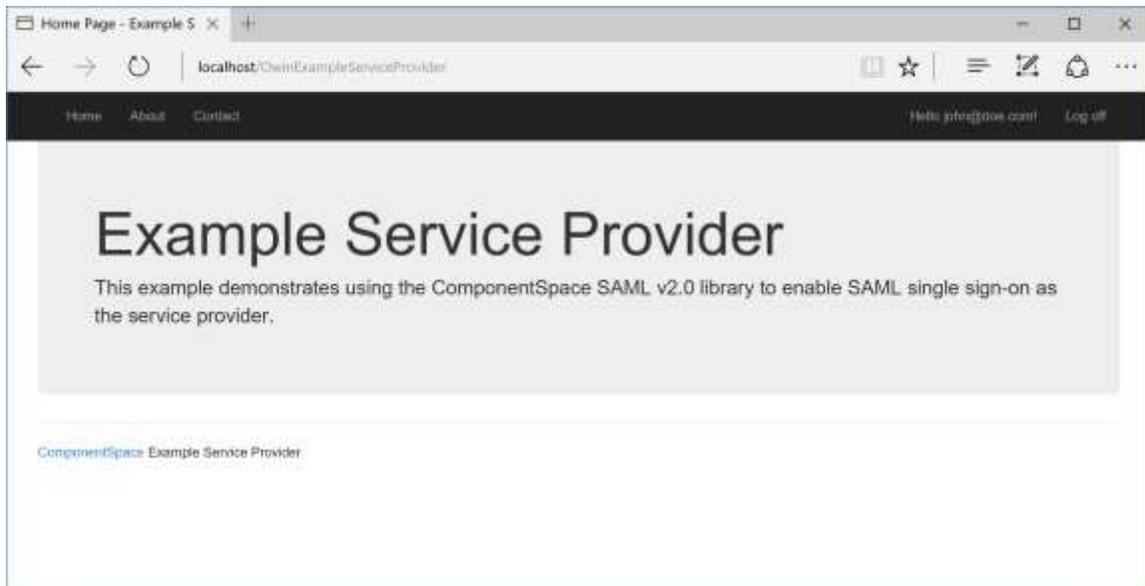


Figure 28 OWIN Example Service Provider Home Page

### 10.3.6 SP-Initiated SSO from the OWIN Service Provider

In this example, the user starts at the service provider site and is attempting to access a protected resource on the service provider. Rather than performing a local login at the service provider, SSO is initiated with a local login occurring at the identity provider and the asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the service provider.

1. Browse to <http://localhost/OwinExampleServiceProvider>.
2. Click the Login link.
3. You should then be presented with the service provider's login.
4. Click the link to single sign-on to the identity provider (see Figure 29).
5. You should be presented with the identity provider's login page (see Figure 30).

If you are not then you must already have logged in at the identity provider. To force a login, close the browser and start again.

6. Login or register if you don't have an account at the identity provider.
7. You should then be presented with the service provider's home page.

This means you've successfully completed a SAML v2.0 SSO and are logged in at the service provider with your identity provider user name.

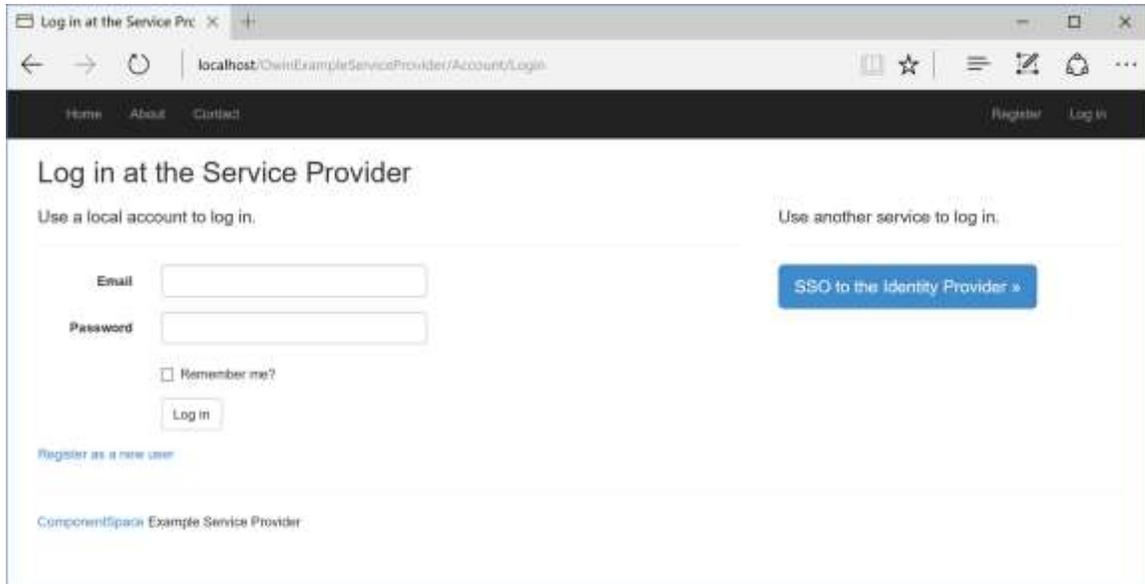


Figure 29 OWIN Example Service Provider Login Page

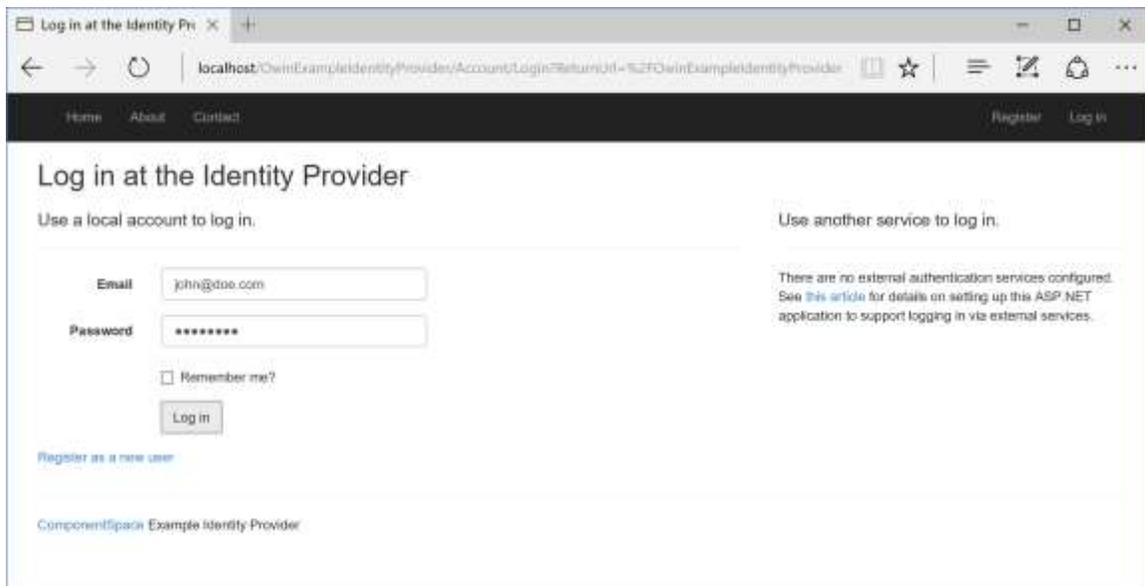


Figure 30 OWIN Example Identity Provider Login Page

### 10.3.7 Code Walkthrough - IdP-Initiated SSO

The following section follows the flow of IdP-initiated SSO between the OwinExampleIdentityProvider and OwinExampleServiceProvider.

1. The user clicks the link on the OwinExampleIdentityProvider's home page and the HomeController calls the SAMLIdentityProvider.InitiateSSO method to initiate SSO.

2. A SAML response containing a SAML assertion is constructed and sent to the service provider's assertion consumer service URL.
3. The OwinExampleServiceProvider's SAMLController calls the SAMLServiceProvider.ReceiveSSO method to receive and process the SAML response.
4. The user is logged in automatically at the OwinExampleServiceProvider.

### **10.3.8 Code Walkthrough - SP-Initiated SSO**

The following section follows the flow of SP-initiated SSO between the OwinExampleIdentityProvider and OwinExampleServiceProvider.

1. The user clicks the link on the OwinExampleServiceProvider's login page and the AccountController calls the SAMLServiceProvider.InitiateSSO method to initiate SSO.
2. An authentication request is constructed and sent to the identity provider's SSO service URL.
3. The OwinExampleIdentityProvider's SAMLController calls the SAMLIdentityProvider.ReceiveSSO method to receive and process the authentication request.
4. The user is prompted to login at the OwinExampleIdentityProvider if not already logged in.
5. The OwinExampleIdentityProvider's SAMLController calls the SAMLIdentityProvider.SendSSO method.
6. A SAML response containing a SAML assertion is constructed and sent to the service provider's assertion consumer service URL.
7. The OwinExampleServiceProvider's SAMLController calls the SAMLServiceProvider.ReceiveSSO method to receive and process the SAML response.
8. The user is logged in automatically at the OwinExampleServiceProvider.

## **10.4 ADFS Interoperability**

The Web Forms and MVC example identity and service providers demonstrate single sign-on with Windows Active Directory Federation Services (ADFS).

The following sections describe the configuration for the Web Forms example identity provider and service provider but, with the appropriate changes, apply equally to the MVC examples.

Refer to sections 10.1 and 10.2 for instructions on installing and configuring the Web Forms and MVC example identity and service providers.

### **10.4.1 Miscellaneous Configuration**

For the purposes of these examples, the host name of the ComponentSpace example applications is [cs.test](http://cs.test) and the host name of the ADFS server is [adfs.test](http://adfs.test).

If using these host names, update the Windows\System32\drivers\etc\hosts file on the test and ADFS servers to include entries for cs.test and adfs.test. For example:

```
192.168.1.20      cs.test
192.168.1.21      adfs.test
```

## 10.4.2 Configuring the Service Provider

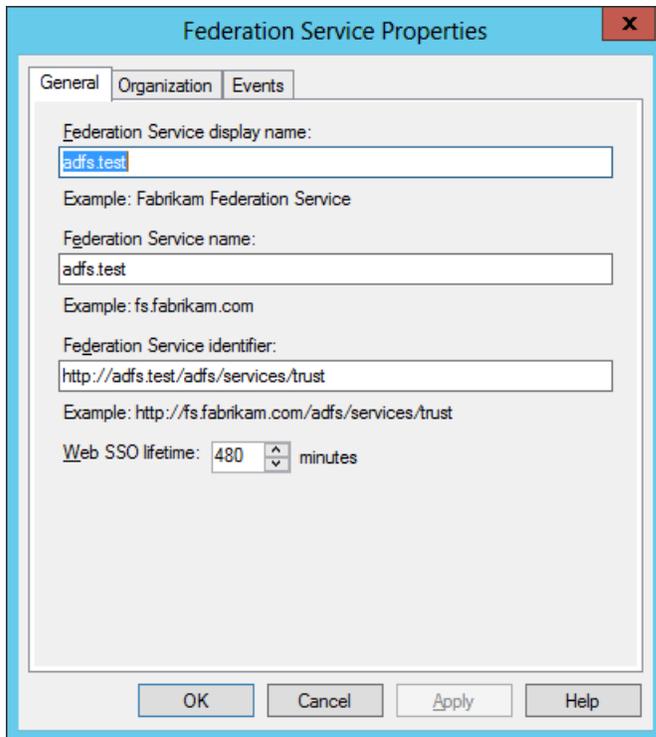
The following sections describe interoperability between the example service provider and ADFS acting as the claims provider (i.e. identity provider).

The saml.config file includes the following entry for the ADFS partner identity provider.

```
<PartnerIdentityProvider Name="http://adfs.test/adfs/services/trust"
  SignAuthnRequest="true"
  WantSAMLResponseSigned="false"
  WantAssertionSigned="true"
  WantAssertionEncrypted="true"
  UseEmbeddedCertificate="true"
  SingleSignOnServiceUrl=
    "https://adfs.test/adfs/ls/" />
```

The name must match with the issuer name ADFS uses in the returned SAML response. For example, if ADFS is deployed to the myadfs server then the name must be <http://myadfs/adfs/services/trust>.

The ADFS federation services properties lists the federation service identifier.



The *UseEmbeddedCertificate* flag is set to simplify the configuration. If not set then the ADFS signature certificate needs to be imported to the service provider and configured in the SAML configuration certificate manager.

The web.config's *PartnerIdP* setting specifies the partner identity provider for SP-initiated SSO and should be set to `http://www.idp.com/adfs/services/trust`.

```
<add key="PartnerIdP" value="http://adfs.test/adfs/services/trust"/>
```

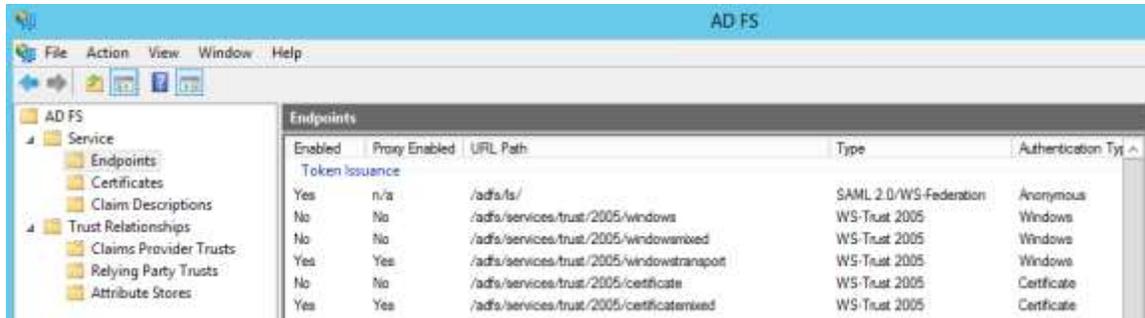
### 10.4.3 Configuring ADFS – Adding a Relying Party

In the ADFS terminology, the service provider is a relying party. Using the ADFS management console, add a relying party trust for the service provider.

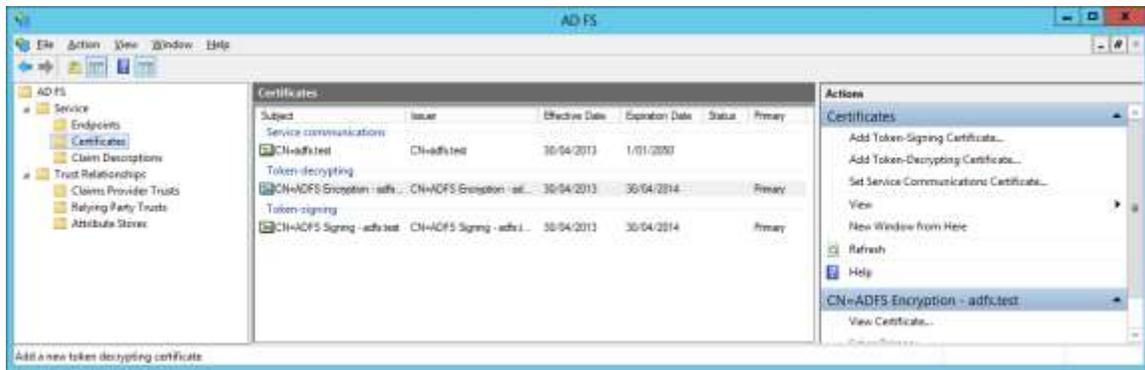
Note that strings in ADFS, including URLs, are case sensitive.

Confirm that the `/adfs/ls` endpoint for SAML v2.0 exists. If it doesn't, refer to the ADFS documentation.

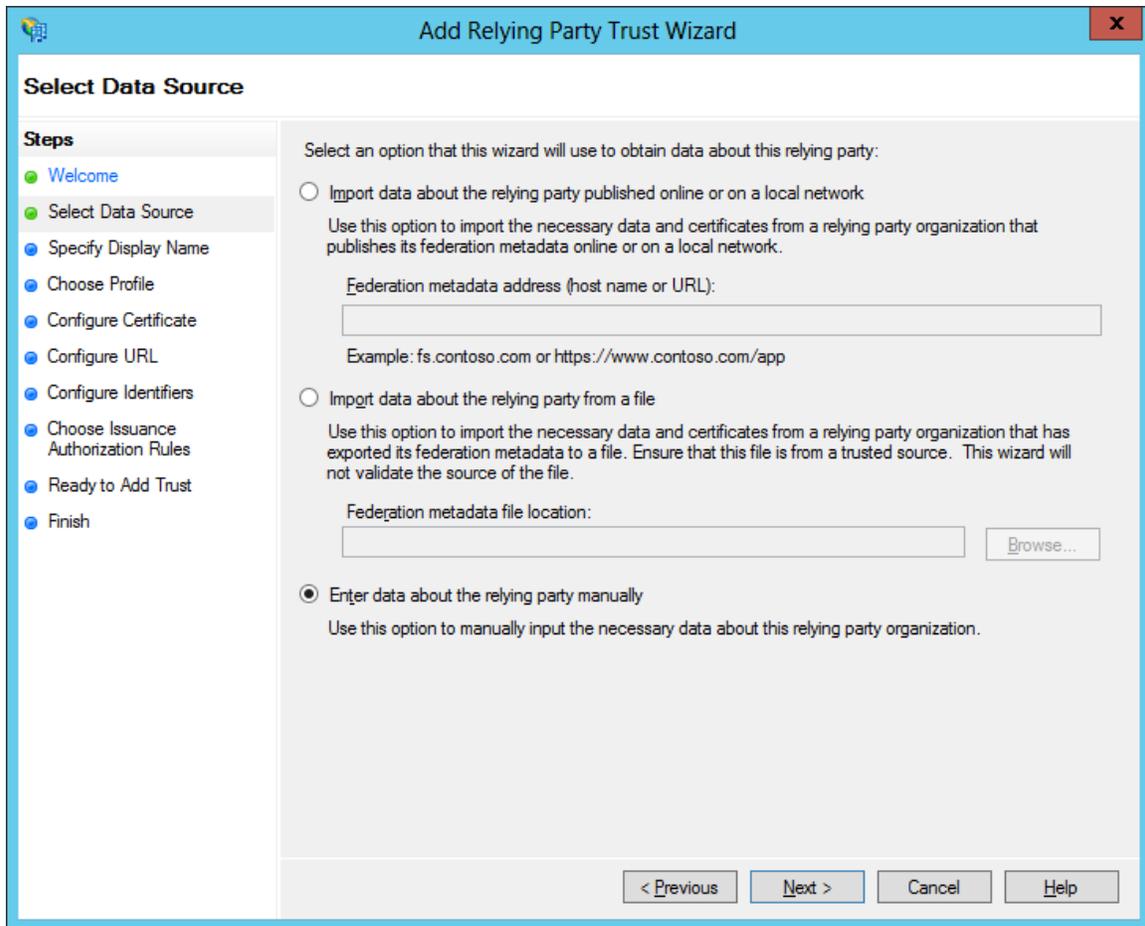
## ComponentSpace SAML v2.0 for .NET Developer Guide



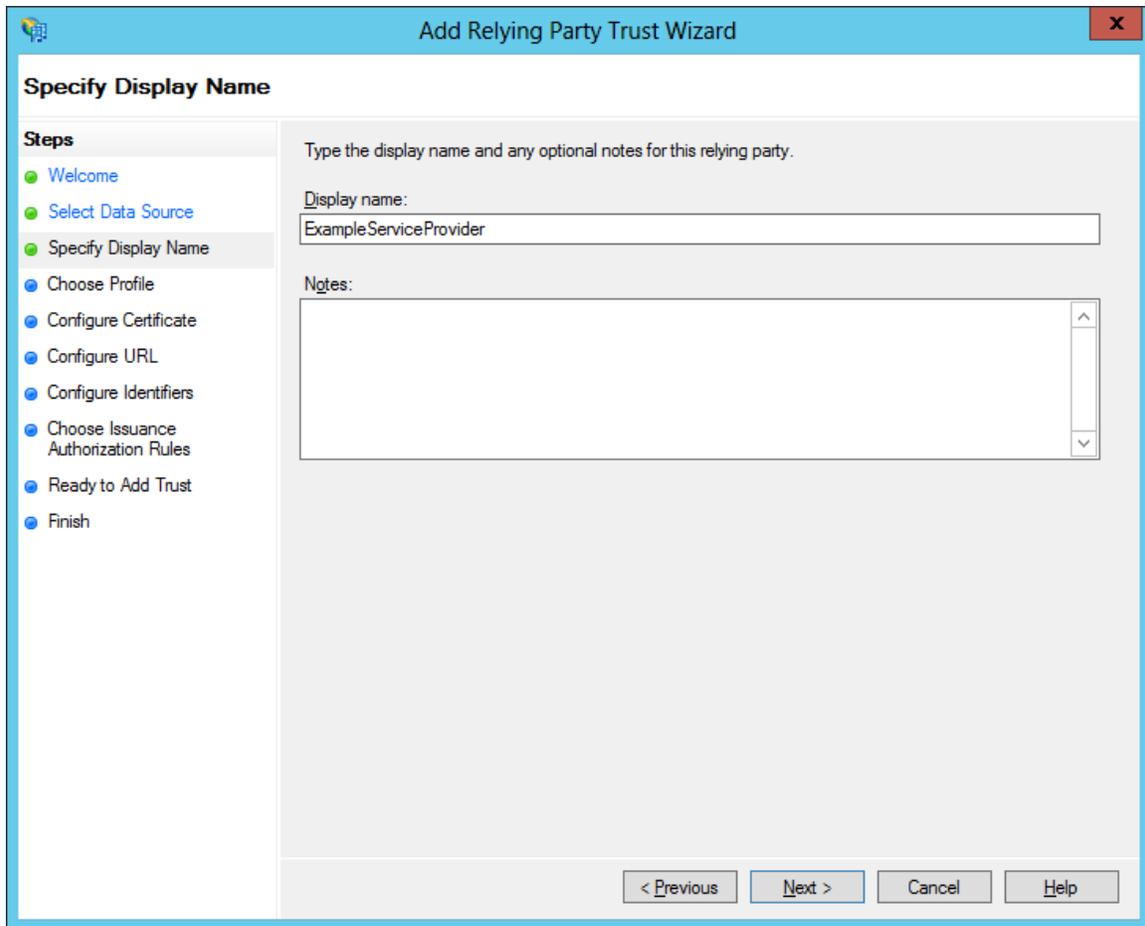
Confirm that the service communications, token decrypting and token encrypting certificates exist. If they don't, refer to the ADFS documentation.



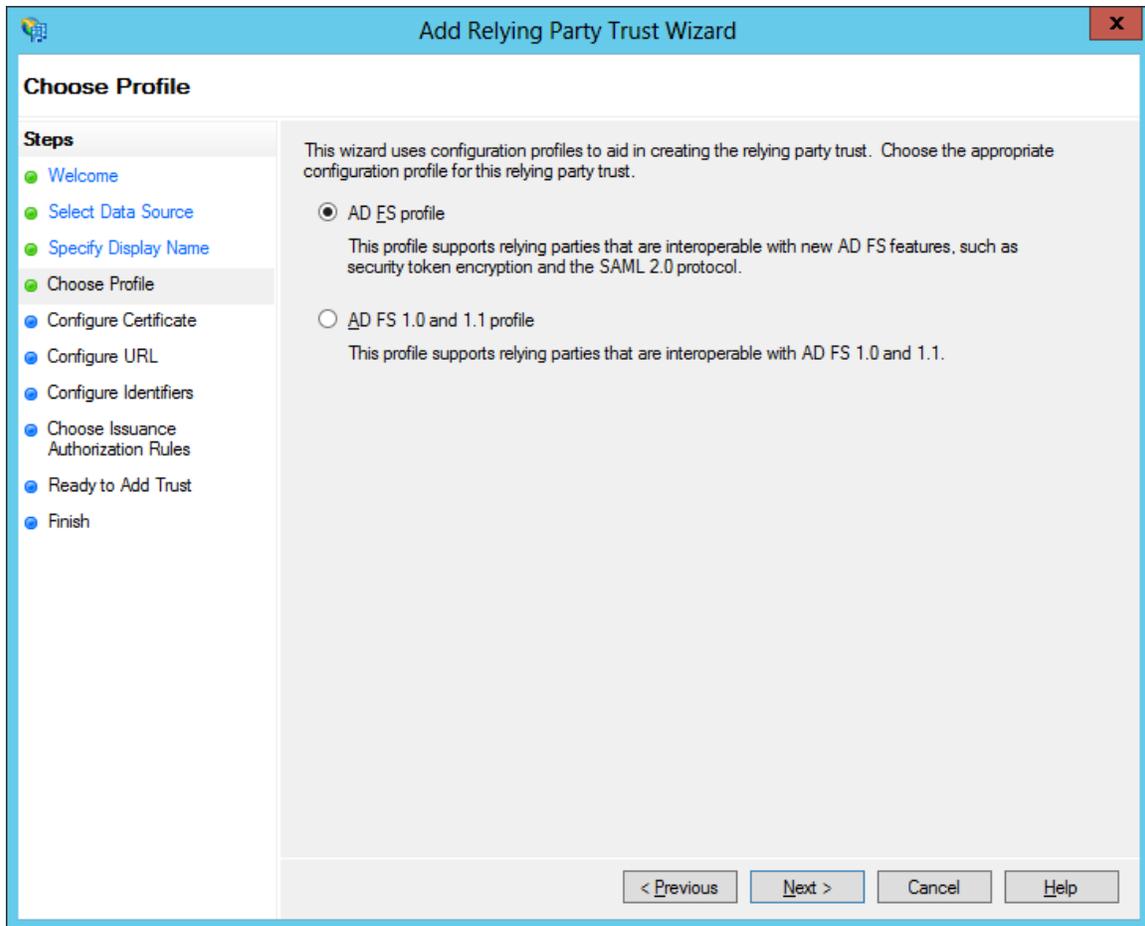
Add a relying party trust and select the option to enter the relying party information manually.



Specify a display name. The display name does not have to match with any other configuration.

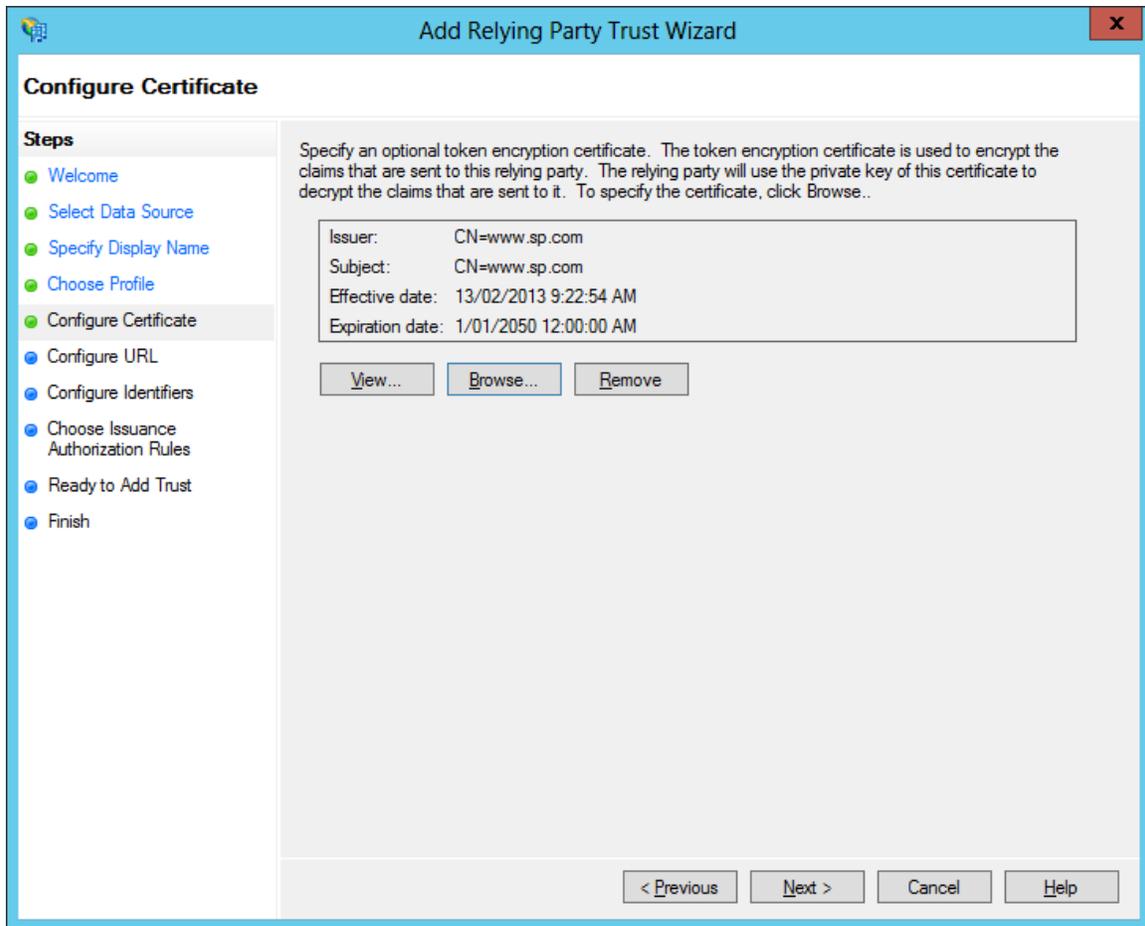


Choose the ADFS profile.



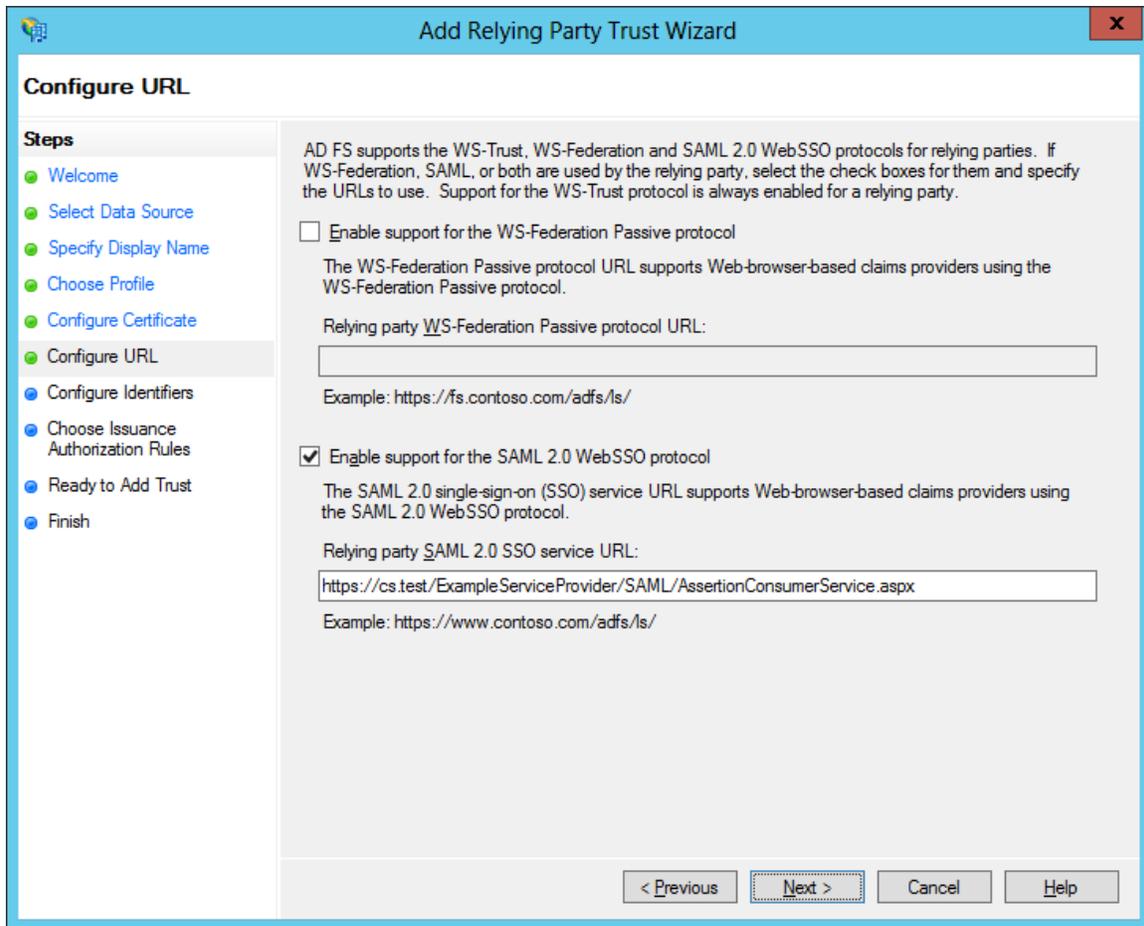
Browse to sp.cer to specify it as the token encryption certificate. Ignore any warnings about the key length.

The token encryption certificate is used to encrypt the SAML assertion. The service provider decrypts the SAML assertion using the associated private key.



Enable support for SAML v2.0 and specify the service provider's assertion consumer service URL. ADFS sends the SAML response to this URL. For example:

<https://cs.test/ExampleServiceProvider/SAML/AssertionConsumerService.aspx>



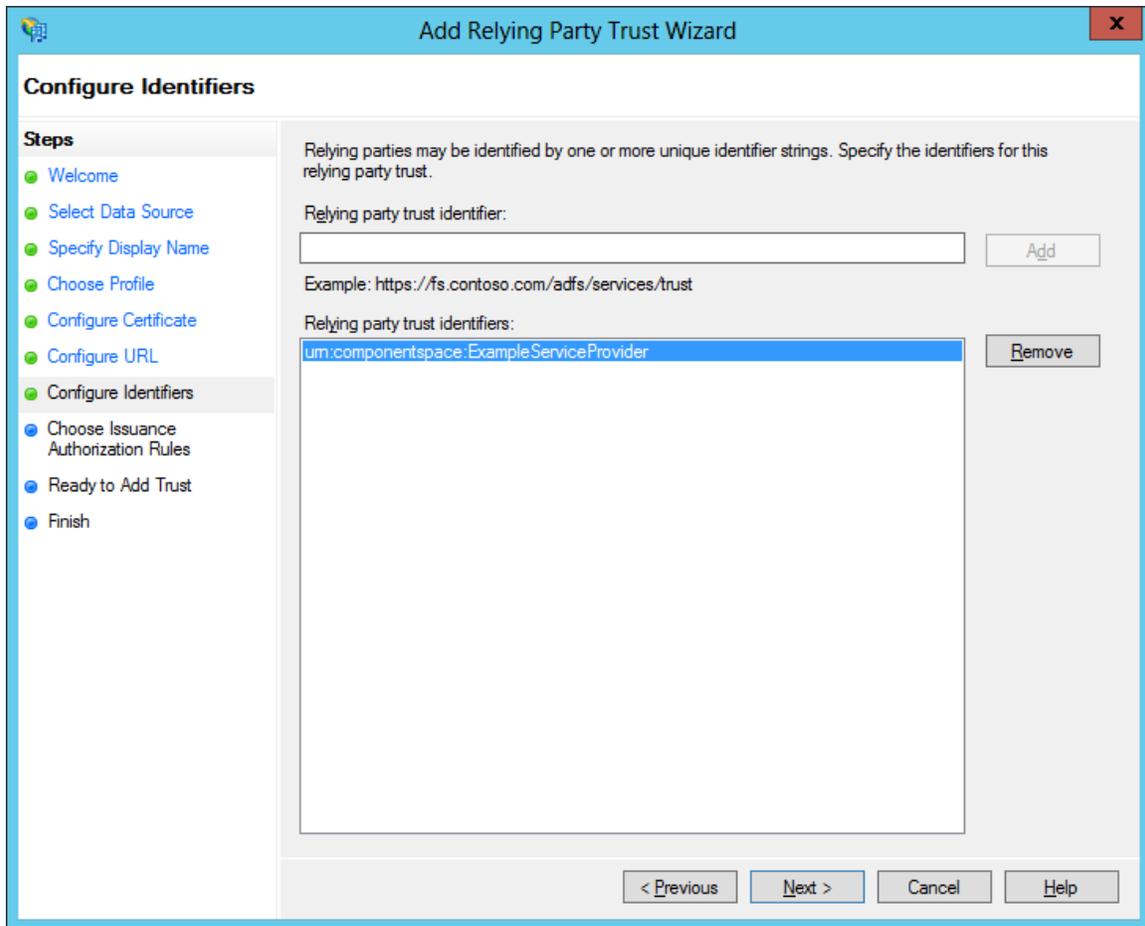
Specify the relying party trust identifier. This identifier must match the issuer field in the authn request sent by the service provider. The ServiceProvider name attribute in the saml.config configuration file is used as the issuer and so this name and the relying party trust identifier must match.

For example, if the saml.config includes:

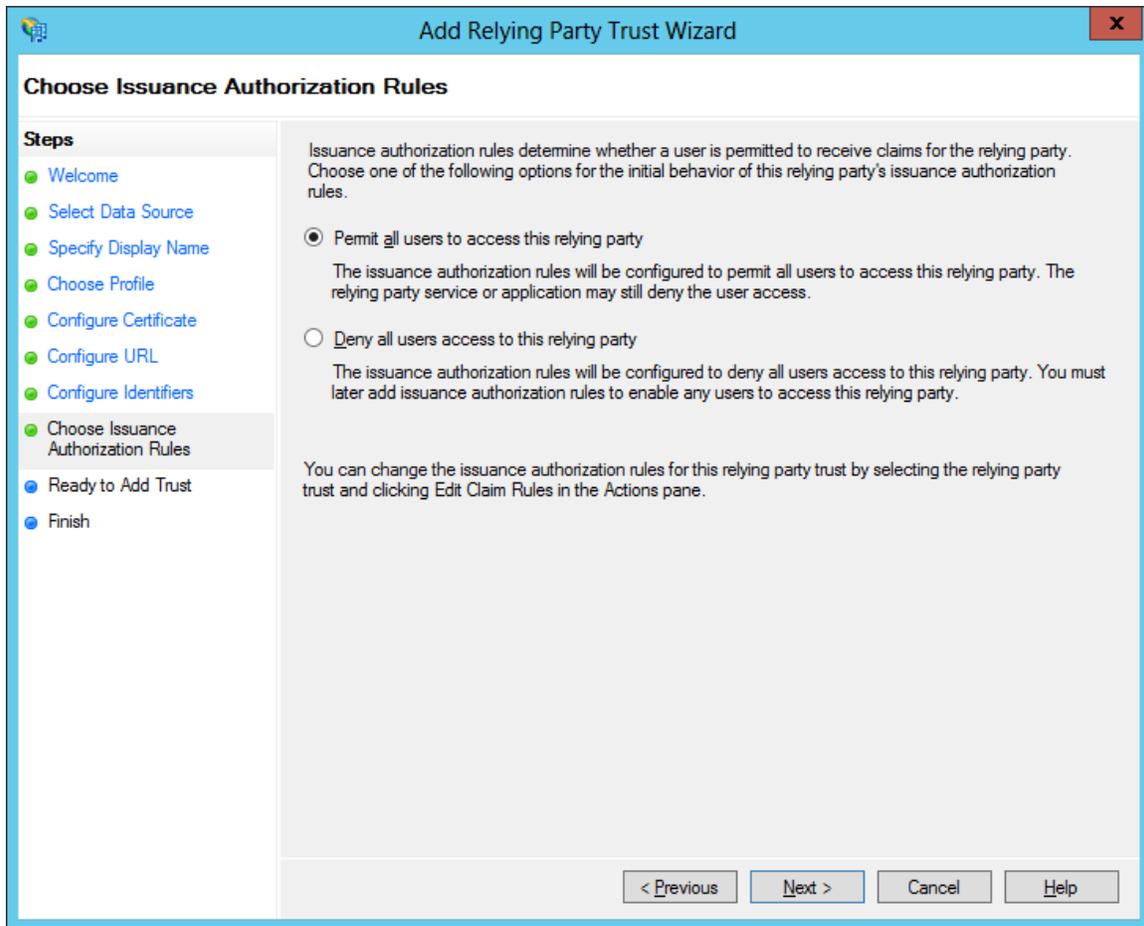
```
<ServiceProvider Name="urn:componentspace:ExampleServiceProvider"
  AssertionConsumerServiceUrl=
    "~/SAML/AssertionConsumerService.aspx"/>
```

Then the relying party trust identifier must be:

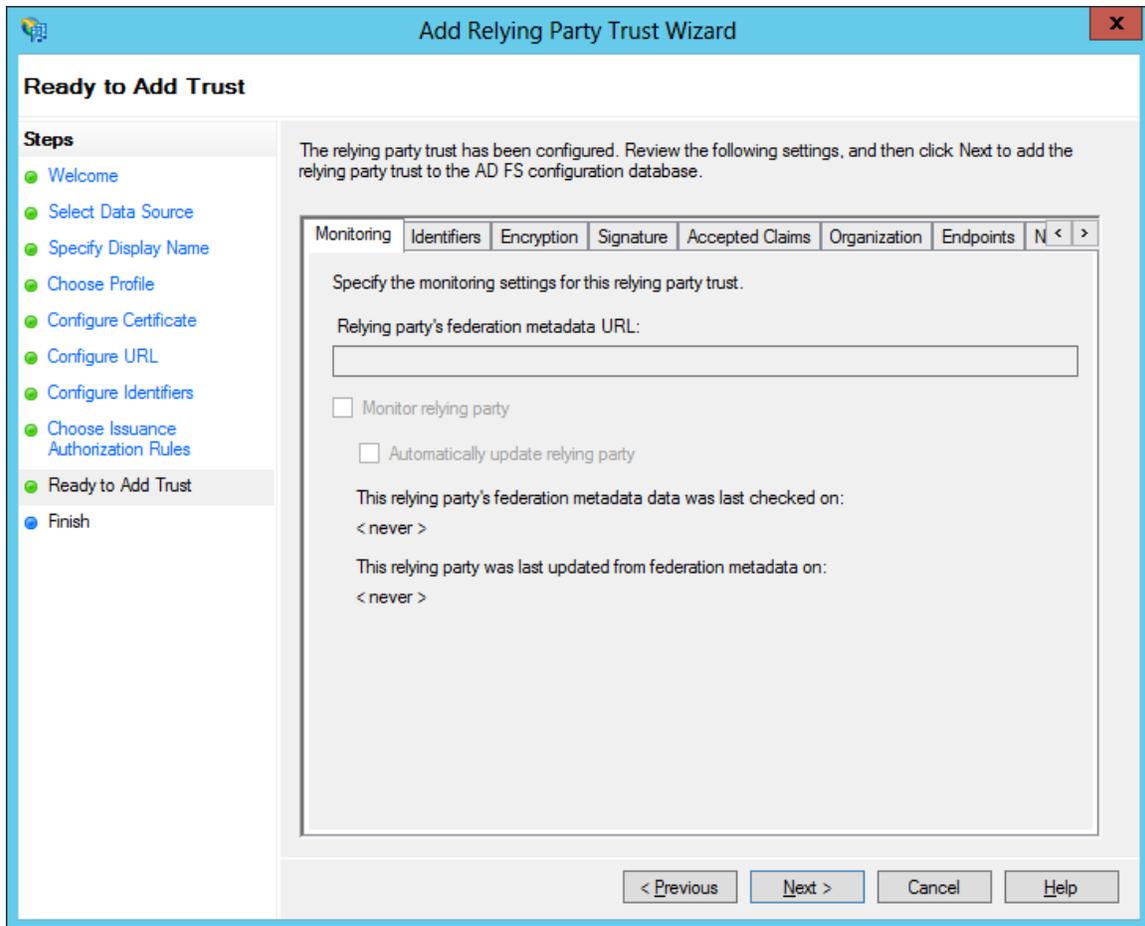
urn:componentspace:ExampleServiceProvider.



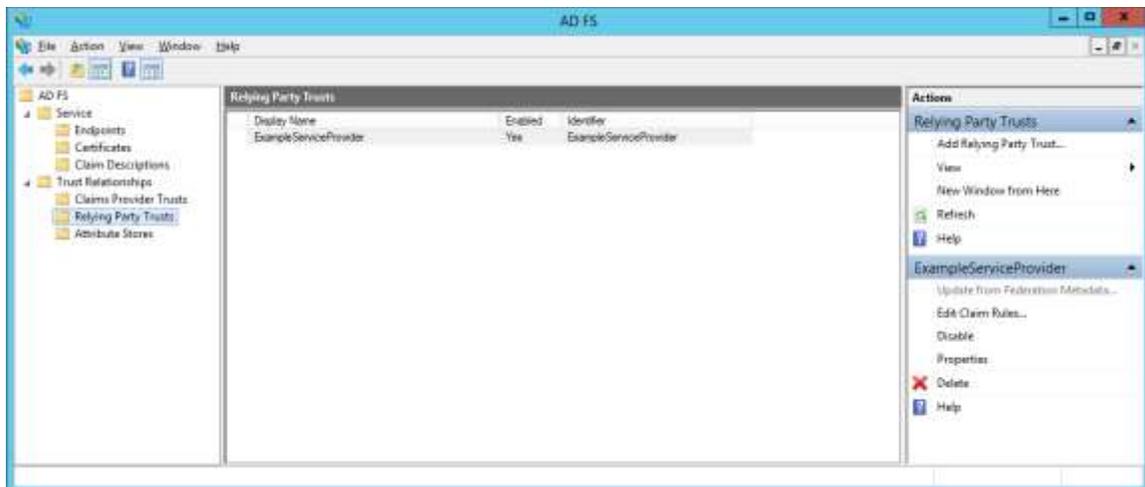
Permit all users access to this relying party.



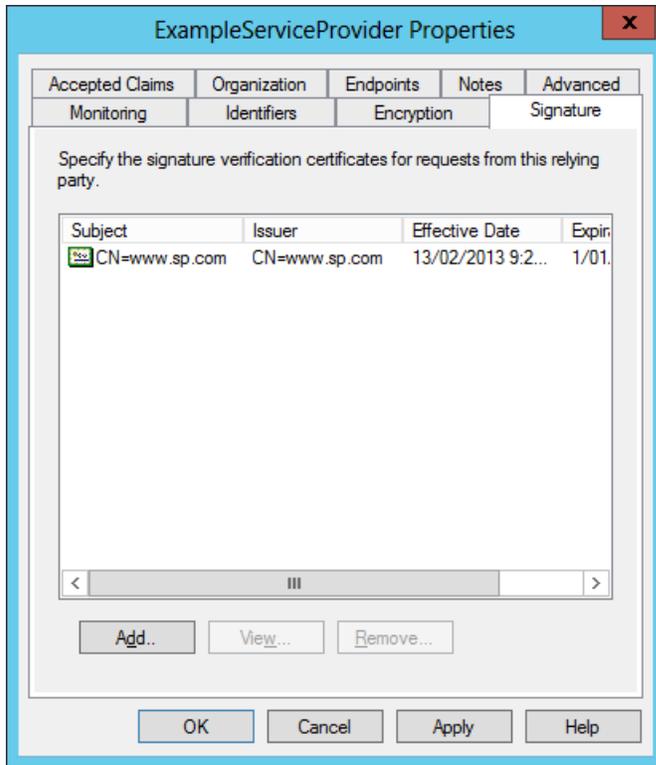
Review the configuration and close the wizard.



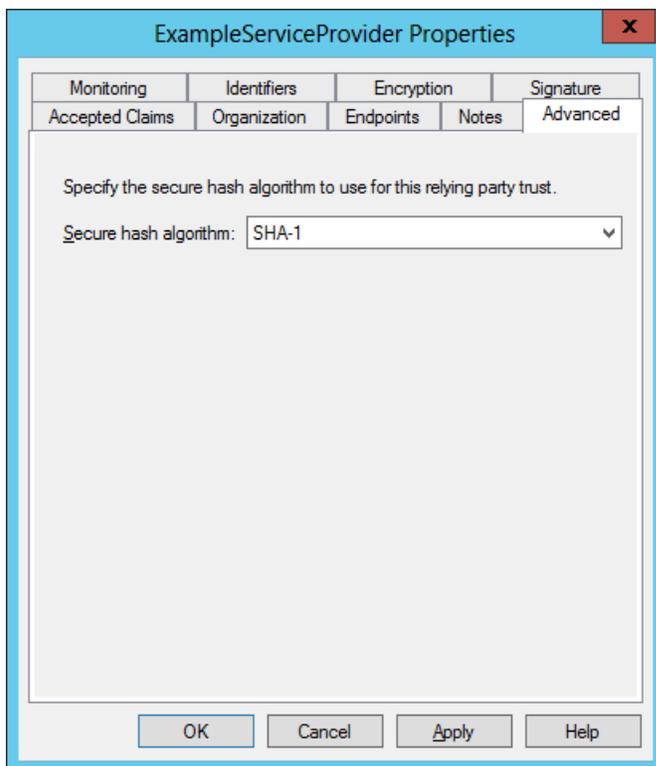
The service provider should be included in the list of relying party trusts.



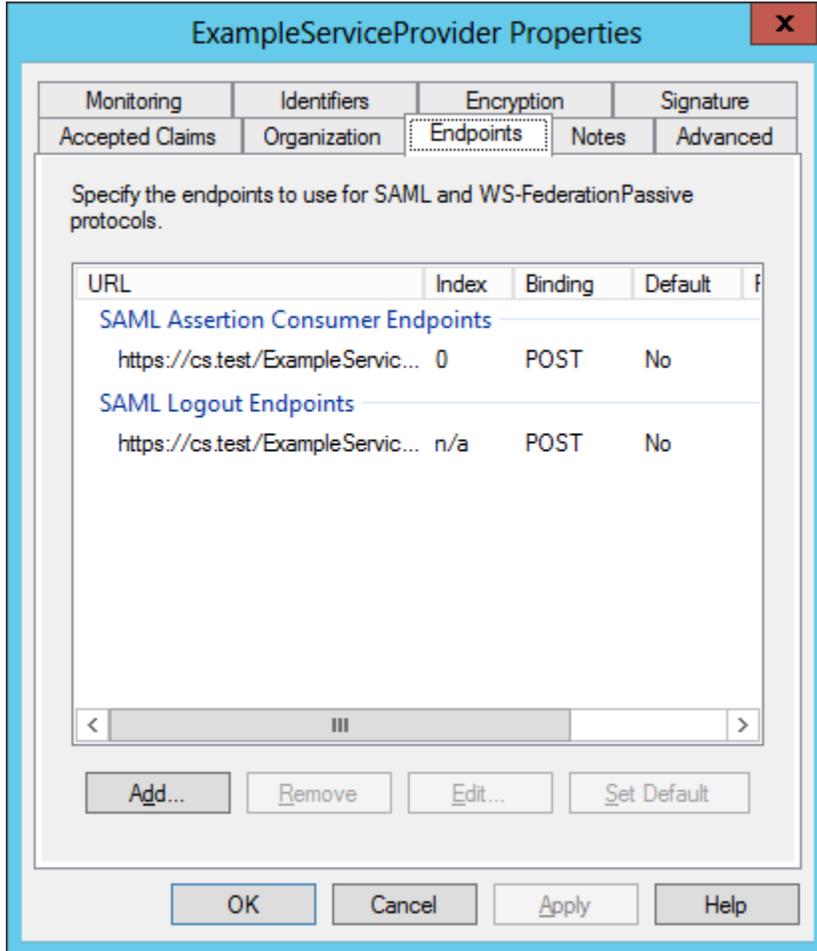
The authn request sent by the service provider is signed. To specify the certificate to use to verify the signature, open the relying party trusts' properties and, under the Signature tab, add the service provider certificate.



Although the SAML v2.0 component supports SHA-256 signatures, for this example SHA-1 is used. To specify this, under the Advanced tab, select SHA-1.



To support SAML logout, specify the logout endpoint.



**Edit Endpoint**

Endpoint type: SAML Logout

Binding: POST

Index: 0

URL: https://cs.test/ExampleServiceProvider/SAML/SLOService.aspx  
Example: https://sts.contoso.com/adfs/ls

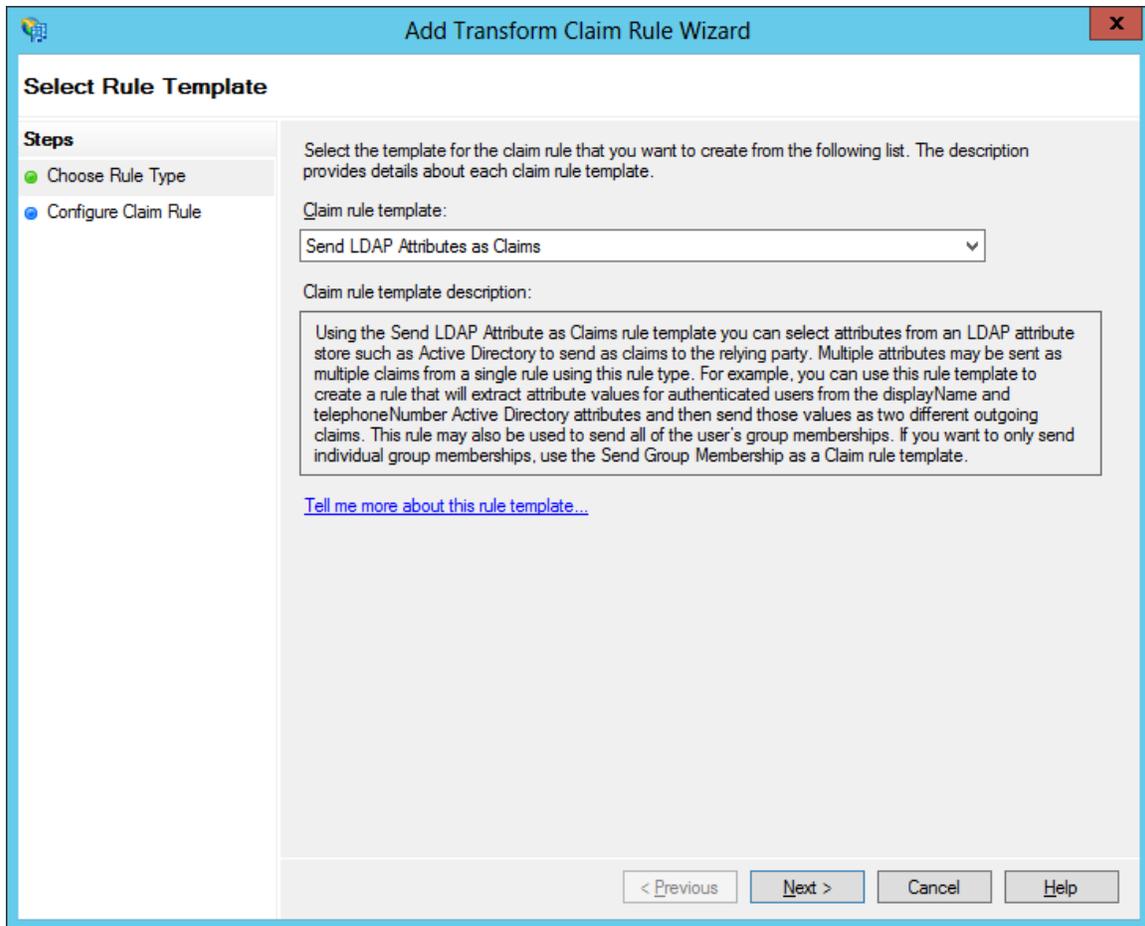
Response URL:   
Example: https://sts.contoso.com/logout

OK Cancel Help

Note that even after logout, ADFS may persist the authentication session depending on which local authentication type is used. For example, if Integrated authentication is used then after logout the next login, in the same browser session, uses the persisted authentication session and the user doesn't have to re-enter their credentials. If Forms authentication is used then a login is required after logout. The following section from the adfs/ls web.config shows the Forms authentication taking precedence over the Integrated authentication.

```
<localAuthenticationTypes>
  <add name="Forms" page="FormsSignIn.aspx" />
  <add name="Integrated" page="auth/integrated/" />
  <add name="TlsClient" page="auth/sslclient/" />
  <add name="Basic" page="auth/basic/" />
</localAuthenticationTypes>
```

Edit the claim rules and add a rule.



Map the Active Directory user principal name to the outgoing Name ID. Map additional Active Directory attributes to include in the SAML assertion as SAML attributes.

**Add Transform Claim Rule Wizard**

---

**Configure Rule**

**Steps**

- Choose Rule Type
- Configure Claim Rule

You can configure this rule to send the values of LDAP attributes as claims. Select an attribute store from which to extract LDAP attributes. Specify how the attributes will map to the outgoing claim types that will be issued from the rule.

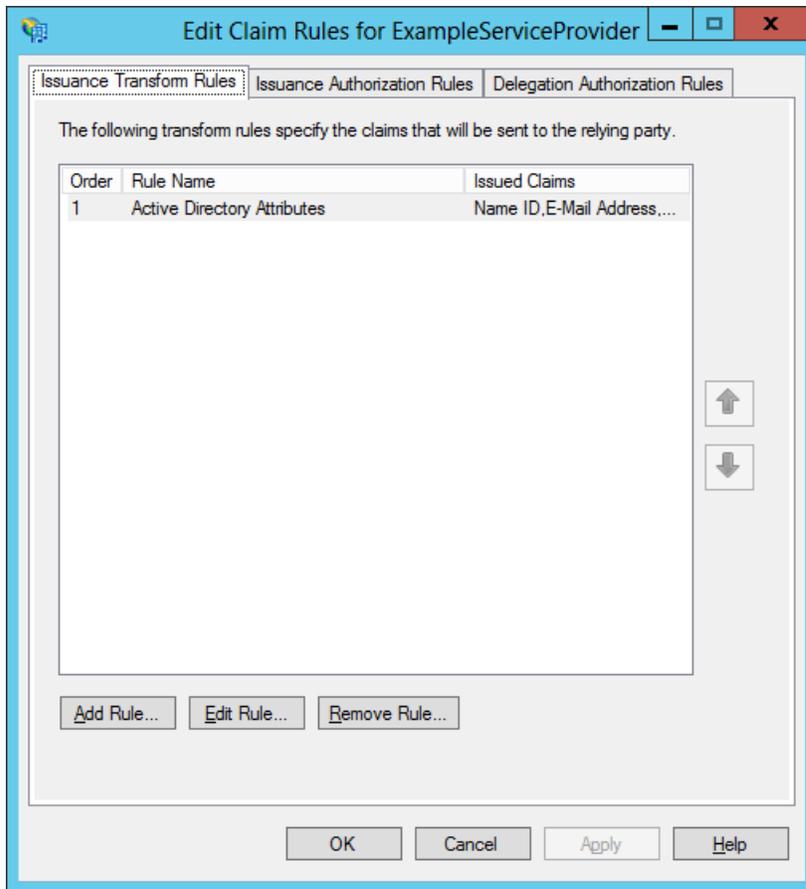
Claim rule name:

Rule template: Send LDAP Attributes as Claims

Attribute store:

Mapping of LDAP attributes to outgoing claim types:

	LDAP Attribute (Select or type to add more)	Outgoing Claim Type (Select or type to add more)
	User-Principal-Name	Name ID
	E-Mail-Addresses	E-Mail Address
	Given-Name	Given Name
▶	Surname	Surname
*		



ADFS should now be ready to communicate with the example service provider.

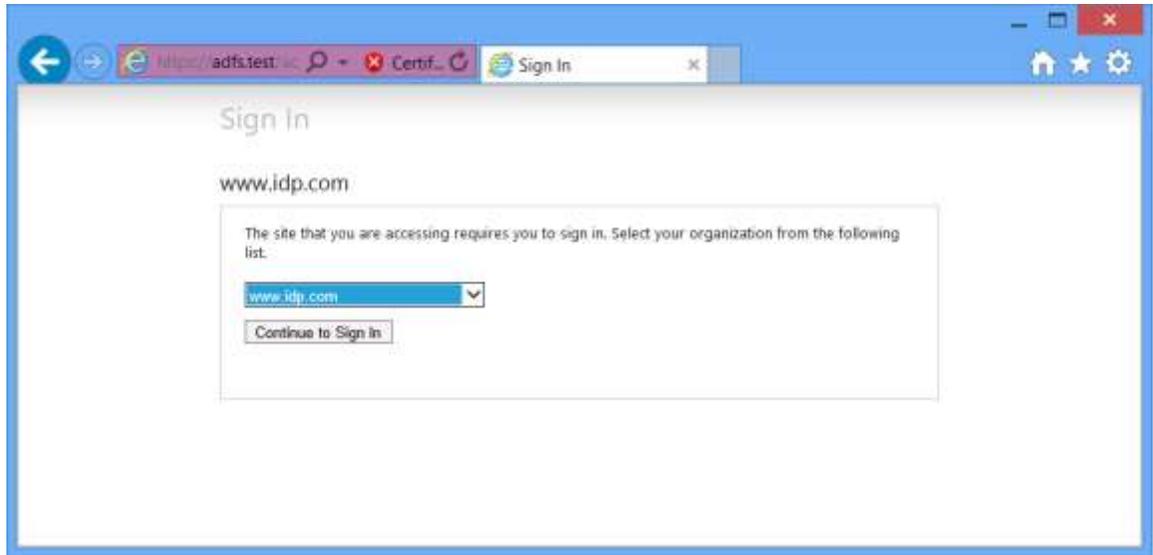
To review the metadata published by ADFS browse to:

<https://adfs.test/FederationMetadata/2007-06/FederationMetadata.xml>

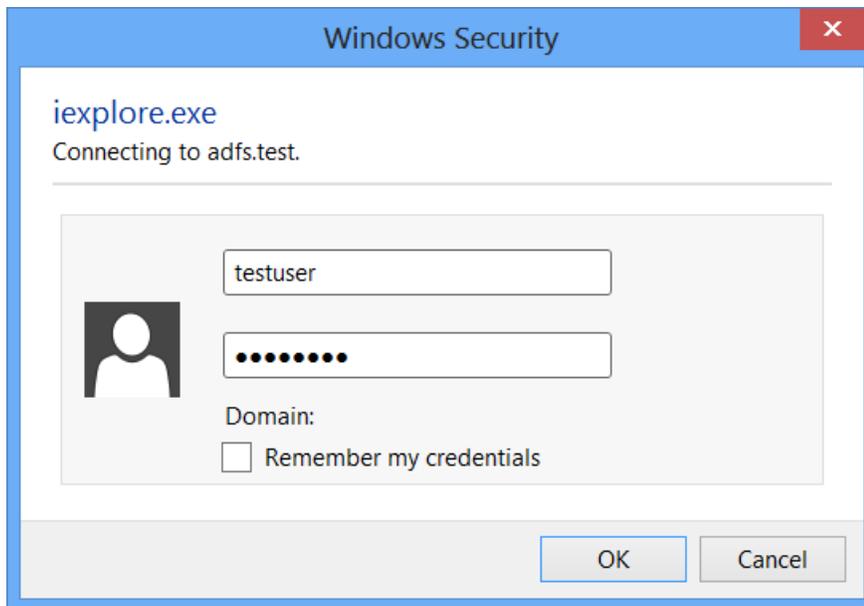
#### 10.4.4 Running the Service Provider with SP-Initiated SSO

In this example, the user is attempting to access a protected resource on the service provider and, rather than performing a local login at the service provider, SSO is initiated with a local login occurring at the ADFS identity provider and the asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the service provider.

1. Browse to <https://cs.test/ExampleServiceProvider>, ignoring any browser certificate warnings.
2. If more than one claim provider is configured on ADFS, you will be presented with the following page. Select the appropriate claim provider for authentication against Active Directory. For example, adfs.test.

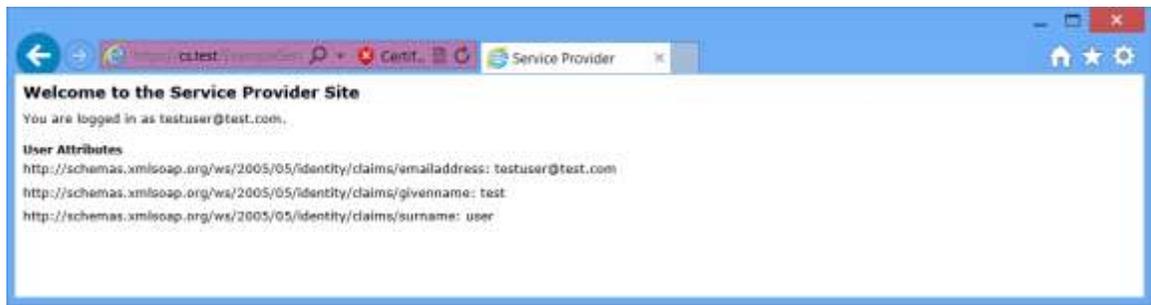


3. You should then be presented with the identity provider login prompt.



4. Login using the user name and password of a user defined in Active Directory.
5. You should then be presented with the service provider's default page.

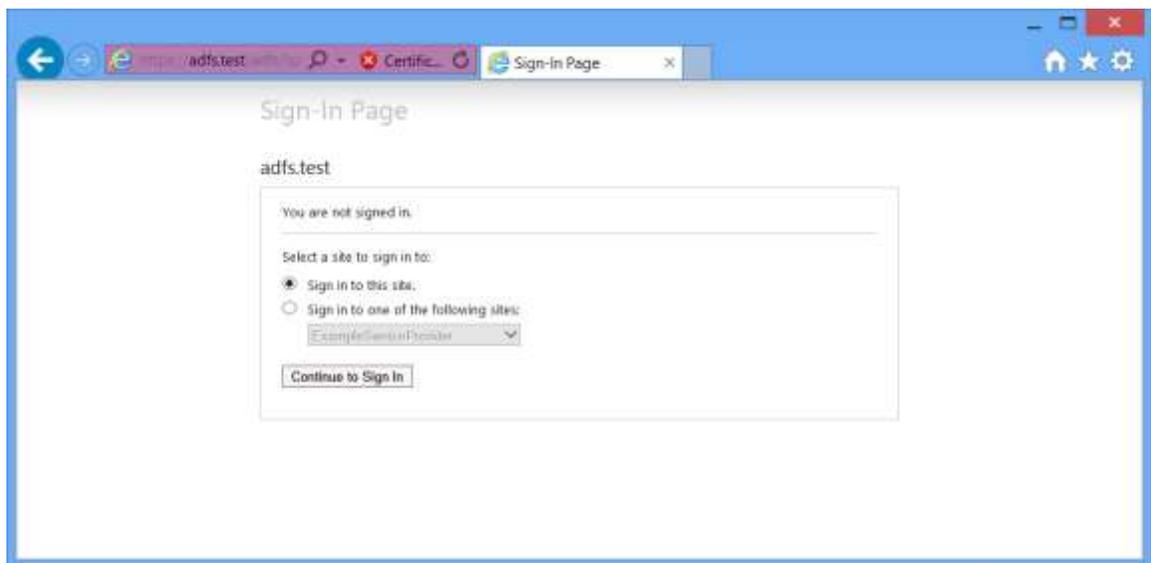
This means you've successfully completed a SAML v2.0 SSO and are logged in at the service provider with your identity provider user name.



### 10.4.5 Running the Service Provider with IdP-Initiated SSO

In this example, the user logs in at ADFS and initiates SSO to the service provider. The asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the service provider.

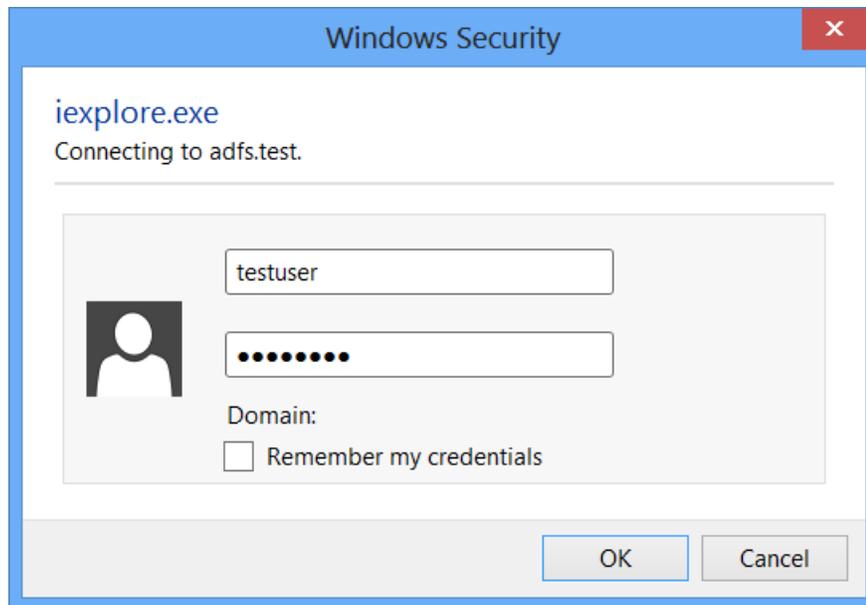
1. Browse to <https://adfs.test/adfs/ls/IdpInitiatedSignon.aspx>, ignoring any browser certificate warnings.
2. You should then be presented with the identity provider sign-in page.



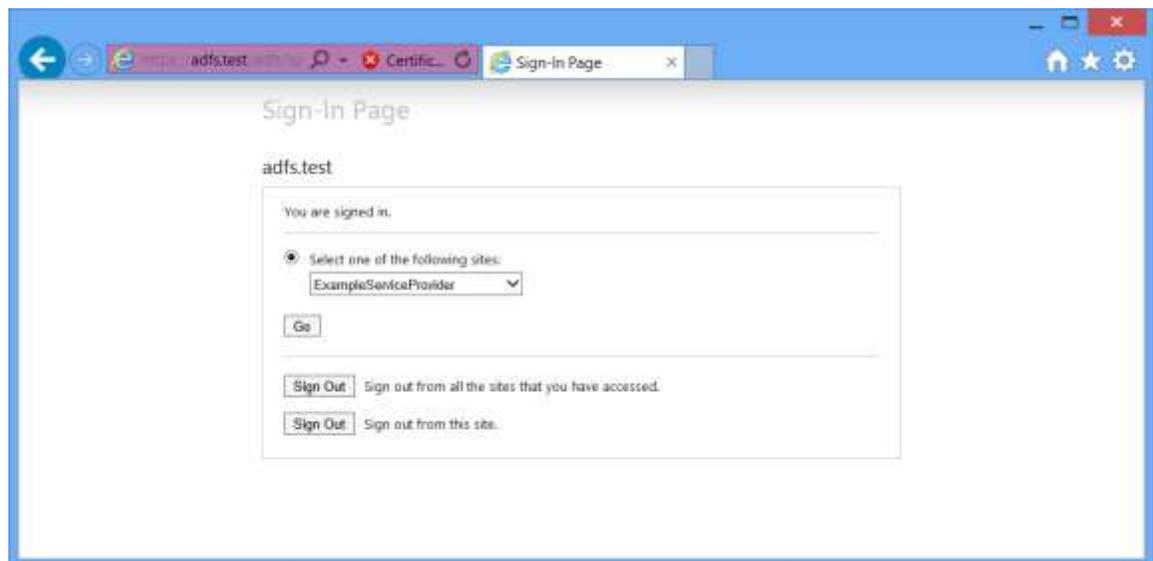
3. Select the “sign in to this site” radio button and click the continue button.

Alternatively, selecting the “sign in to one of the following sites” radio button performs SSO to the selected service provider immediately after login.

You should then be presented with the identity provider login prompt.

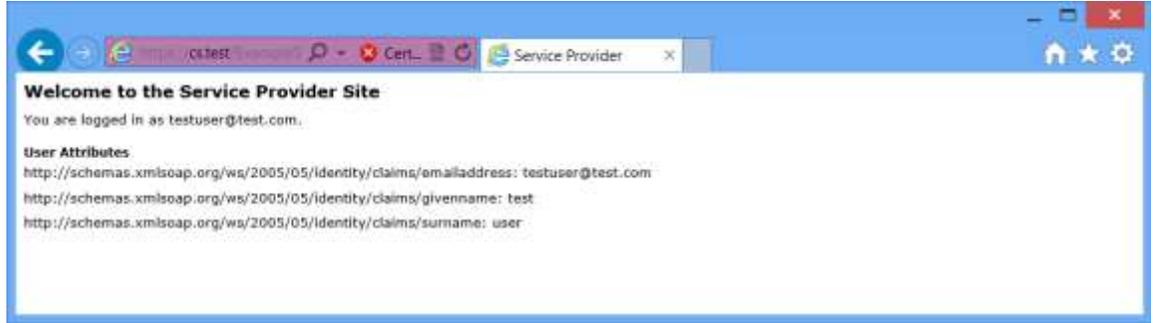


4. Login using the user name and password of a user defined in Active Directory.
5. Select the service provider and click Go to initiate SSO.



6. You should then be presented with the service provider's default page.

This means you've successfully completed a SAML v2.0 SSO and are logged in at the service provider with your identity provider user name.



### 10.4.6 Configuring the Identity Provider

The following sections describe interoperability between the example identity provider and ADFS acting as the relying party (i.e. service provider).

The `saml.config` file includes the following entry for the ADFS partner service provider.

```
<PartnerServiceProvider Name="http://adfs.test/adfs/services/trust"
    WantAuthnRequestSigned="false"
    SignSAMLResponse="false"
    SignAssertion="true"
    EncryptAssertion="false"
    AssertionConsumerServiceUrl=
        "https://adfs.test/adfs/ls/" />
```

The name must match with the issuer name ADFS uses in the authn request. For example, if ADFS is deployed to the `myadfs` server then the name must be `http://myadfs/adfs/services/trust`.

The `web.config`'s `PartnerSP` setting specifies the partner service provider for IdP-initiated SSO and should be set to `http://adfs.test/adfs/services/trust`.

```
<add key="PartnerSP" value="http://adfs.test/adfs/services/trust" />
```

The `web.config`'s `TargetUrl` setting specifies, for IdP-initiated SSO, the relying party configured in ADFS and should be set to `RPID=ExampleServiceProvider`.

The `RPID` syntax is specific to ADFS. If not specified then ADFS will convert the IdP-initiated SSO into SP-initiated SSO.

```
<add key="TargetUrl" value="RPID=ExampleServiceProvider" />
```

### 10.4.7 Configuring ADFS – Adding a Claims Provider

To support IdP-initiated SSO, edit the ADFS `web.config` at `C:\inetpub\adfs\ls`. In the `microsoft.identityServer.web`, add the following entry:

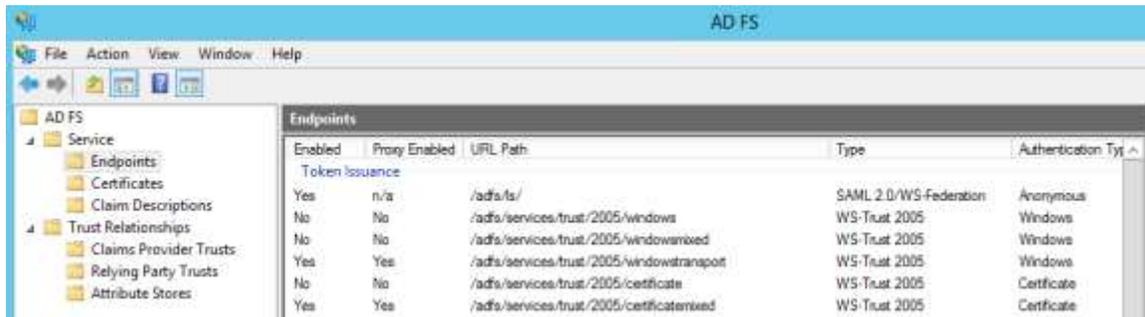
```
<useRelayStateForIdpInitiatedSignOn enabled="true" />
```

If not enabled, ADFS will convert IdP-initiated SSO into SP-initiated SSO.

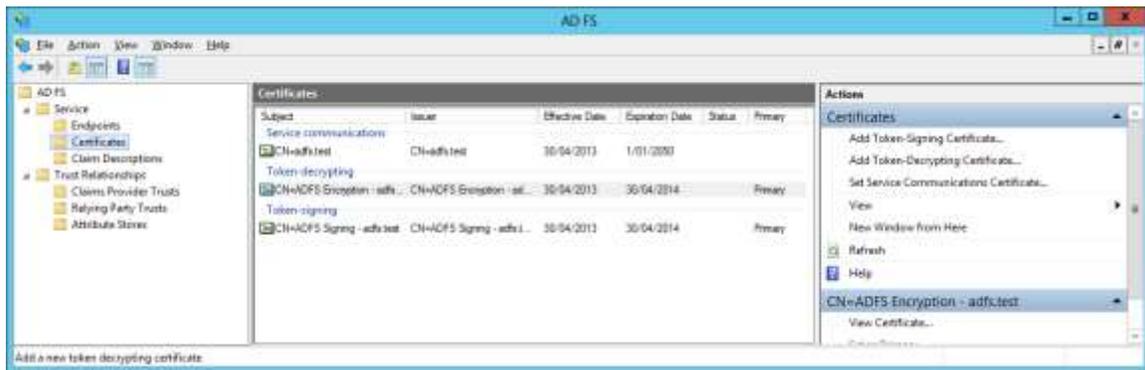
In the ADFS terminology, the identity provider is a claims provider. Using the ADFS management console, add a claims provider trust for the identity provider.

Note that strings in ADFS, including URLs, are case sensitive.

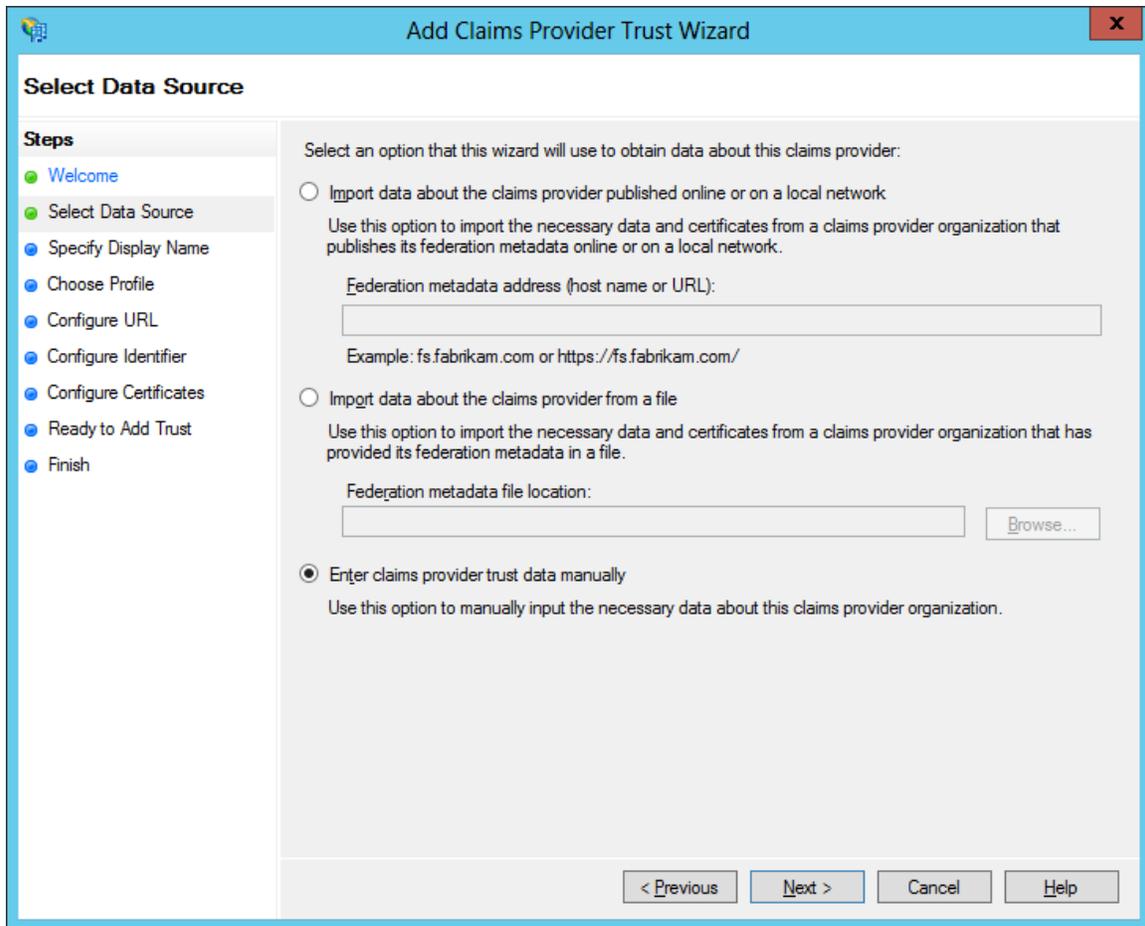
Confirm that the /adfs/ls endpoint for SAML v2.0 exists. If it doesn't, refer to the ADFS documentation.



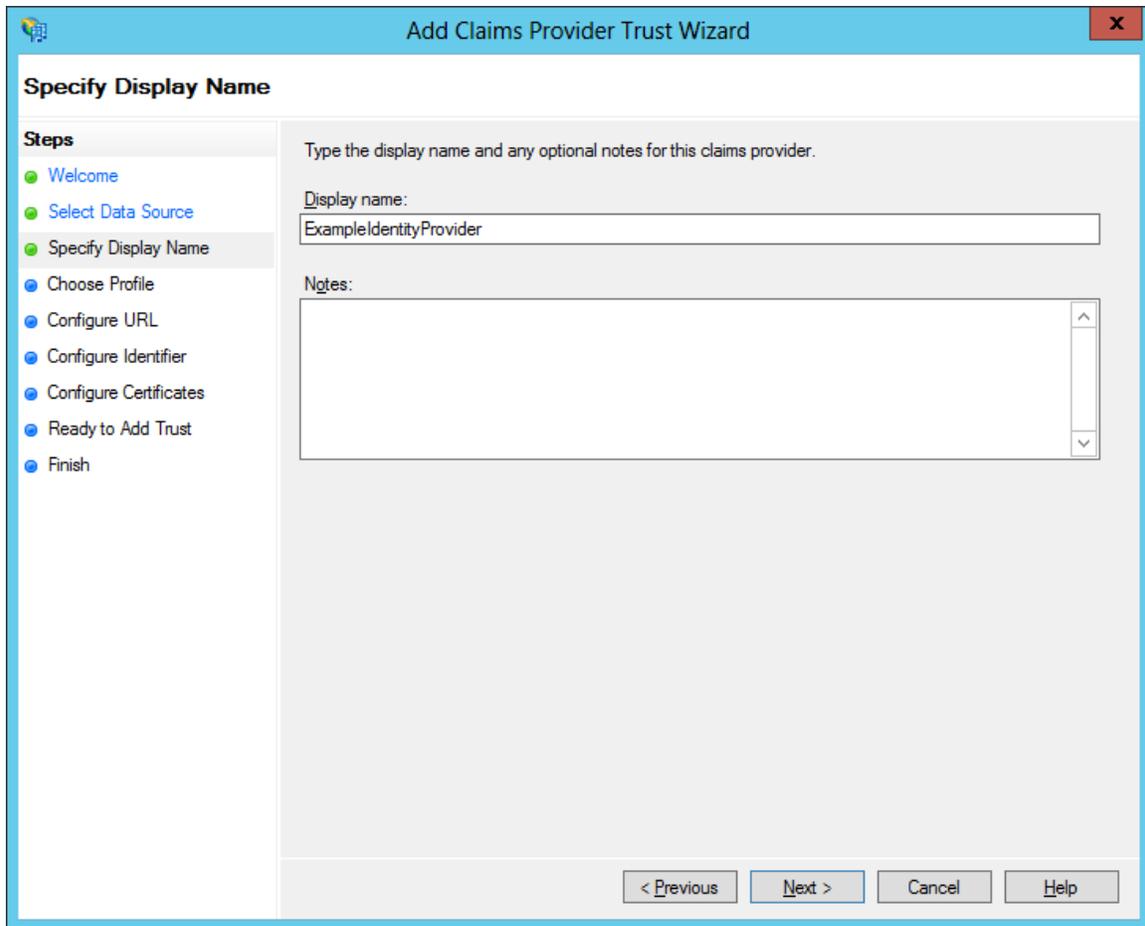
Confirm that the service communications, token decrypting and token encrypting certificates exist. If they don't, refer to the ADFS documentation.



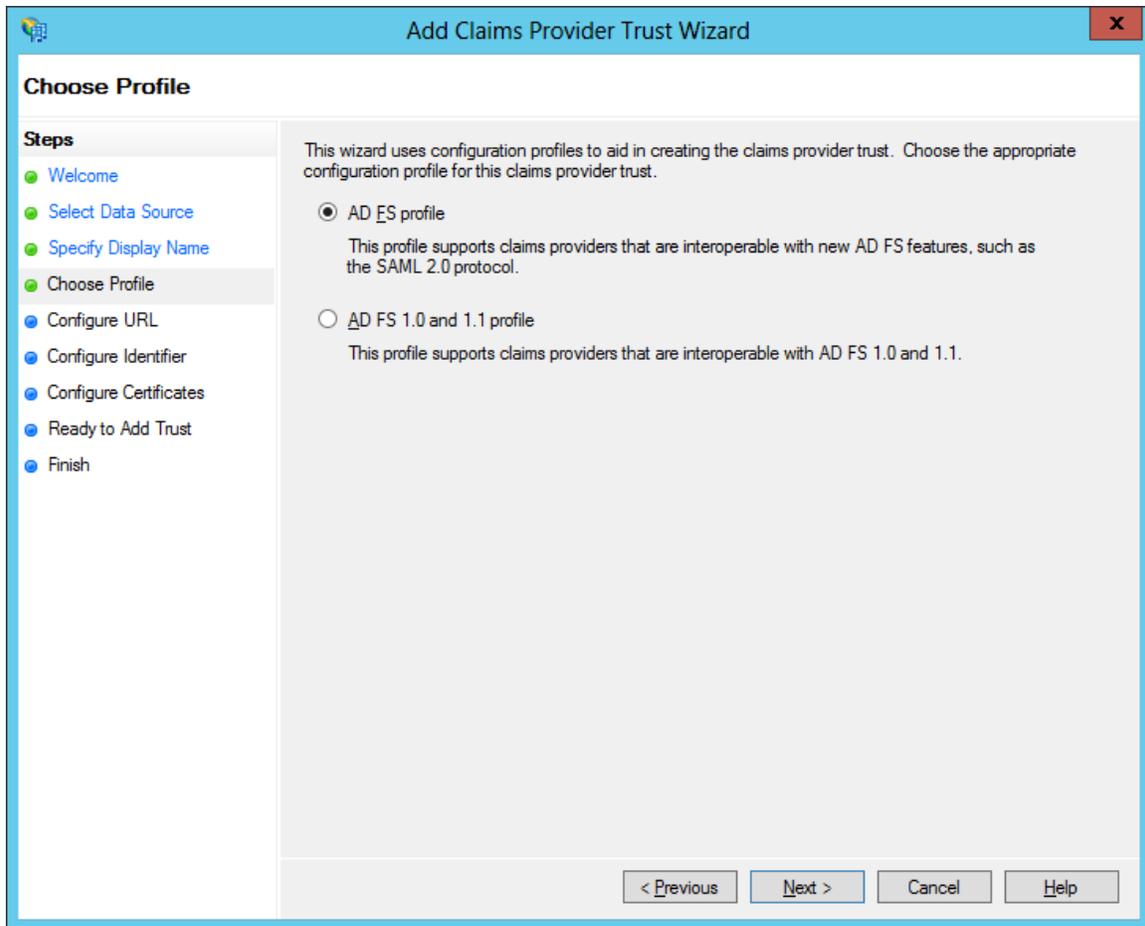
Add a claims provider trust and select the option to enter the claims provider information manually.



Specify a display name. The display name does not have to match with any other configuration.

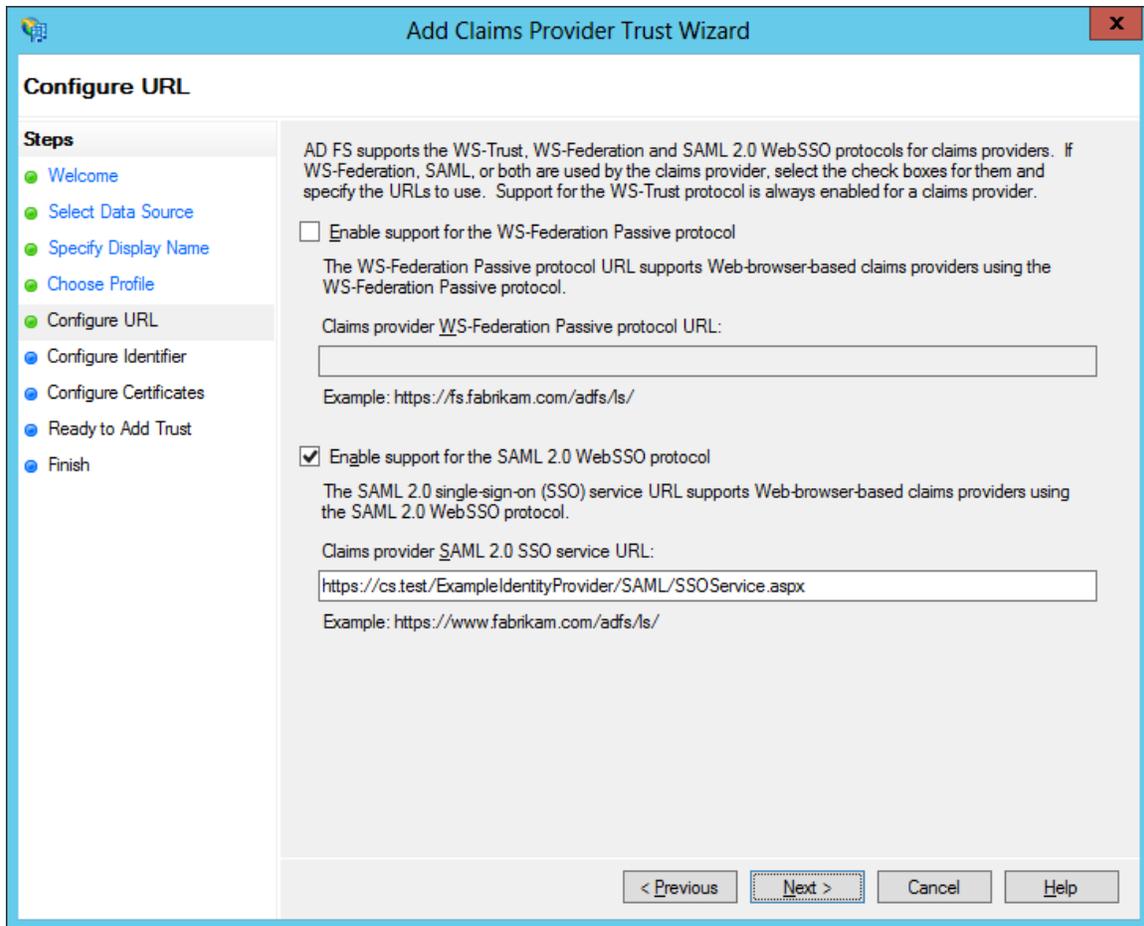


Choose the ADFS profile.



Enable support for SAML v2.0 and specify the identity provider's SSO service URL. ADFS sends the authn request to this URL. For example:

<https://cs.test/ExampleIdentityProvider/SAML/SSOService.aspx>



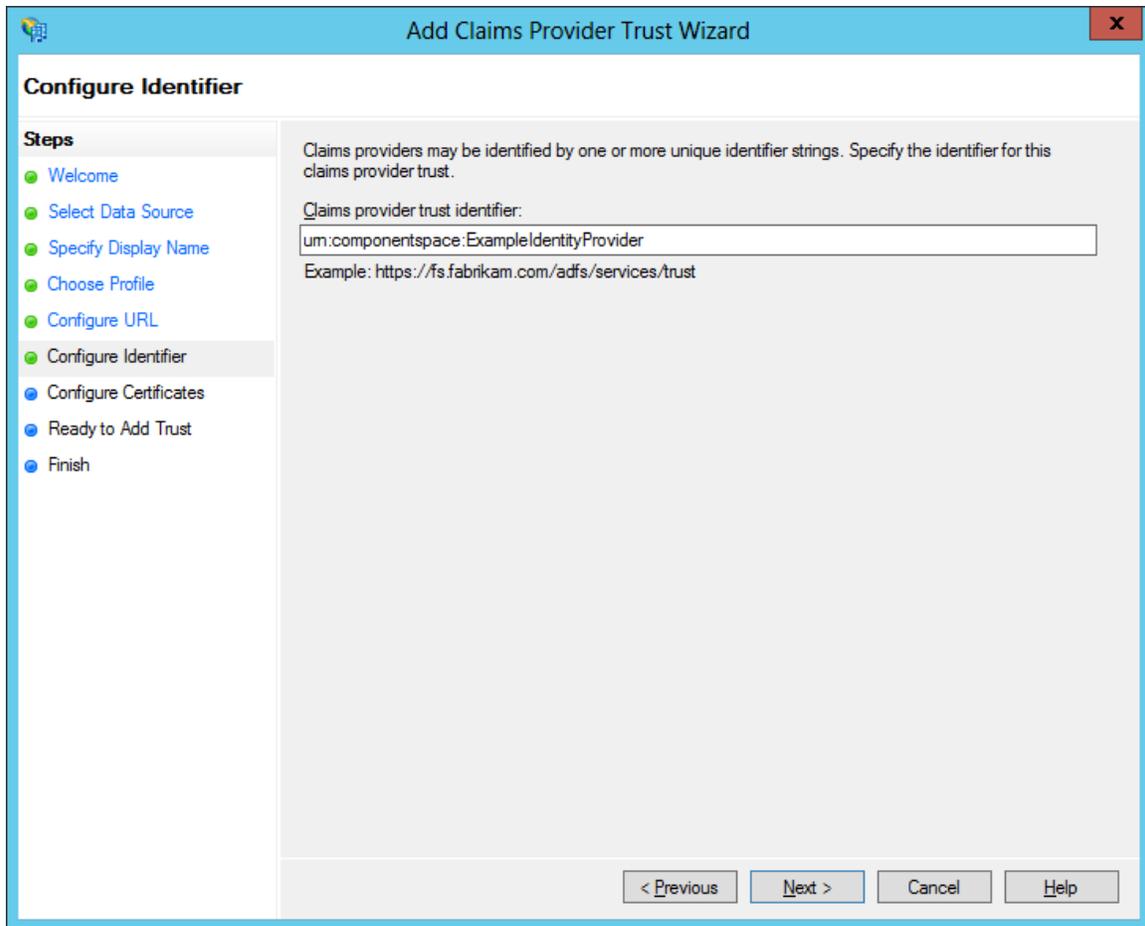
Specify the claims provider trust identifier. This identifier must match the issuer field in the authn request sent by the service provider. The IdentityProvider name attribute in the saml.config configuration file is used as the issuer and so this name and the claims provider trust identifier must match.

For example, if the saml.config includes:

```
<IdentityProvider Name="urn:componentspace:ExampleIdentityProvider"/>
```

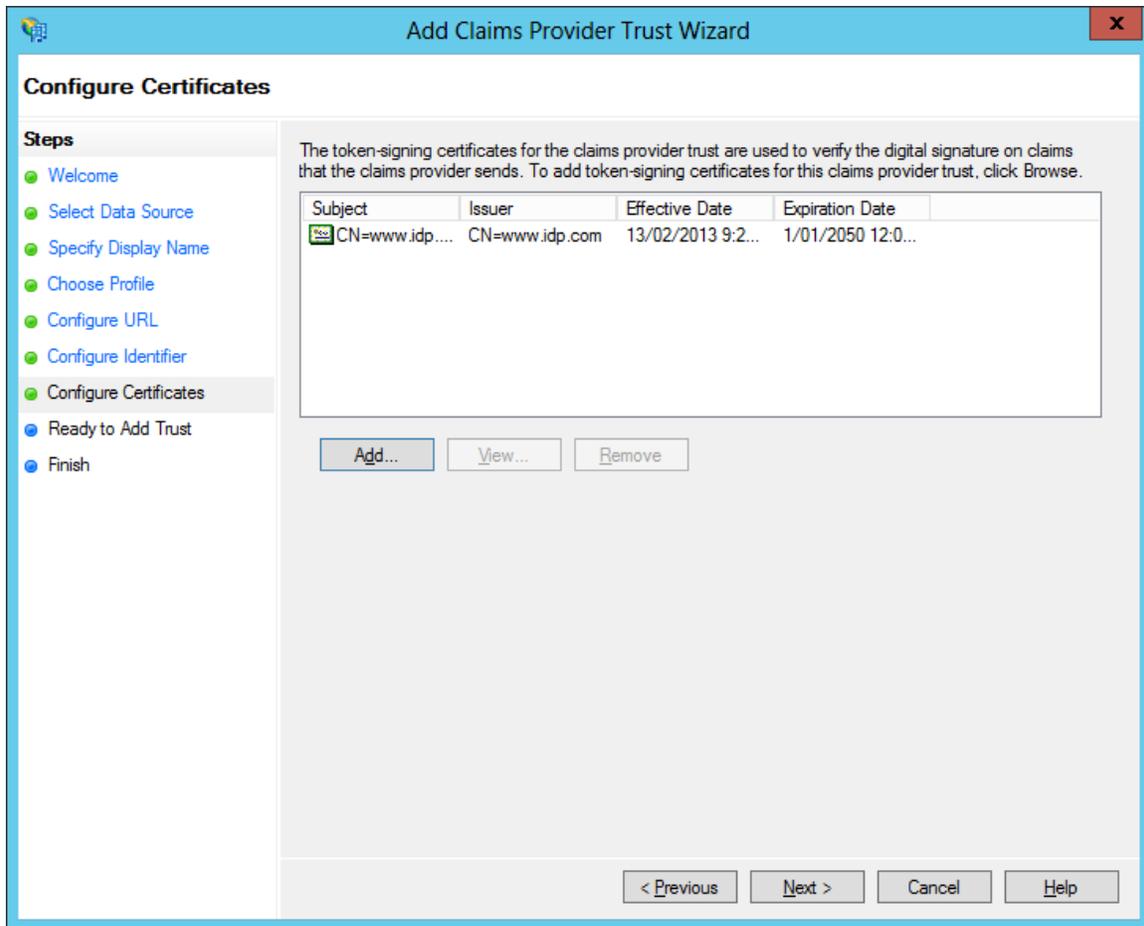
Then the claims provider trust identifier must be:

```
urn:componentspace:ExampleIdentityProvider.
```

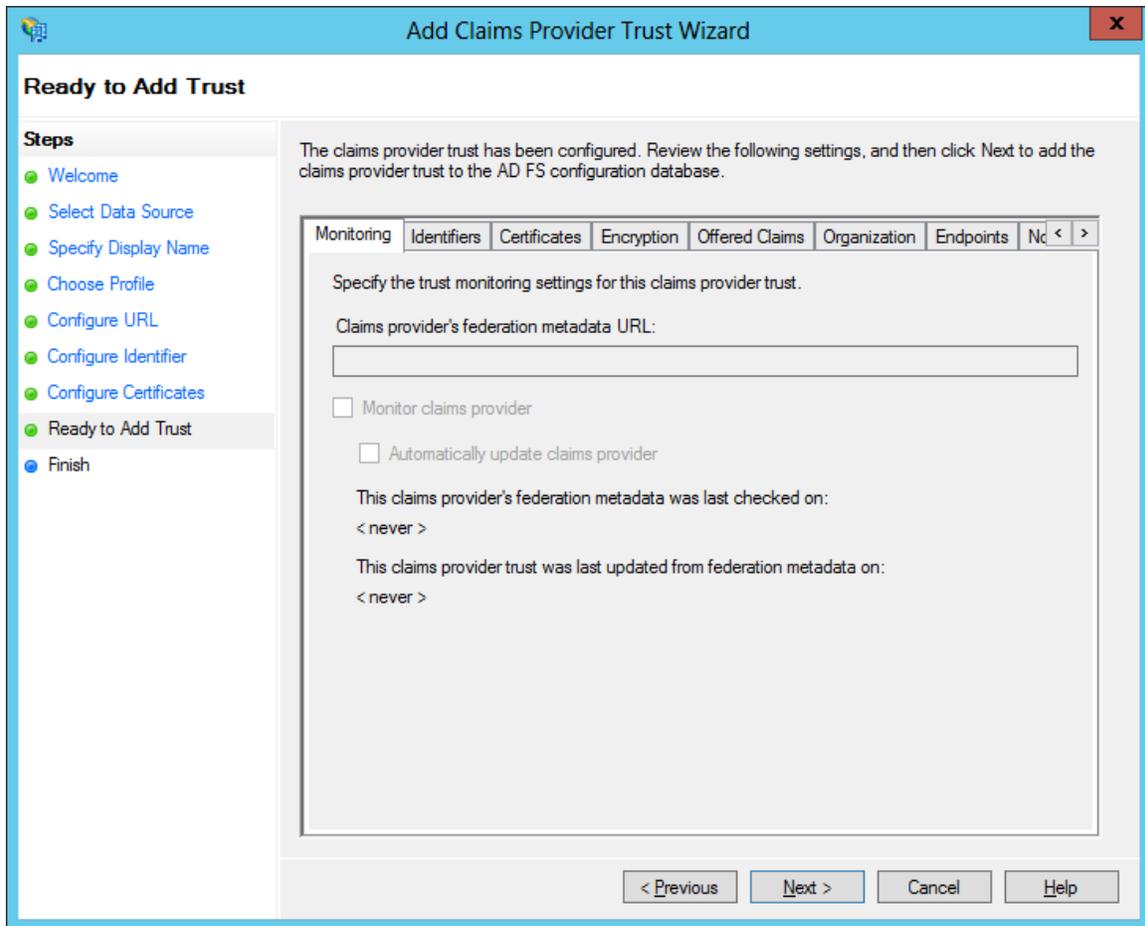


Browse to idp.cer to specify it as the token signing certificate. Ignore any warnings about the key length.

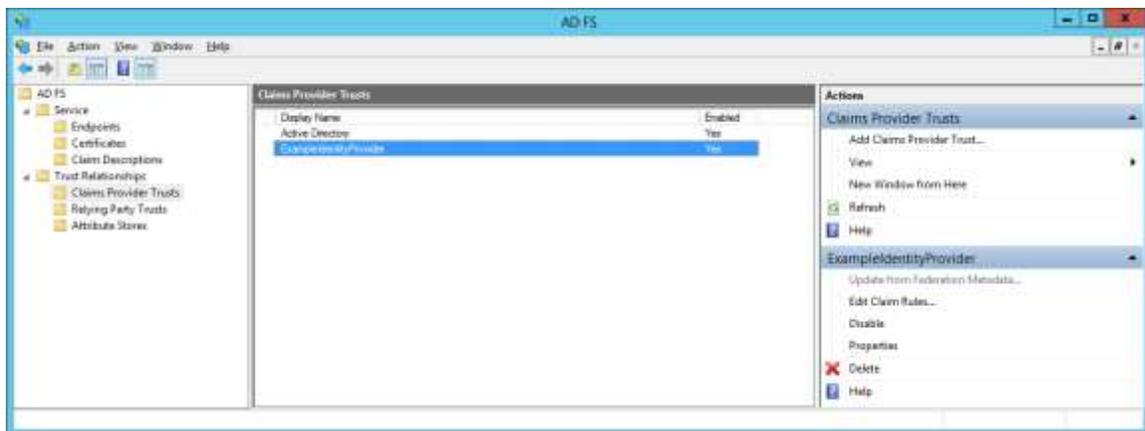
ADFS uses the token signing certificate to verify the SAML assertion signature.



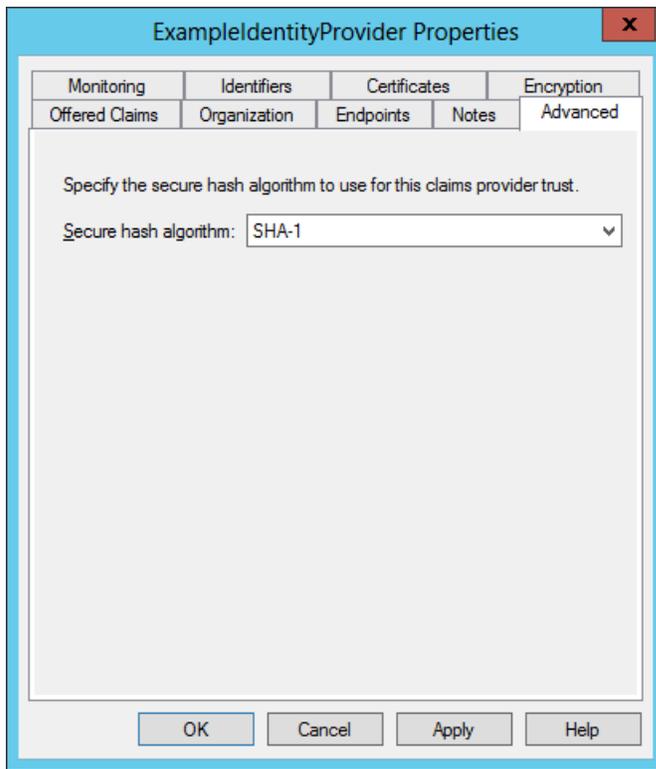
Review the configuration and close the wizard.



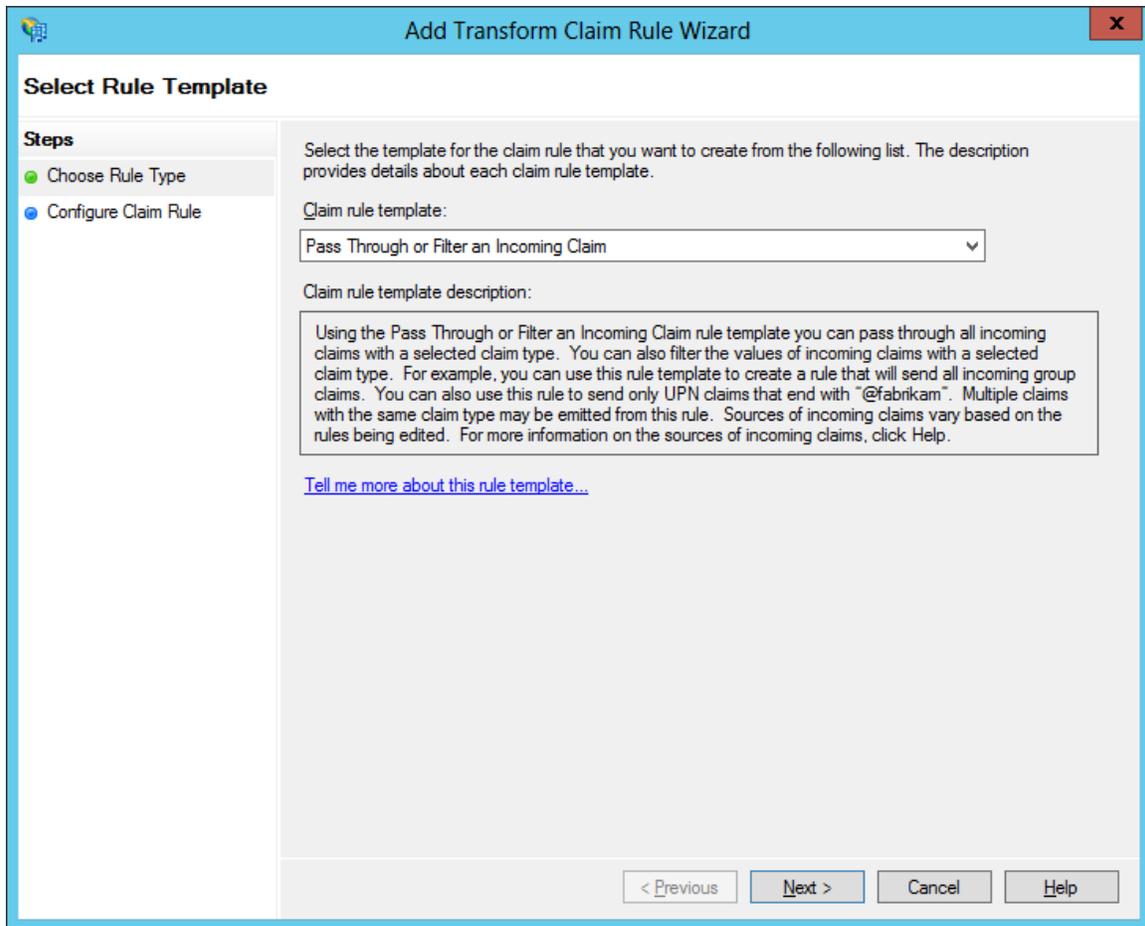
The identity provider should be included in the list of claims provider trusts.



Although the SAML v2.0 component supports SHA-256 signatures, for this example SHA-1 is used. To specify this, open the claims provider trusts' properties and, under the Advanced tab, select SHA-1.



Edit the claim rules and add a rule. Use the pass through template.



Add a rule to pass through the Name ID. Ignore any warning.

The screenshot shows the 'Add Transform Claim Rule Wizard' dialog box, specifically the 'Configure Rule' step. The dialog has a blue title bar with the text 'Add Transform Claim Rule Wizard' and a close button (X) in the top right corner. On the left side, there is a 'Steps' pane with two items: 'Choose Rule Type' (indicated by a green dot) and 'Configure Claim Rule' (indicated by a green dot and highlighted). The main area contains the following configuration options:

- Claim rule name:** A text box containing 'Name ID'.
- Rule template:** 'Pass Through or Filter an Incoming Claim'.
- Incoming claim type:** A dropdown menu set to 'Name ID'.
- Incoming name ID format:** A dropdown menu set to 'Unspecified'.
- Pass through all claim values:** A radio button that is selected.
- Pass through only a specific claim value:** An unselected radio button with a text box for 'Incoming claim value'.
- Pass through only claim values that match a specific email suffix value:** An unselected radio button with a text box for 'Email suffix value' and an example 'Example: fabrikam.com'.
- Pass through only claim values that start with a specific value:** An unselected radio button with a text box for 'Starts with' and an example 'Example: FABRIKAM\'\'.'

At the bottom of the dialog, there are four buttons: '< Previous', 'Finish', 'Cancel', and 'Help'.

### 10.4.8 Running the Identity Provider with IdP-Initiated SSO

In this example, the user logs in at the identity provider and initiates SSO to ADFS. ADFS forwards this to the specified service provider. The asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the service provider.

1. Browse to <https://cs.test/ExampleIdentityProvider>, ignoring any browser certificate warnings.



2. Click the link to single sign-on to the service provider.

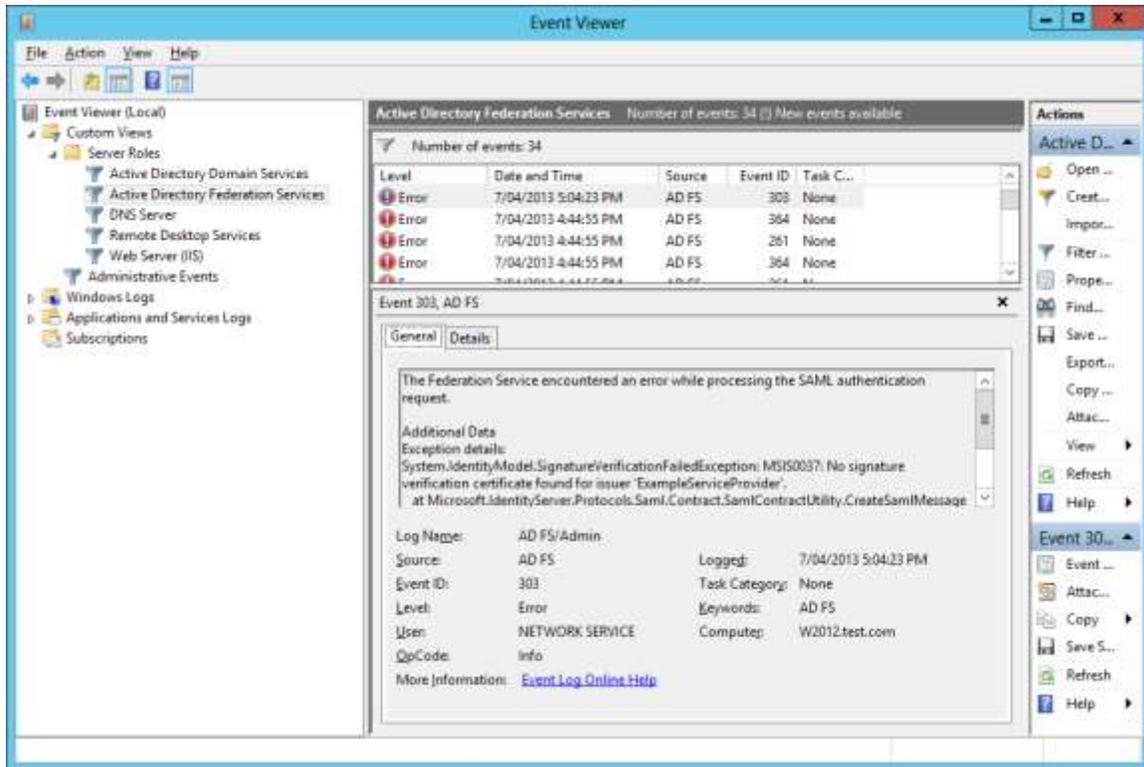


3. You should then be presented with the service provider's default page.



## 10.4.9 Troubleshooting ADFS SSO

Configuration errors will result in a cryptic message displayed in the browser by ADFS. To troubleshoot configuration and other problems, refer to the ADFS event log.



ADFS metadata may be viewed at the FederationMetadata/2007-06/FederationMetadata.xml endpoint. For example:

<https://adfs.test/FederationMetadata/2007-06/FederationMetadata.xml>

## 10.5 Office 365 Interoperability

The Web Forms and MVC example identity providers demonstrate single sign-on with Office 365.

The following sections describe the configuration for the Web Forms example identity provider but, with the appropriate changes, apply equally to the MVC examples.

Refer to sections 10.1 and 10.2 for instructions on installing and configuring the Web Forms and MVC example identity providers.

### 10.5.1 Configuring the Identity Provider

The saml.config file includes the following entry for the Office 365 partner service provider.

```
<PartnerServiceProvider Name="urn:federation:MicrosoftOnline"
    WantAuthnRequestSigned="false"
    SignSAMLResponse="false"
    SignAssertion="true"
    EncryptAssertion="false"
    NameIDFormat=
    "urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
    AuthnContext=
    "urn:oasis:names:tc:SAML:2.0:ac:classes:Password"
    AssertionConsumerServiceUrl=
    "https://login.microsoftonline.com/login.srf"
    SingleLogoutServiceUrl=
    "https://login.microsoftonline.com/login.srf"/>
```

The web.config's *PartnerSP* setting specifies the partner service provider for IdP-initiated SSO and should be set to urn:federation:MicrosoftOnline.

```
<add key="PartnerSP" value="urn:federation:MicrosoftOnline"/>
```

The web.config's *SubjectName* setting specifies the subject's name identifier. This must match with the user's immutable identifier configured in Office 365. In this example, a fixed immutable identifier is used. Refer to section 10.5.3.1.

```
<add key="SubjectName" value="12345678"/>
```

The web.config specifies the SAML attributes. The IDPEmail attribute must match with the user's principal name configured in Office 365. In this example, a fixed principal name is used.

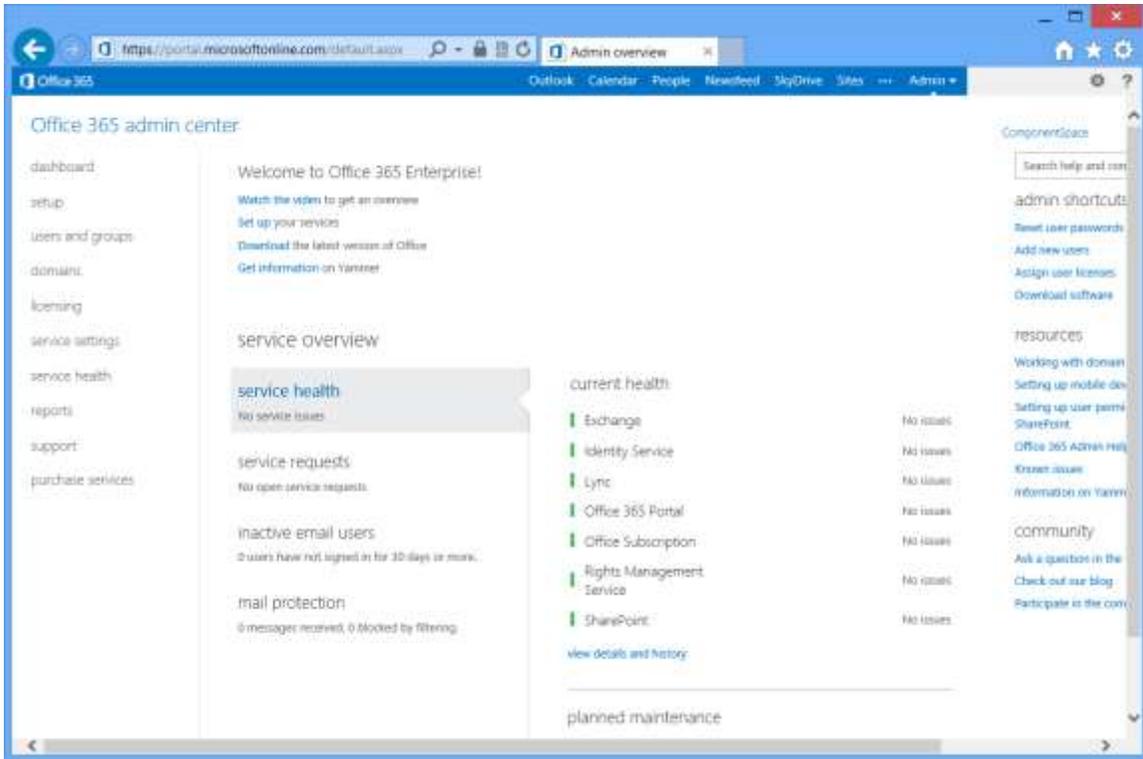
```
<add key="Attribute_IDPEmail" value="test@componentspace.com"/>
```

To keep the example identity provider simple, fixed values are used for the user's immutable identifier and principal name. In a production application, these values would be retrieved from the user store (e.g. Active Directory or a user database).

## 10.5.2 Configuring Office 365

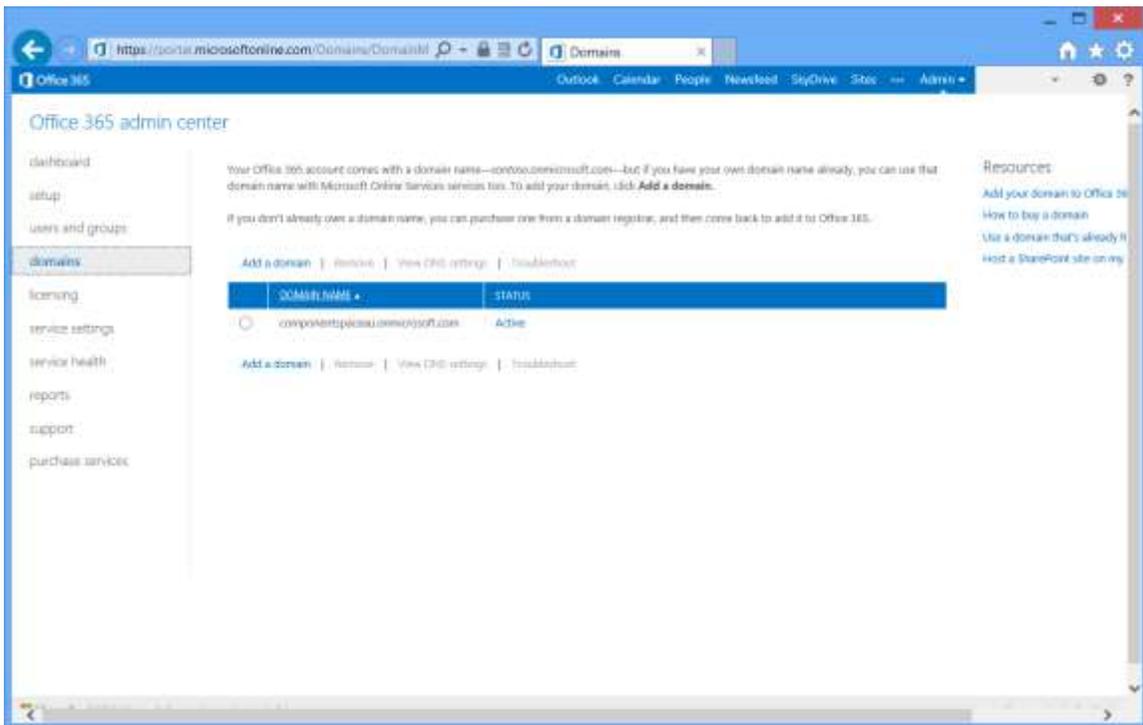
Login, as an administrator, to the Office 365 administration center at:

<https://portal.microsoftonline.com/>

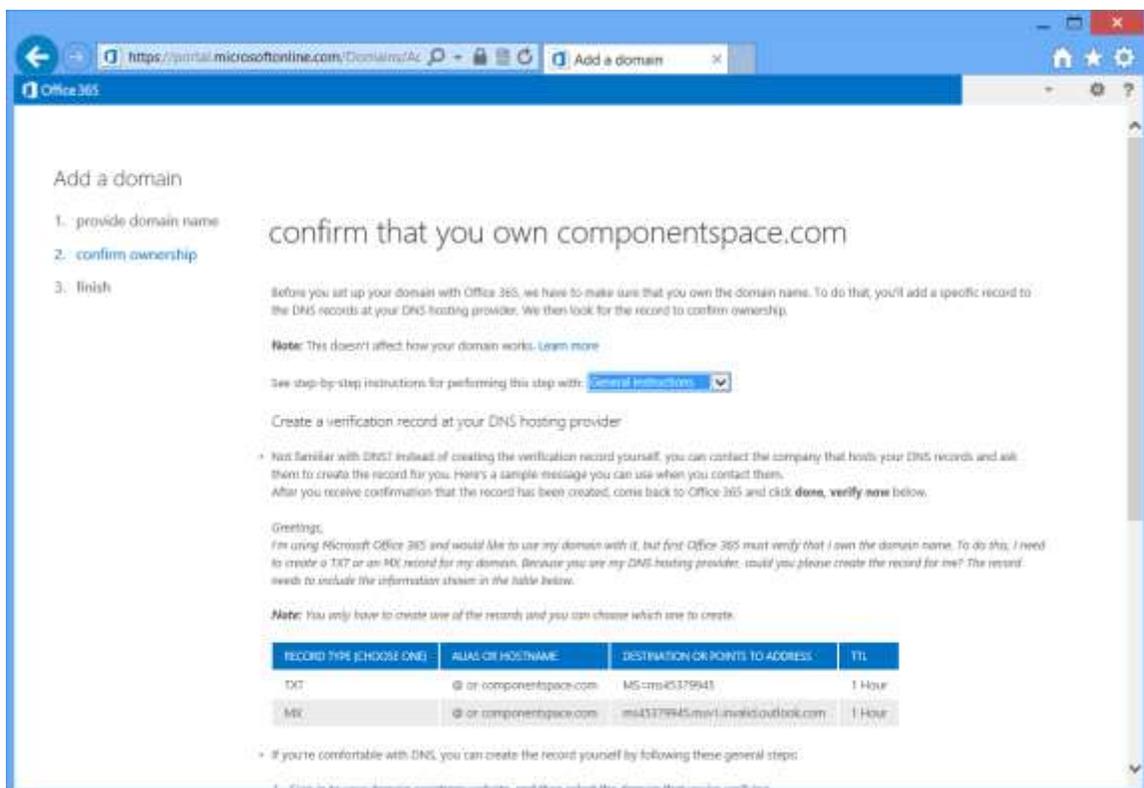
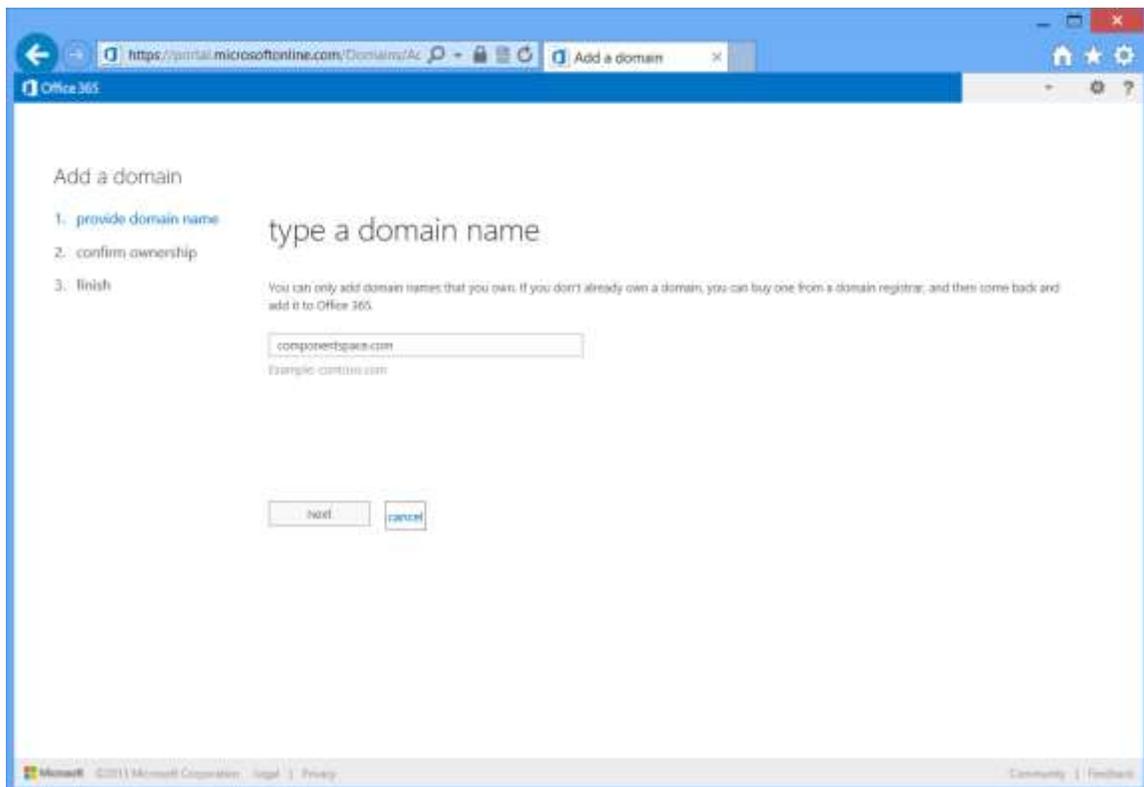


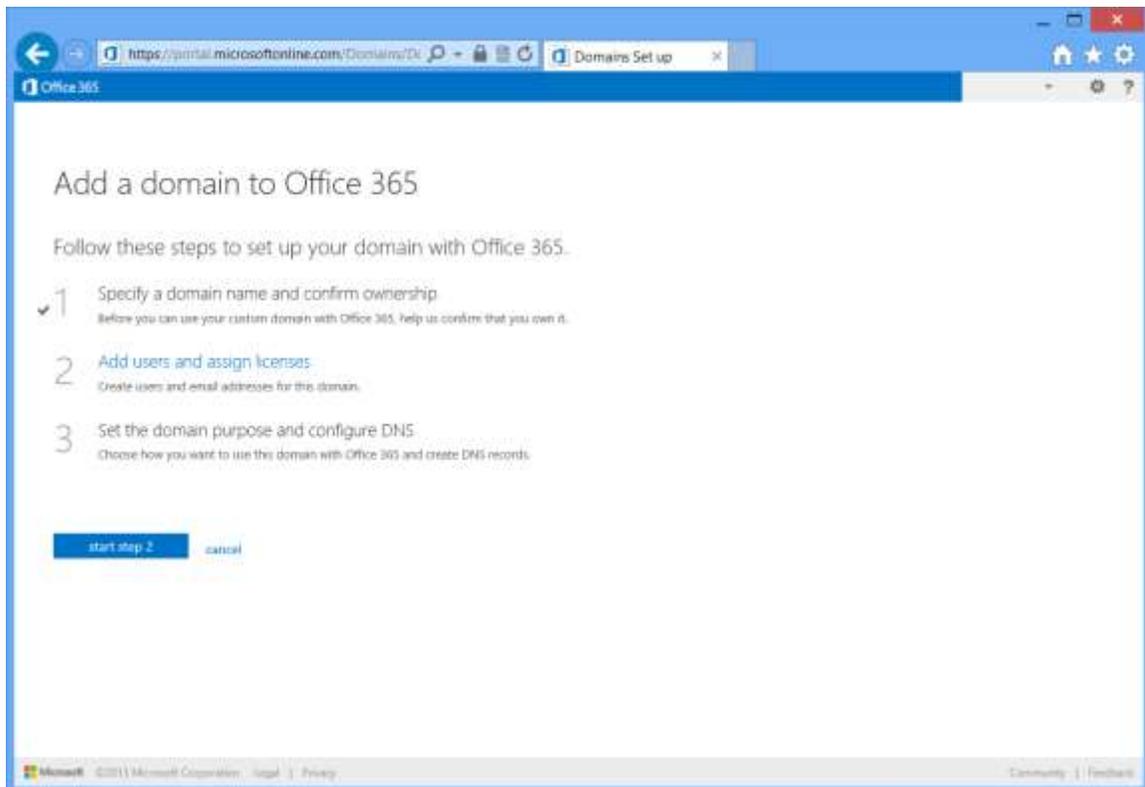
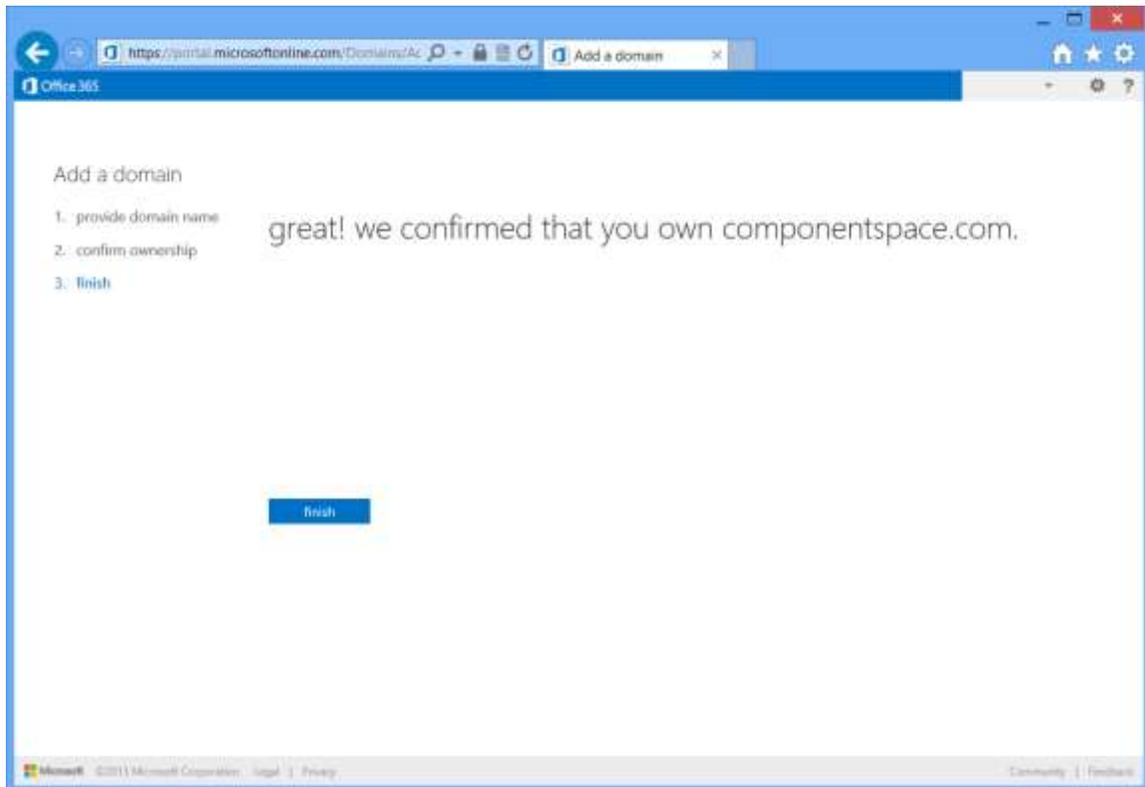
### 10.5.2.1 Add a Domain

Add a domain that will be used for single sign on.

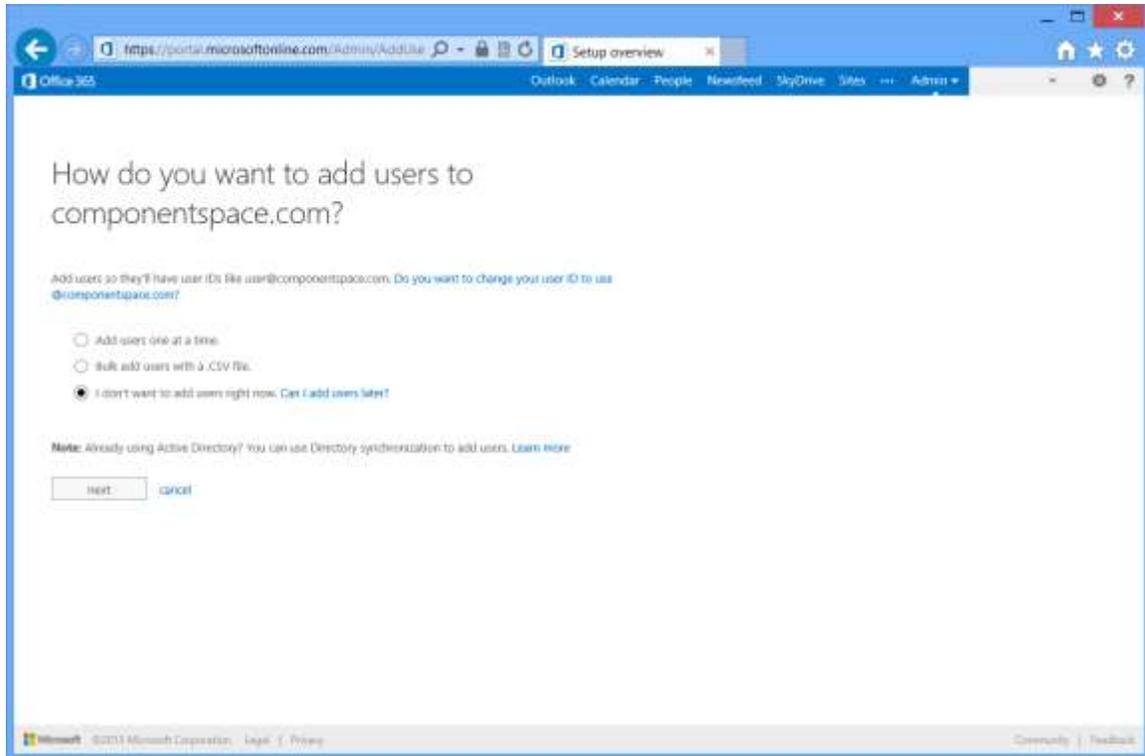


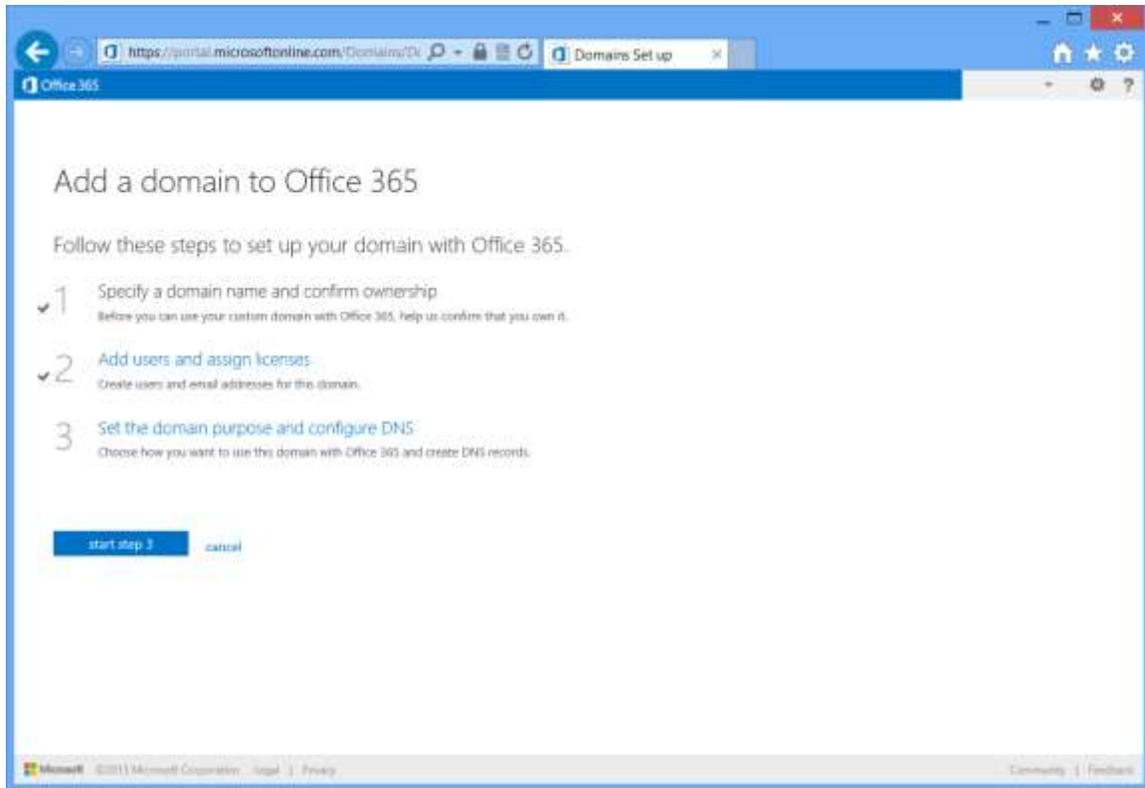
# ComponentSpace SAML v2.0 for .NET Developer Guide





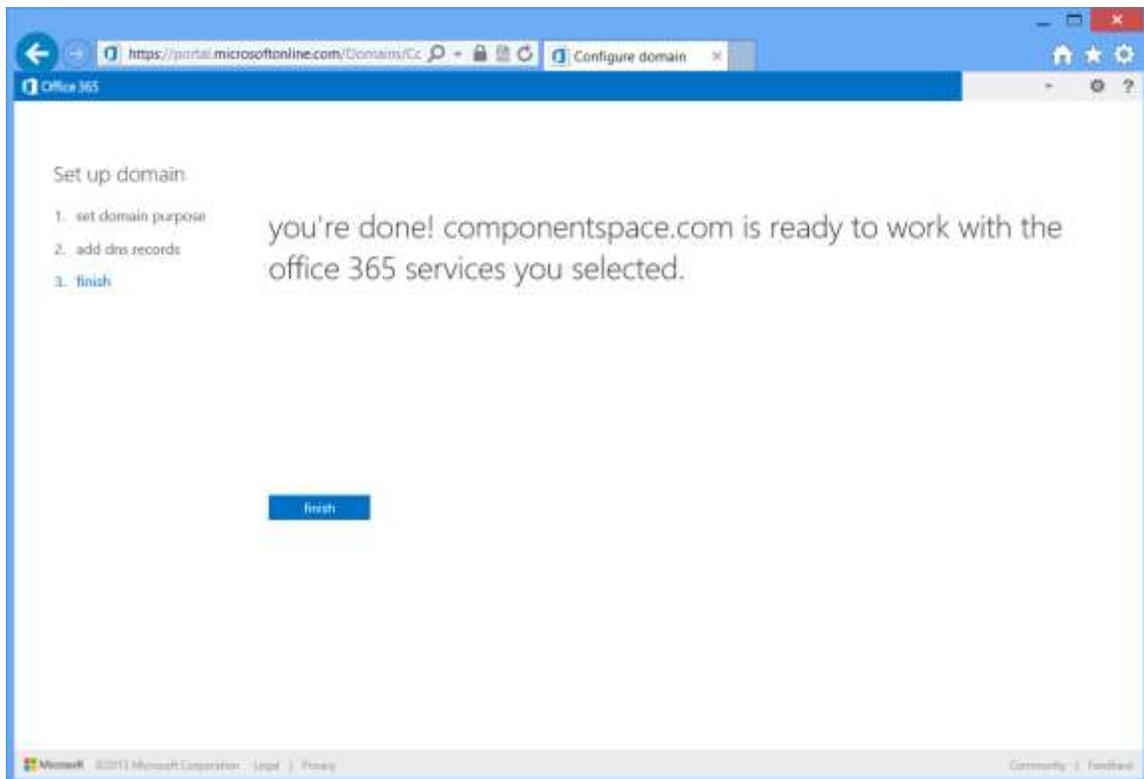
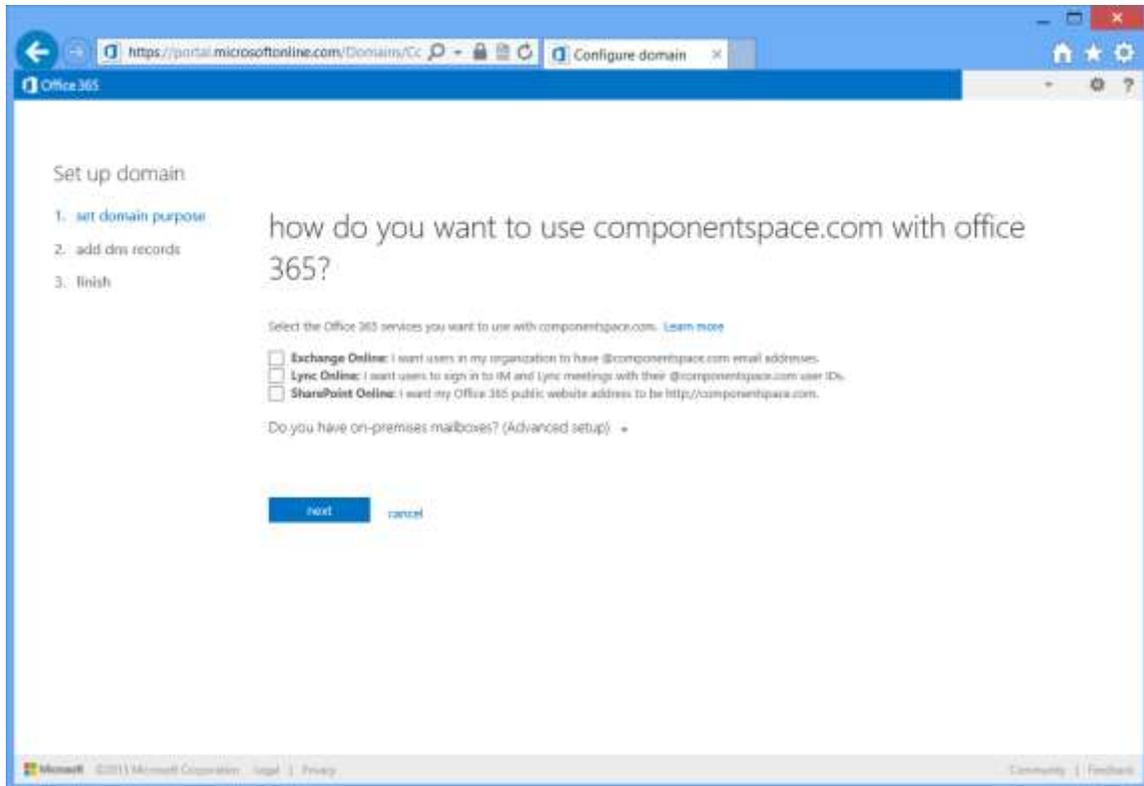
Don't add any users at this stage.

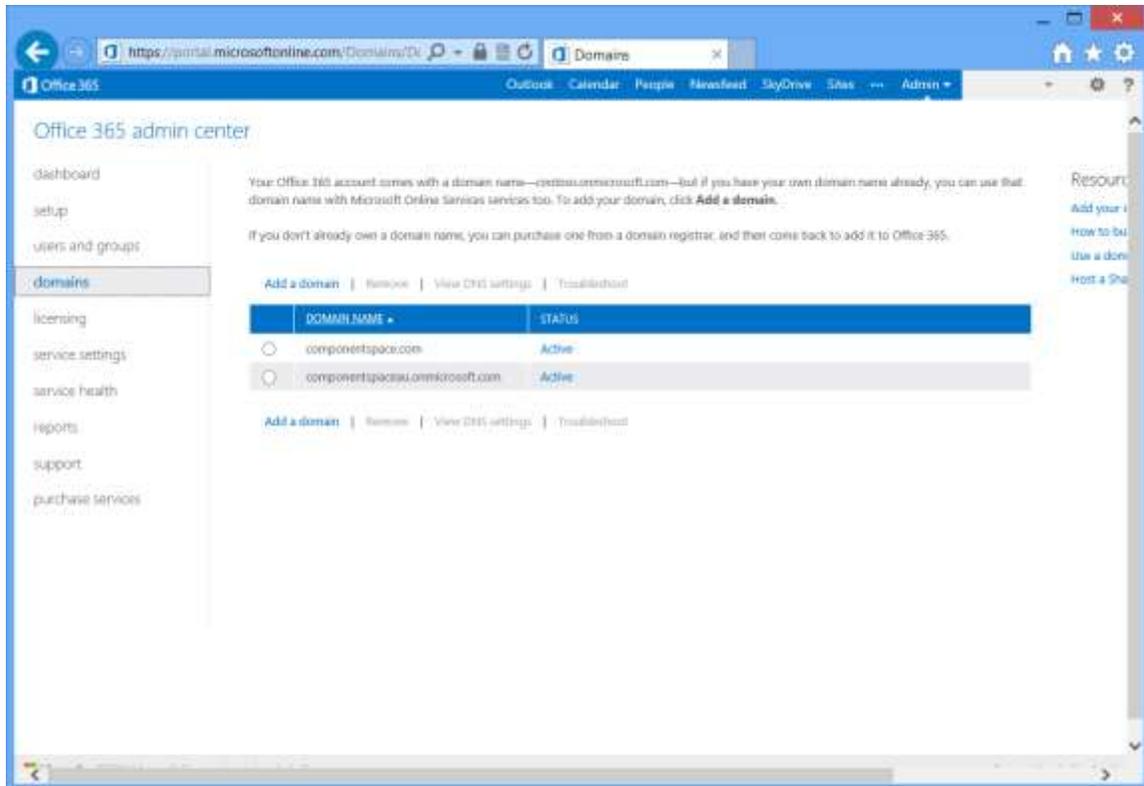




Unless DNS entries are to be updated, uncheck the Exchange Online and Lync Online check boxes.

## ComponentSpace SAML v2.0 for .NET Developer Guide





### 10.5.2.2 Set the Default Domain

Ensure the newly added domain is not the default domain.

Click the organization's name in the top right corner.

Change the default domain to the onmicrosoft.com domain (e.g. componentspaceau.onmicrosoft.com instead of componentspace.com).

### 10.5.2.3 Install the Azure PowerShell Cmdlets

Single sign on cannot be configured using the Office 365 administration center. Instead, the Windows Azure Active Directory Module for Windows PowerShell cmdlets are used to configure Office 365 for single sign on.

Download and install the cmdlets from:

<http://technet.microsoft.com/en-us/library/jj151815.aspx>

More information about these cmdlets and using them to configure single sign on may be found at:

<http://technet.microsoft.com/en-us/library/jj151815.aspx>

and

<http://technet.microsoft.com/en-us/library/hh967628.aspx>

### 10.5.2.4 Configuring the Domain for Single Sign On

Run the Set-MsolDomainAuthentication cmdlet to configure single sign on.

The following PowerShell script configures the componentsspace.com domain.

For convenience, it's recommended this is included in a PowerShell .ps1 script file.

```
# Configure Office 365 SSO

# Prompt for the administrator's credentials
$cred=Get-Credential
Connect-MsolService -Credential $cred

$domain = "componentsspace.com"
$issuer = "urn:componentsspace:ExampleIdentityProvider"
$sssoUrl =
"https://test.componentsspace.com/ExampleIdentityProvider/SAML/SSOService.aspx"
$ecpUrl =
"https://test.componentsspace.com/ExampleIdentityProvider/SAML/ECP.aspx"
$logoffUrl =
"https://test.componentsspace.com/ExampleIdentityProvider/SAML/SLOService.aspx"
$cert =
"MIIBrzCCARigAwIBAgIQWJaxa3/MnJlO88oamyZTuzANBgkqhkiG9w0BAQUFADAWMRQwEg
YDVQQDEwt3d3cuaWRwLmNvbTAeFw0xMzAyMTIyMzIyNDRaFw00OTEyMzExNDAwMDBaMBYx
F DASBgNVBAMTC3d3dy5pZHAuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDEru
hL B9LyKJsuuXTl39vEYMYUg+/o+SoOLfuRH7g/o/FJV6QT0gFDL70uN/YDdDnC8zDza8
WZbDl eA0W7ot76Uq71vFwf69fV4VnhxsQOmDAGlmQDeZtbWWbJEmnX9oAwkqQEOd8sBZ
nyYrbwPy zgnWM3HLOu2vVvNtXUmWJ6owIDAQABMA0GCSqGSIb3DQEBAQUAA4GBAKN1
MwB8ug6TLiqDw XcYqfbsFDfPoI04pH2Pzu19NBs6v9P0G+SF2tRlZ4NVQ/ADQkUPu
WM0GKiluJws898R+RF6 znCvW93rl14FdQli660JO2PnDlSViBfc07hj0atOYolweFFt
BLinAYIZL6P/S1IcHTYpo3Mg oQGVInCMMYTUW"

Set-MsolDomainAuthentication -FederationBrandName $domain -DomainName
$domain -Authentication federated -PreferredAuthenticationProtocol
SAML -IssuerUri $issuer -SigningCertificate $cert -PassiveLogOnUri
$sssoUrl -ActiveLogOnUri $ecpUrl -LogOffUri $logoffUrl -Verbose
```

The following is a template for configuring a domain.

```
# Configure Office 365 SSO

# Prompt for the administrator's credentials
$cred=Get-Credential
Connect-MsolService -Credential $cred

$domain = "TODO: specify domain name"
$issuer = "TODO: specify issuer"
$sssoUrl = "TODO: specify SSO service URL"
$ecpUrl = "TODO: specify ECP service URL"
$logoffUrl = "TODO: specify the SLO service URL"
$cert = "TODO: specify the identity provider's certificate"
```

## ComponentSpace SAML v2.0 for .NET Developer Guide

```
Set-MsolDomainAuthentication -FederationBrandName $domain -DomainName
$domain -Authentication federated -PreferredAuthenticationProtocol
SAML -IssuerUri $issuer -SigningCertificate $cert -PassiveLogOnUri
$sssoUrl -ActiveLogOnUri $secpUrl -LogOffUri $logoffUrl -Verbose
```

The `$domain` is the domain name previously configured in Office 365 for single sign on.

The `$issuer` is the identity provider name. This name must match with the `IdentityProvider` name configured in the identity provider's `saml.config`. For example, if the identity provider name is `urn:componentspace:ExampleIdentityProvider` then the `$issuer` must be set to the same value.

The `$sssoUrl` is the identity provider's SSO service URL. In browser-based SP-initiated SSO, Office 365 will send an authentication request to this endpoint.

The `$secpUrl` is the identity provider's Enhanced Client or Proxy URL. In non-browser-based SP-initiated SSO, Office 365 will send an authentication request to this endpoint.

The `$logoffUrl` is the identity provider's SLO service URL.

The `$cert` is the identity provider's certificate. Office 365 will use this certificate to verify signed SAML assertions from the identity provider.

The `Set-MsolDomainAuthentication` cmdlet configures authentication for the domain.

The `"-Authentication federated"` parameter specifies to use single sign on. The `"-PreferredAuthenticationProtocol SAML"` parameter specifies to use the SAML protocol rather than WS-Federation.

### 10.5.2.5 Confirming the Domain's SSO Settings

Run the `Get-MsolDomainFederationSettings` cmdlet to confirm the single sign on settings. For example:

```
$domain = "componentspace.com"
Get-MsolDomainFederationSettings -DomainName $domain
```

### 10.5.3 Adding a User

Run the `New-MsolUser` cmdlet to add a user to the domain. For example:

```
New-MsolUser -UserPrincipalName test@componentspace.com -ImmutableId
12345678 -FirstName Test -LastName User -DisplayName "Test User" -
LicenseAssignment "componentspaceau:ENTERPRISEPACK" -usageLocation US
```

The `UserPrincipalName` is the user ID.

The `ImmutableId` is a unique ID that identifies the user. See section 10.5.3.1.

The `LicenseAssignment` assigns licenses to the user. Use the `Get-MsolAccountSku` cmdlet to get the value for the license assignment.

### 10.5.3.1 User Immutable Identifier

The immutable identifier uniquely and permanently identifies the user.

The SAML response sent by the identity provider includes the immutable identifier as the subject name identifier in the SAML assertion. The user principal name is included as the IDPEmail SAML attribute. Both these values must match with the Office 365 configuration for single sign on to be successful.

For user information stored in Active Directory, the user's object GUID (objectGUID attribute) may be used as the immutable identifier.

For user information stored in a database or some other user registry, some other unique identifier must be assigned as the immutable identifier.

In the example identity provider, a fixed immutable identifier is used.

### 10.5.4 Deleting a User

During testing, it may be necessary to delete and reconfigure users in Office 365.

Users may be deleted using the Office 365 administration center or by using the PowerShell Remove-MsolUser cmdlet. For example:

```
Remove-MsolUser -UserPrincipalName test@componentspace.com
```

Deleting the user moves the user to the Office 365 recycle bin. To create a user with the same name, the user first must be removed from the recycle bin. This requires the object identifier associated with the user.

The Get-MsolUser cmdlet is used to retrieve the object identifier. For example:

```
Get-MsolUser -ReturnDeletedUsers -SearchString test@componentspace.com  
| select UserPrincipalName, ObjectId
```

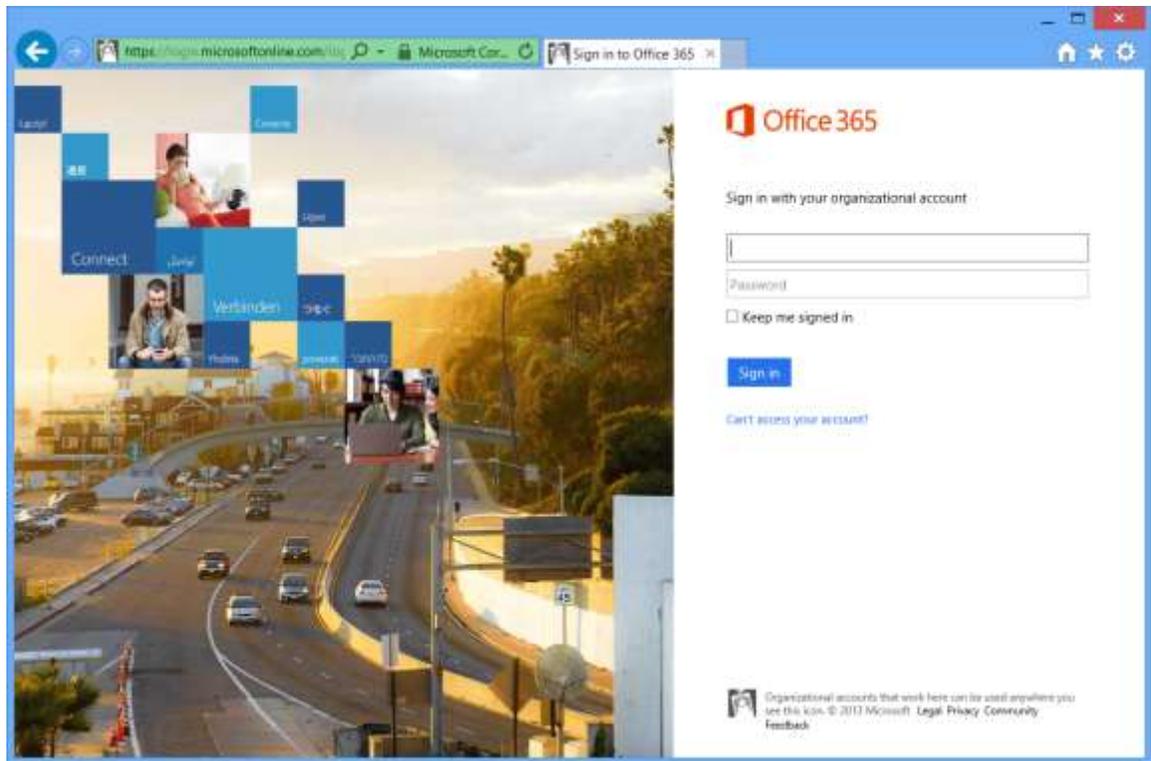
The Remove-MsolUser cmdlet is used to delete the user from the recycle bin. For example:

```
Remove-MsolUser -RemoveFromRecycleBin -ObjectId [objectID value]
```

### 10.5.5 Running the Identity Provider with SP-Initiated SSO

In this example, the user is attempting to login at Office 365 and, rather than performing a local login at Office 365, SSO is initiated with a local login occurring at the example identity provider and the asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the Office 365.

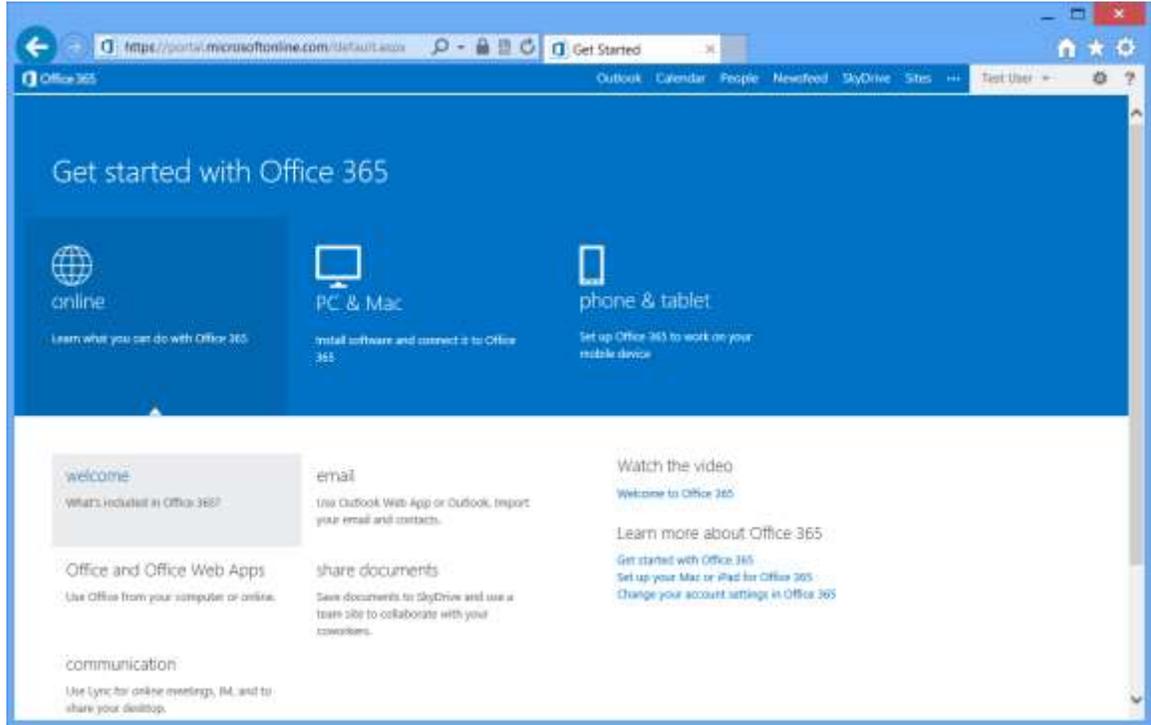
1. Browse to <https://portal.microsoftonline.com/>.



2. Specify the user e.g. [test@componentspace.com](mailto:test@componentspace.com). Although a prompt for a password is initially displayed, Office 365 determines the domain is federated and automatically redirects to the identity provider for login.



4. You should now be logged in at Office 365.



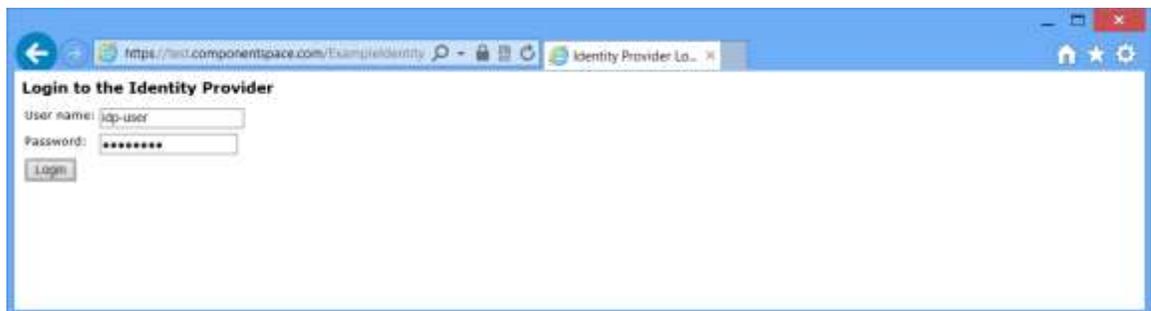
### 10.5.6 Running the Identity Provider with IdP-Initiated SSO

In this example, the user is logged in at the identity provider and clicks a link to SSO to Office 365.

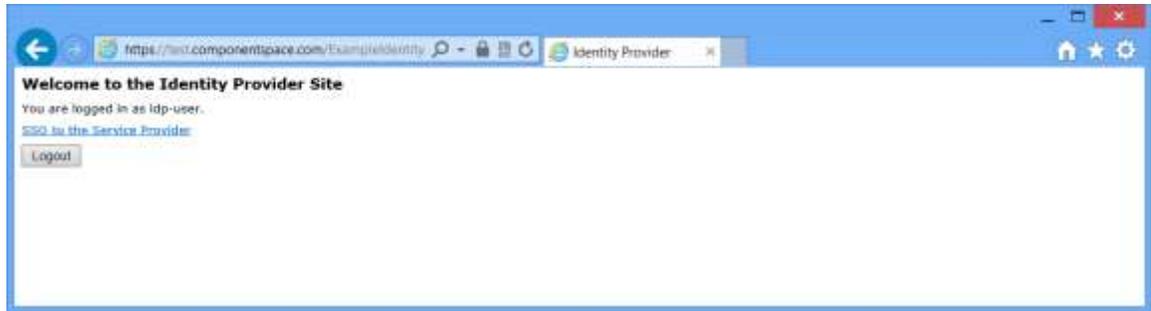
1. Browse to the example identity provider.

For example: <https://test.componentspace.com/ExampleIdentityProvider>.

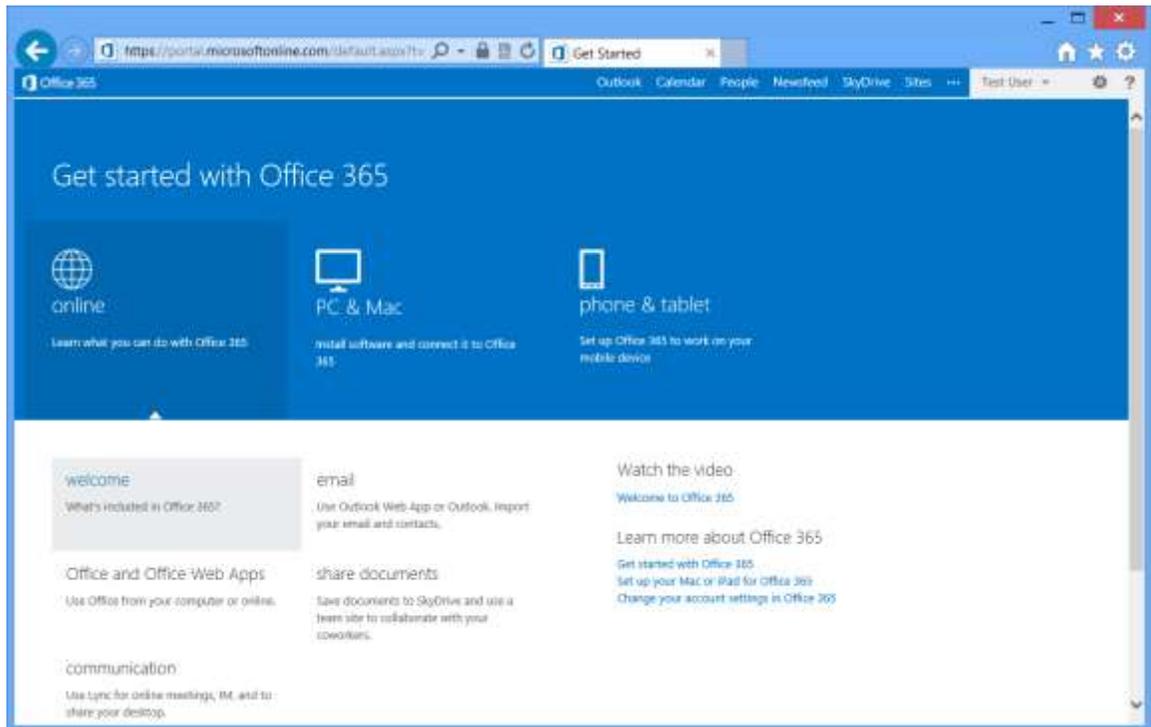
2. Login at the example identity provider.



3. Click the link to SSO to Office 365.



4. You should now be logged in at Office 365.



### 10.5.7 Email Client Support

Office 365 supports users of email clients, such as Microsoft Outlook, logging in through an identity provider.

The user's name and password are sent by the email client to Office 365 which delegates user authentication to the identity provider.

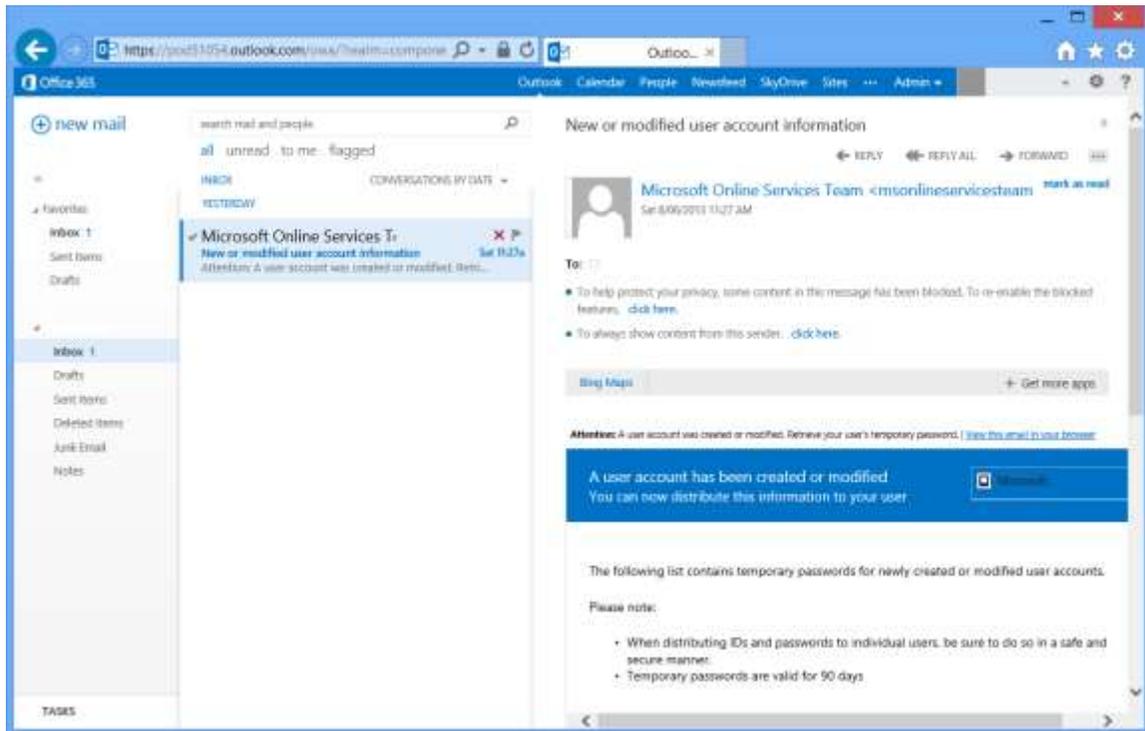
The SAML Enhanced Client or Proxy (ECP) profile is used for the exchange of SAML messages between Office 365 and the identity provider.

Along with the SAML authentication request sent to the identity provider, Office 365 includes the user's name and password in the HTTP authorization header.

The identity provider uses these credentials to authenticate the user. If authenticated, the identity provider returns a SAML response to Office 365.

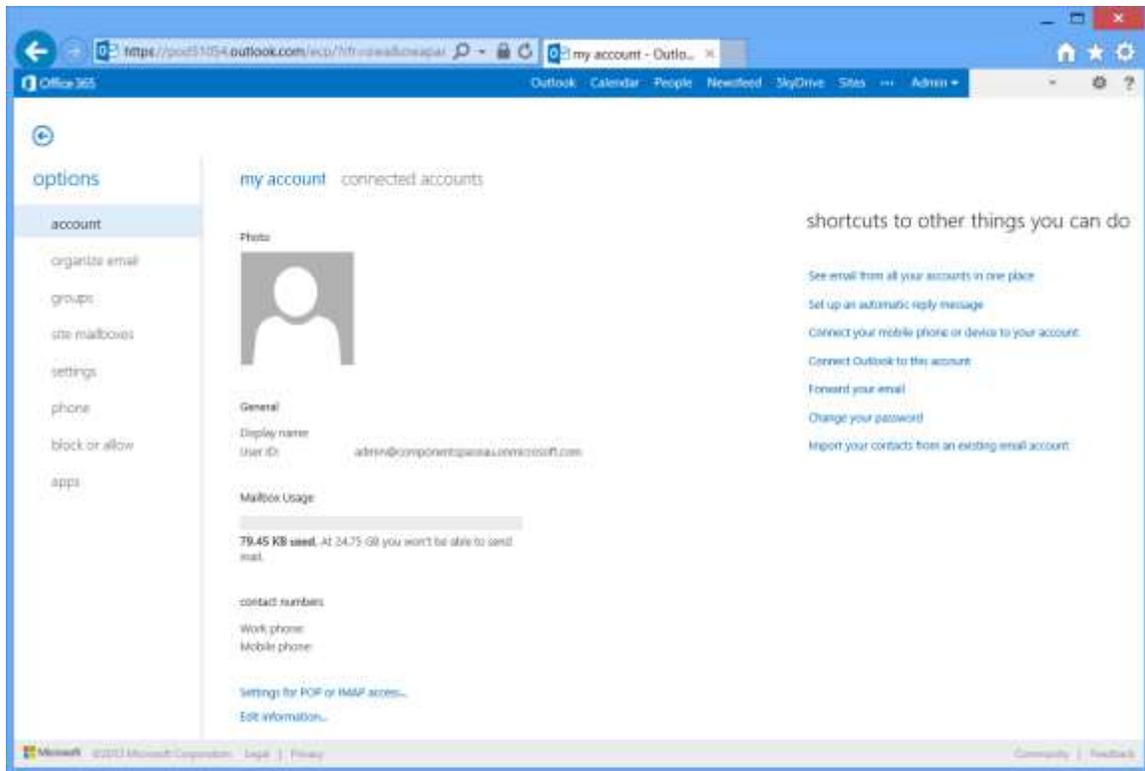
## 10.5.8 Configuring an Email Client

Login as the Office 365 administrator and click the Outlook link at the top of the page. This applies regardless of whether Outlook or some other email client is used.

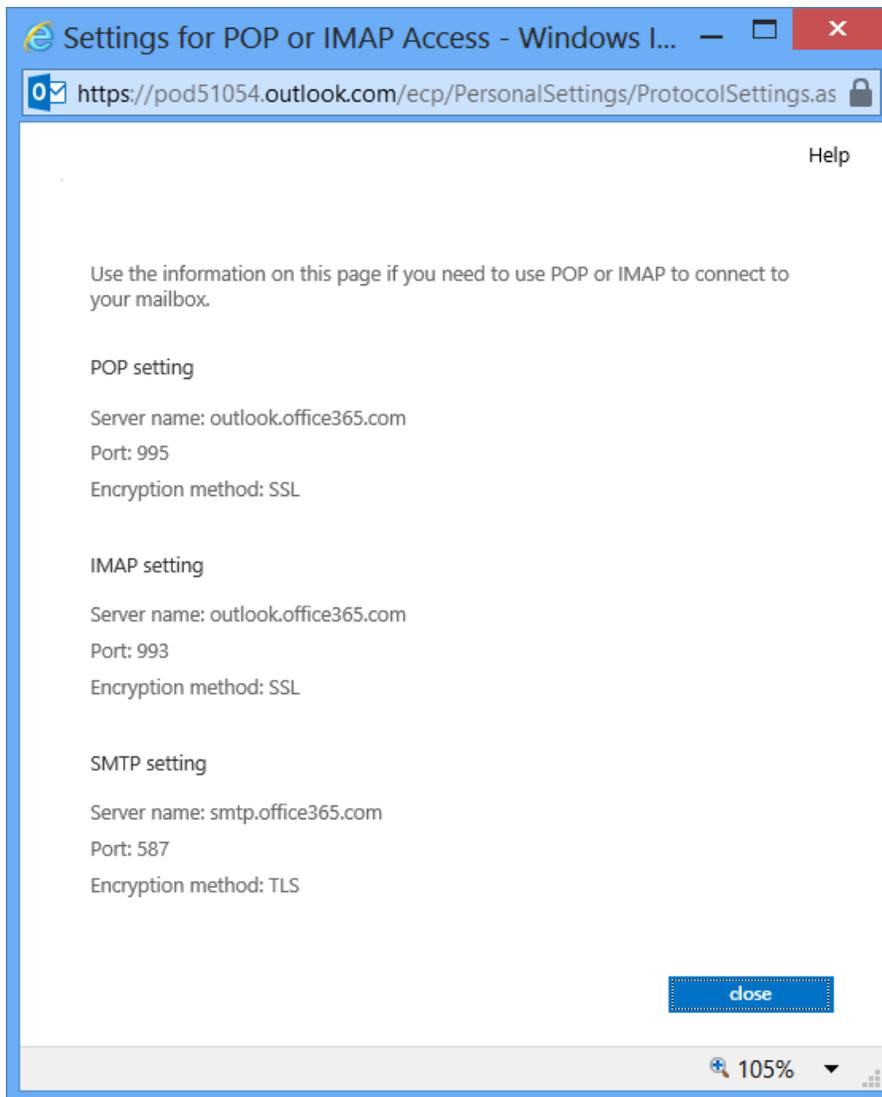


Click the settings cog at the top right and select Options.

## ComponentSpace SAML v2.0 for .NET Developer Guide



Click Settings for POP or IMAP access and take note of these settings.



Open the Microsoft Outlook client and configure a new account.

**Add Account**

**Auto Account Setup**  
Manual setup of an account or connect to other server types.

**E-mail Account**

Your Name:   
Example: Ellen Adams

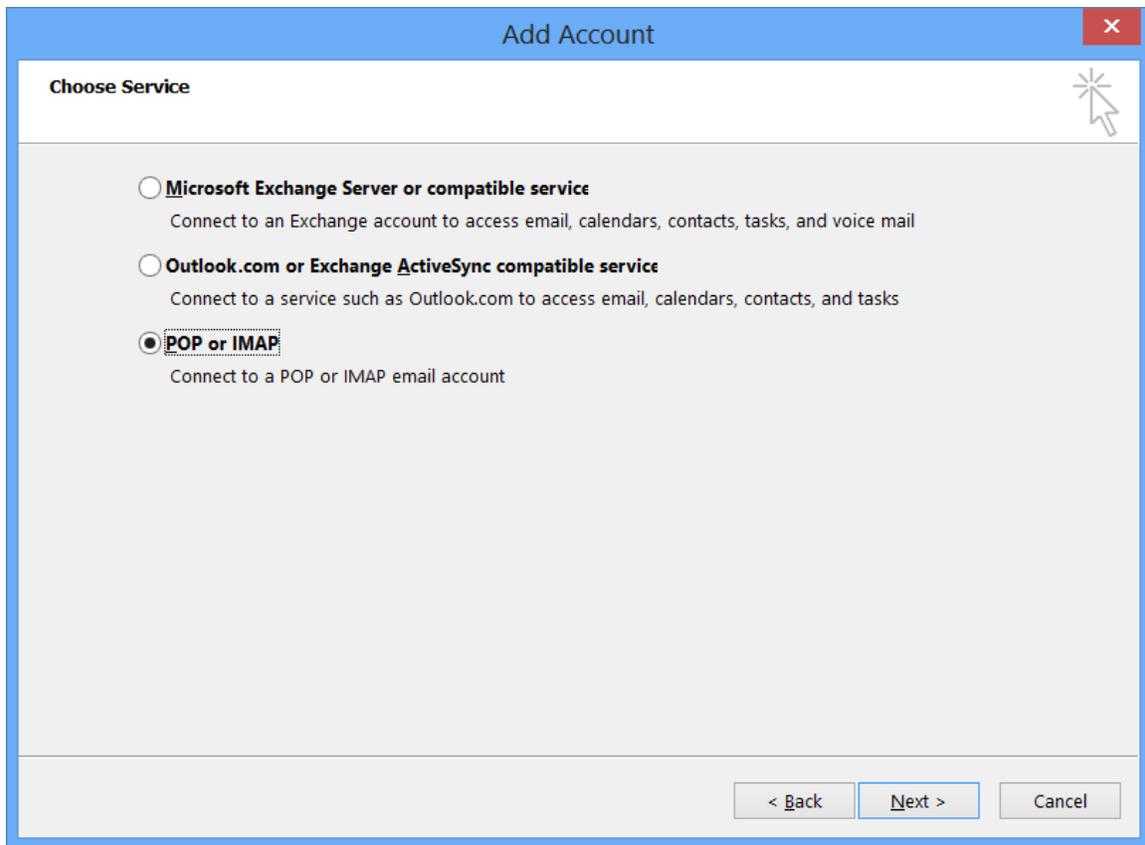
E-mail Address:   
Example: ellen@contoso.com

Password:

Retype Password:   
Type the password your Internet service provider has given you.

**Manual setup or additional server types**

< Back   Next >   Cancel



Specify the settings from Office 365.

**Add Account** ✕

---

**POP and IMAP Account Settings**  
Enter the mail server settings for your account. ✖

**User Information**

Your Name:

Email Address:

**Server Information**

Account Type:  ▼

Incoming mail server:

Outgoing mail server (SMTP):

**Logon Information**

User Name:

Password:

Remember password

Require logon using Secure Password Authentication (SPA)

---

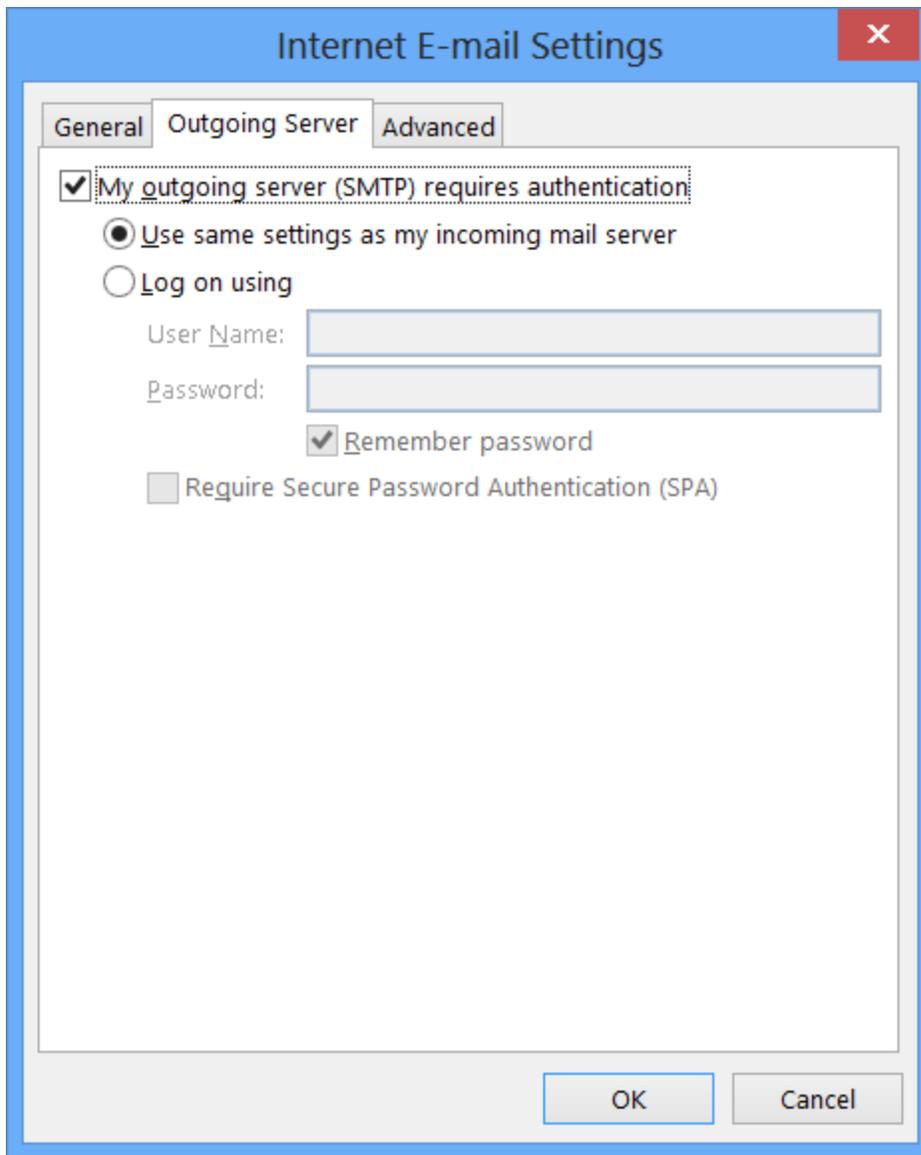
**Internet E-mail Settings** [X]

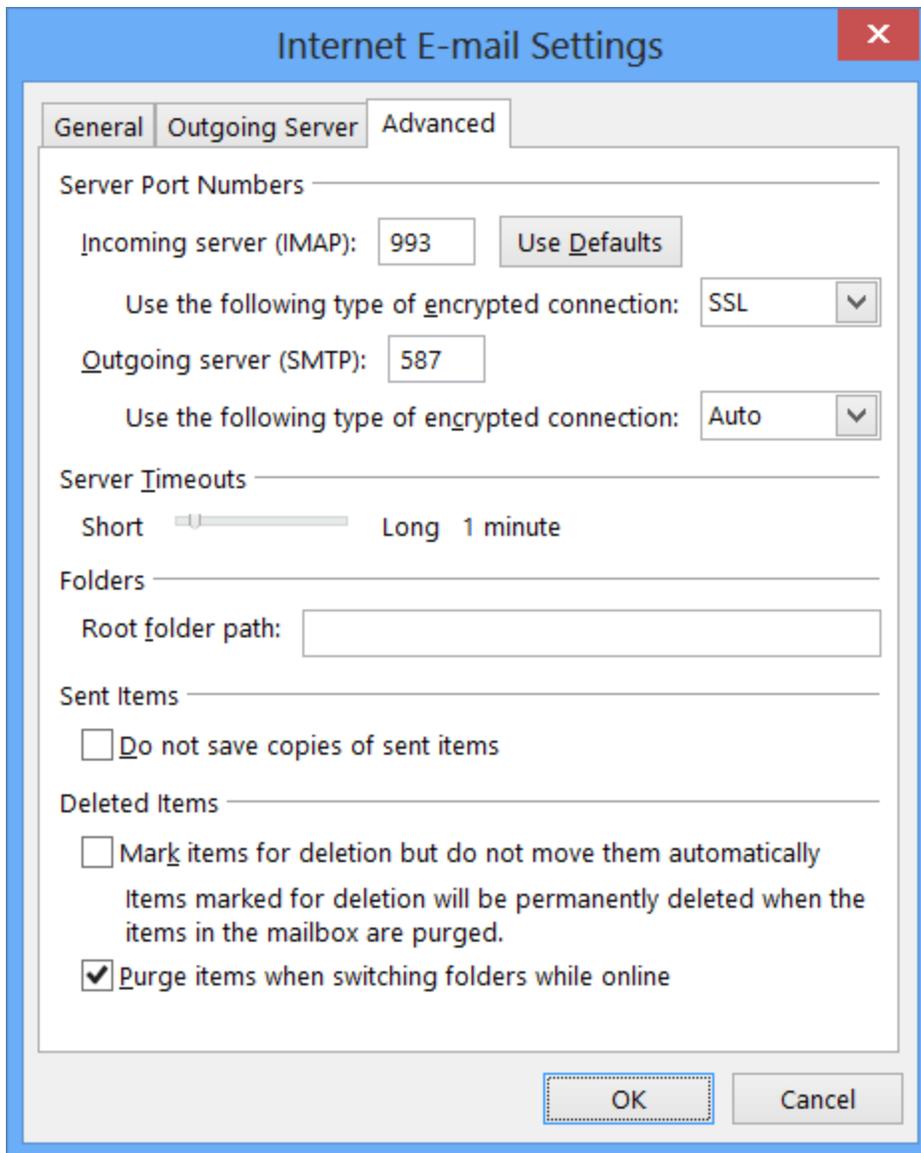
General | **Outgoing Server** | Advanced

Mail Account \_\_\_\_\_  
Type the name by which you want to refer to this account. For example: "Work" or "Microsoft Mail Server"

Other User Information \_\_\_\_\_  
Organization:   
Reply E-mail:

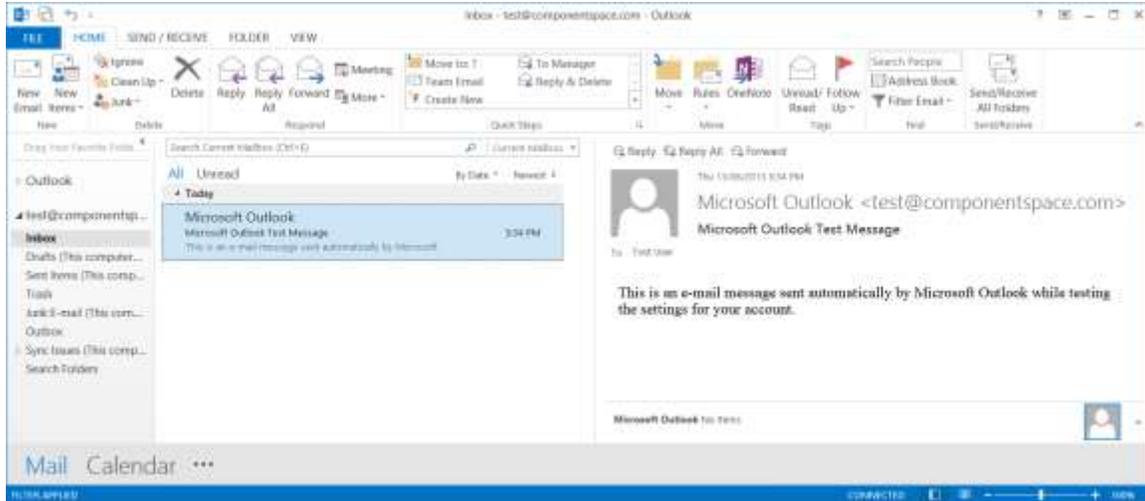
OK Cancel





### 10.5.9 Running the Email Client

Running the email client will cause user authentication to occur at the identity provider.



### 10.5.10 Troubleshooting Office 365 SSO

Refer to the Office 365 documentation pages. The following article lists error codes and recommended actions.

<http://support.microsoft.com/kb/2615736>

Office 365 SAML metadata may be retrieved from:

<https://nexus.microsoftonline-p.com/federationmetadata/saml20/federationmetadata.xml>

## 10.6 Google Apps Interoperability

The Web Forms and MVC example identity providers demonstrate SP initiated single sign-on with Google Apps.

The following sections describe the configuration for the Web Forms identity service provider but, with the appropriate changes, apply equally to the MVC example identity provider.

Refer to sections 10.1 and 10.1.7 for installing and configuring the Web Forms and MVC example identity providers.

### 10.6.1 Configuring the Identity Provider

The saml.config file includes the following entry for the Google Apps partner service provider.

```
<PartnerServiceProvider Name="google.com"
  WantAuthnRequestSigned="false"
  SignSAMLResponse="true"
  SignAssertion="false"
  EncryptAssertion="false"/>
```

The name matches with the issuer name Google Apps uses in the authn request.

The assertion consumer service URL specified in the authn request is used rather than configuring this URL. Alternatively, the assertion consumer service URL may be configured (e.g. <https://www.google.com/a/<domain-name>/acs>).

## 10.6.2 Configuring Google Apps

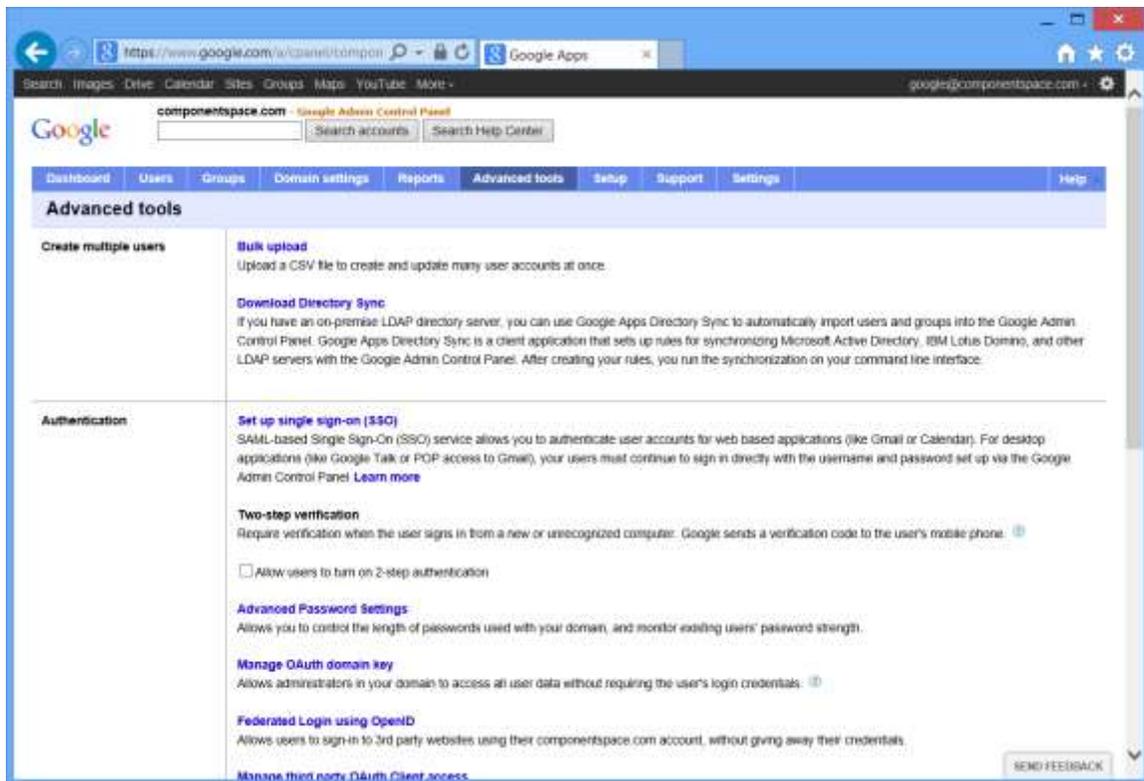
Login as an administrator to the Google Admin Control Panel at:

<https://www.google.com/a/<domain-name>>

For example:

<https://www.google.com/a/componentspace.com>

Select the Advanced tools tab and under the Authentication section, click the Set up single sign-on (SSO) link.



Specify the sign-in page URL. This is the identity provider's single sign-on service where the authn request is sent.

For example:

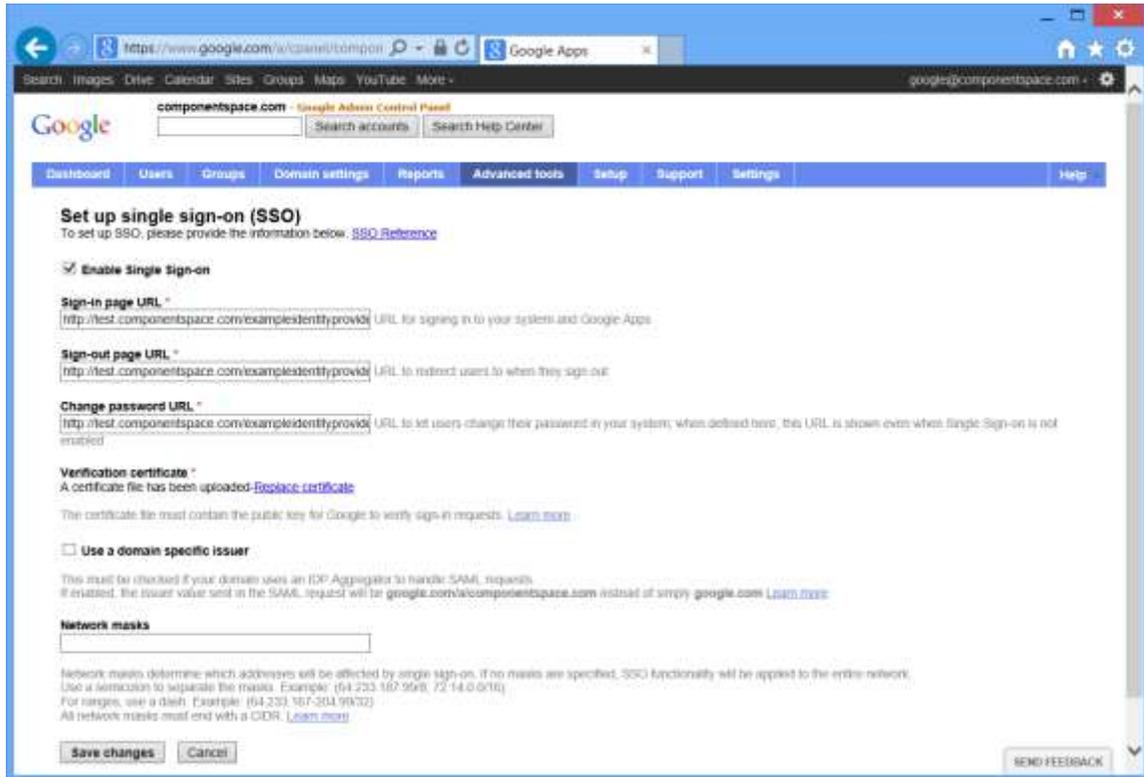
<https://test.componentspace.com/ExampleIdentityProvider/SAML/SSOService.aspx>

Specify the sign-out page URL. When the user signs out of Google Apps they are redirected to the sign-out page. Google Apps does not send a SAML logout request.

Specify the change password URL. The user is redirected to this page when they wish to change their password. This does not involve SAML.

Upload the identity provider's certificate.

For example: idp.cer.



### 10.6.3 Running Google Apps with SSO

In this example, the user is attempting to access a protected resource on the service provider and, rather than performing a local login at the service provider, SSO is initiated with a local login occurring at the identity provider and the asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the service provider.

1. Browse to:

<https://mail.google.com/a/<domain-name>>

For example:

<https://mail.google.com/a/componentspace>

2. You should then be presented with the identity provider login prompt.
3. Login using the user name and password of a user known to the identity provider. The user account should also exist in Google Apps.

4. You should then be presented with the Google Mail default page.

This means you've successfully completed a SAML v2.0 SSO and are logged in at Google Apps with your identity provider user name.

### 10.6.4 Troubleshooting Google Apps SSO

Refer to the Troubleshooting Single Sign-On (SSO) article in the Google Apps documentation.

## 10.7 Salesforce Interoperability

The Web Forms and MVC example identity providers demonstrate both IdP and SP initiated single sign-on with Salesforce.

The following sections describe the configuration for the Web Forms identity and service providers but, with the appropriate changes, apply equally to the MVC example identity and service providers.

Refer to sections 10.1 and 10.1.7 for installing and configuring the Web Forms and MVC example identity providers.

### 10.7.1 Configuring the Identity Provider

The saml.config file includes the following entry for the Salesforce partner service provider.

```
<PartnerServiceProvider Name="https://saml.salesforce.com"
  WantAuthnRequestSigned="false"
  SignSAMLResponse="true"
  SignAssertion="false"
  EncryptAssertion="false"
  AssertionConsumerServiceURL=
    "https://login.salesforce.com"/>
```

The web.config file identifies the partner service provider. This must specify the Salesforce service provider.

```
<add key="PartnerSP" value="https://saml.salesforce.com"/>
```

### 10.7.2 Configuring Salesforce as a Service Provider

Login as an administrator to Salesforce at:

<https://login.salesforce.com>

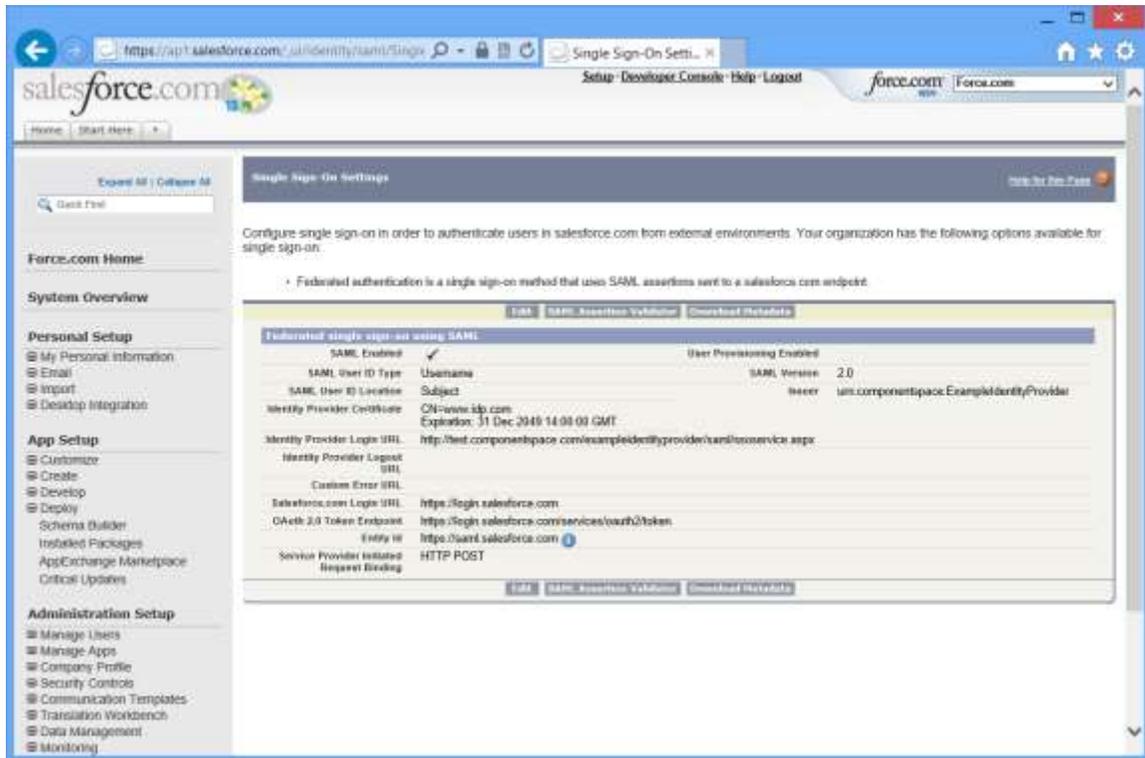
Select Setup > Security Controls > Single Sign-On Settings.

Enable SAML.

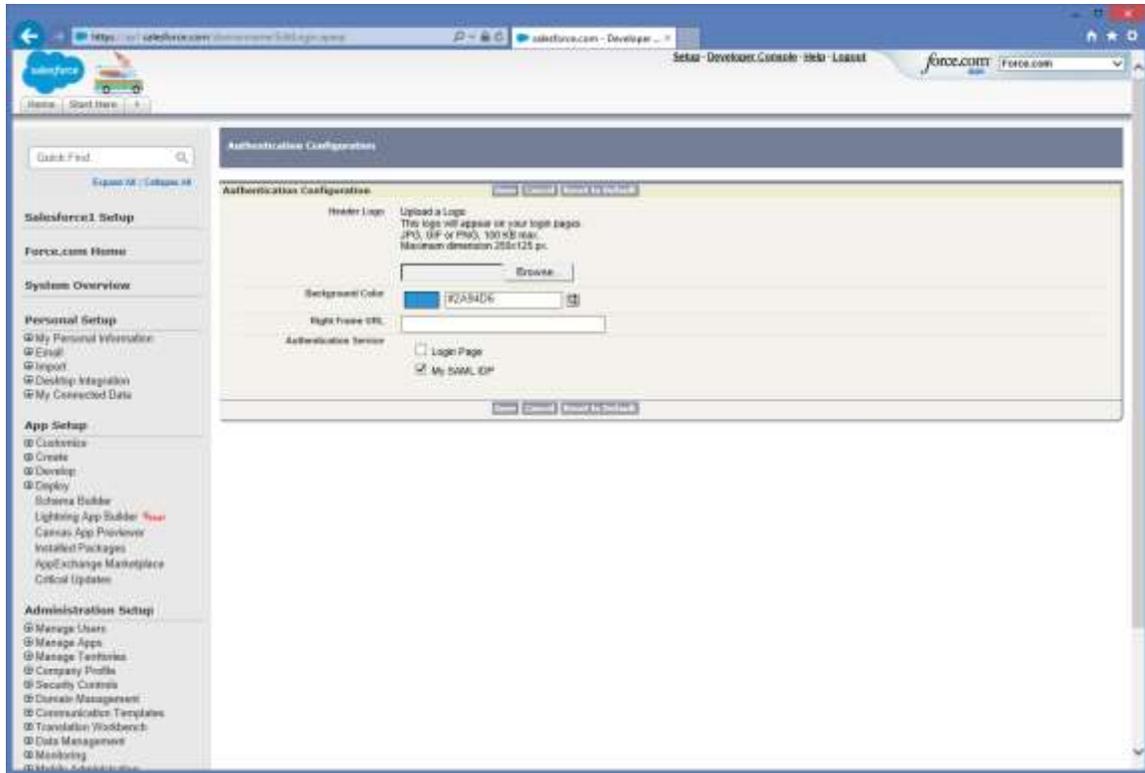
Specify the issuer, upload the identity provider's certificate and specify the login URL.

For example, the issuer is urn:componentSpace:ExampleIdentityProvider, upload the idp.cer file, and specify

<http://test.componentspace.com/ExampleIdentityProvider/SAML/SSOService.aspx> as the login URL.



For SP-initiated SSO, select Setup > Domain Management > My Domain.  
Ensure the Authentication Service is set to My SAML IDP.



Take note of the domain name. Browsing to this domain name will start SP-initiated SSO.

### 10.7.3 Running the Example Identity Provider – IdP-Initiated SSO

1. Browse to:

<http://localhost/ExampleIdentityProvider>

You should then be presented with the identity provider login prompt.

2. Login using the user name and password of a user known to the identity provider.

The user account must also exist in Salesforce.

You may have to update the credentials section of the example identity provider's web.config to include the user name.

3. Click the link to SSO to Salesforce.

The Salesforce main page should be displayed.

This means you've successfully completed a SAML v2.0 SSO and are logged in at Salesforce with your identity provider user name.

## 10.7.4 Running the Example Identity Provider – SP-Initiated SSO

4. Browse to the domain name configured in Salesforce:

<http://componentspace-dev-ed.my.salesforce.com/>

Refer to section 10.7.2 for information on determining the domain name.

You should then be presented with the identity provider login prompt.

5. Login using the user name and password of a user known to the identity provider.

The user account must also exist in Salesforce.

You may have to update the credentials section of the example identity provider's web.config to include the user name.

6. The Salesforce main page should be displayed.

This means you've successfully completed a SAML v2.0 SSO and are logged in at Salesforce with your identity provider user name.

## 10.7.5 Configuring the Service Provider

The saml.config file includes the following entry for the Salesforce partner identity provider.

```
<PartnerIdentityProvider
  Name="https://componentspace-dev-ed.my.salesforce.com"
  SignAuthnRequest="false"
  WantSAMLResponseSigned="true"
  WantAssertionSigned="false"
  WantAssertionEncrypted="false"
  UseEmbeddedCertificate="true"
  SingleSignOnServiceUrl=
"https://componentspace-dev-
ed.my.salesforce.com/idp/endpoint/HttpRedirect"/>
```

The partner identity provider name must match with the issuer name generated by Salesforce. See section 10.7.6.

The web.config file identifies the partner identity provider. This must specify the Salesforce identity provider.

```
<add key="PartnerIdP"
  value=" https://componentspace-dev-ed.my.salesforce.com "/>
```

## 10.7.6 Configuring Salesforce as an Identity Provider

Login as an administrator to Salesforce at:

<https://login.salesforce.com>

Select Setup > Security Controls > Identity Provider.

Enable the Identity Provider. The generated self-signed certificate is okay to use.

Add a service provider.

Specify the name as ExampleServiceProvider and the entity ID as urn:componentspace:ExampleServiceProvider.

Specify the ACS URL. For example:

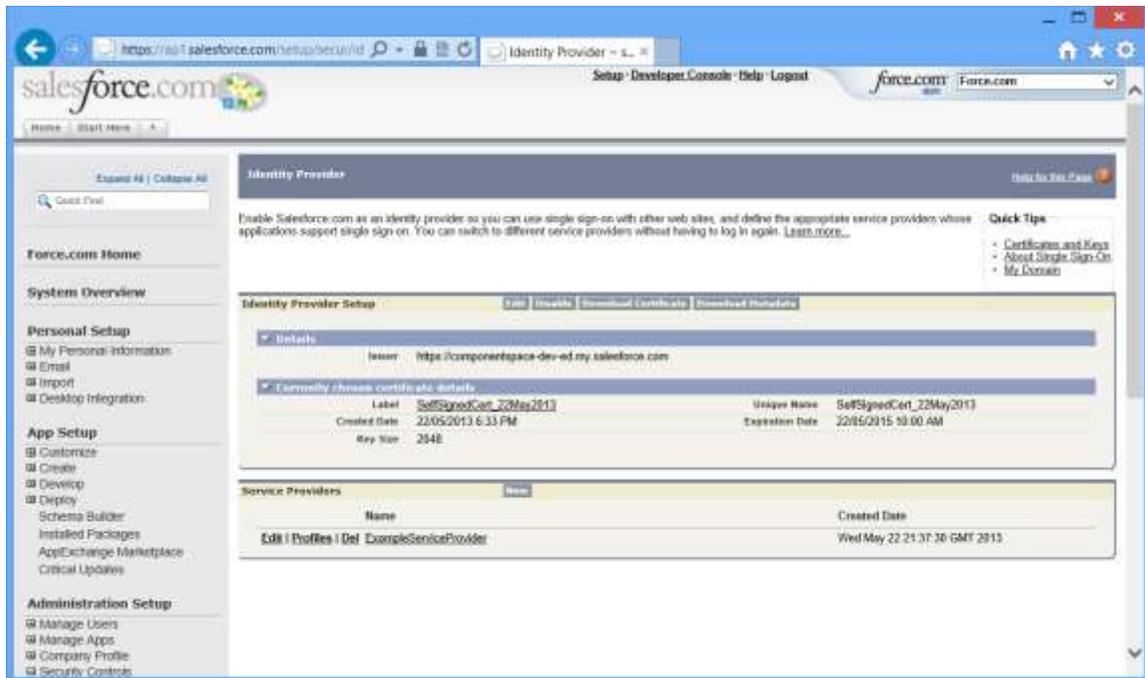
<http://test.componentspace.com/ExampleServiceProvider/SAML/AssertionConsumerService.aspx>

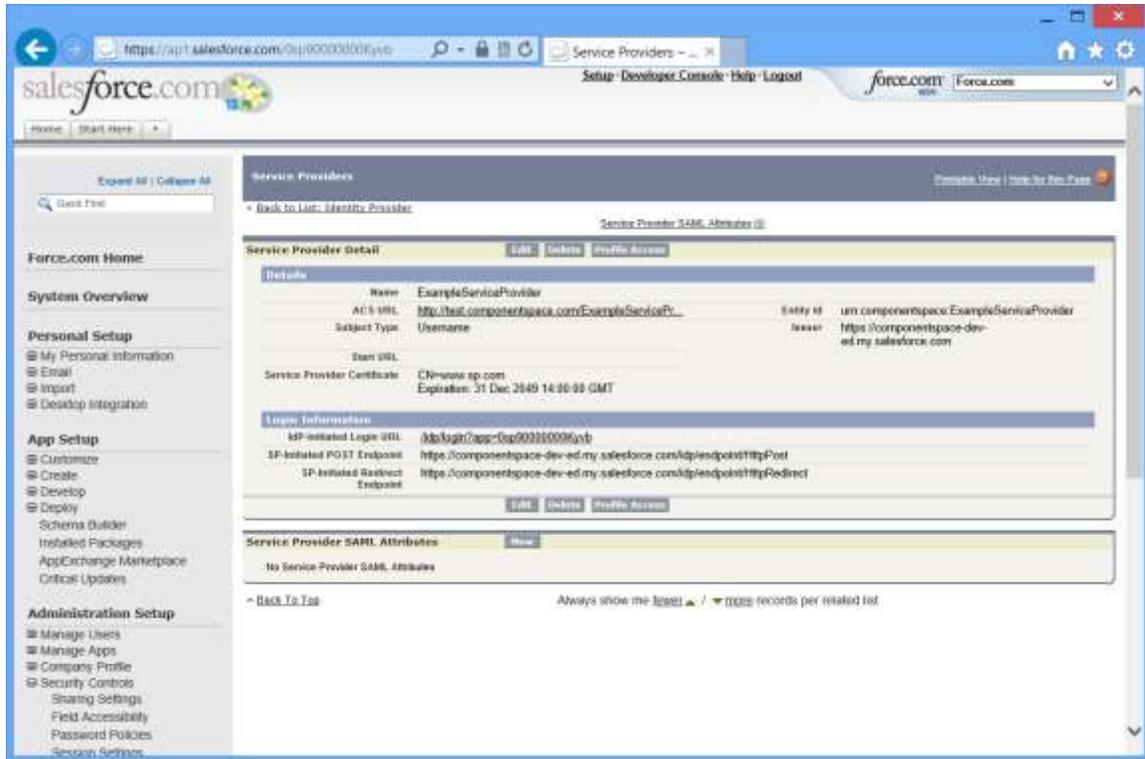
A start URL is not required.

Select user name as the subject type.

Select the user profiles which will have access to the service provider.

Note that if SAML is enabled as described in section 10.7.2, the identity provider role is delegated to the configured identity provider. To have Salesforce act as the identity provider, disable SAML.





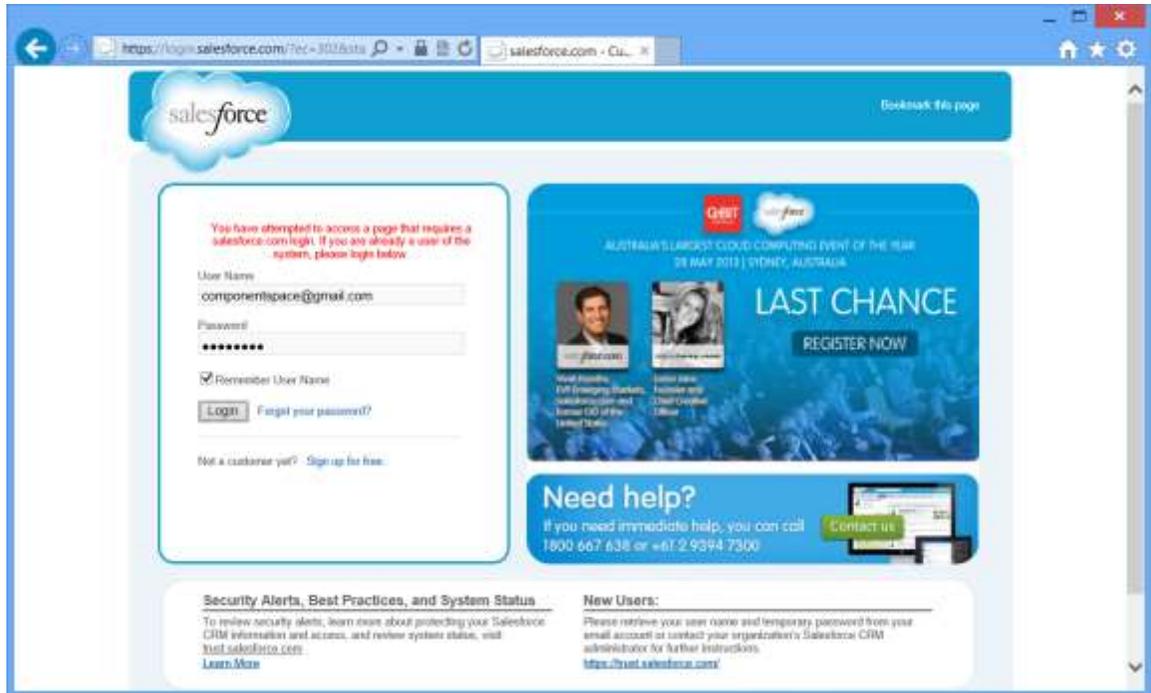
### 10.7.7 Running the Example Service Provider – IdP-Initiated SSO

1. Browse to the IdP-Initiated login URL specified under the login information for the service provider in the Salesforce configuration. For example:

<https://ap1.salesforce.com/idp/login?app=0sp90000000Kyvb>

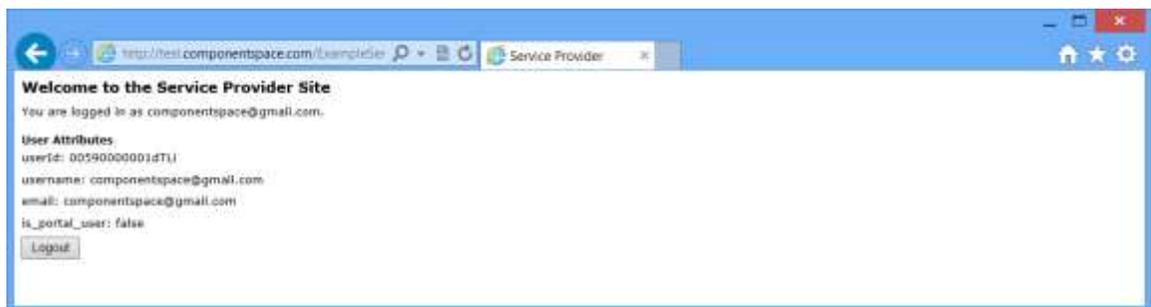
You should then be presented with the Salesforce login prompt.

2. Login using the user name and password of a user known to Salesforce.



3. The example service provider page should be displayed.

This means you've successfully completed a SAML v2.0 SSO and are logged in at the example service provider with your Salesforce user name.



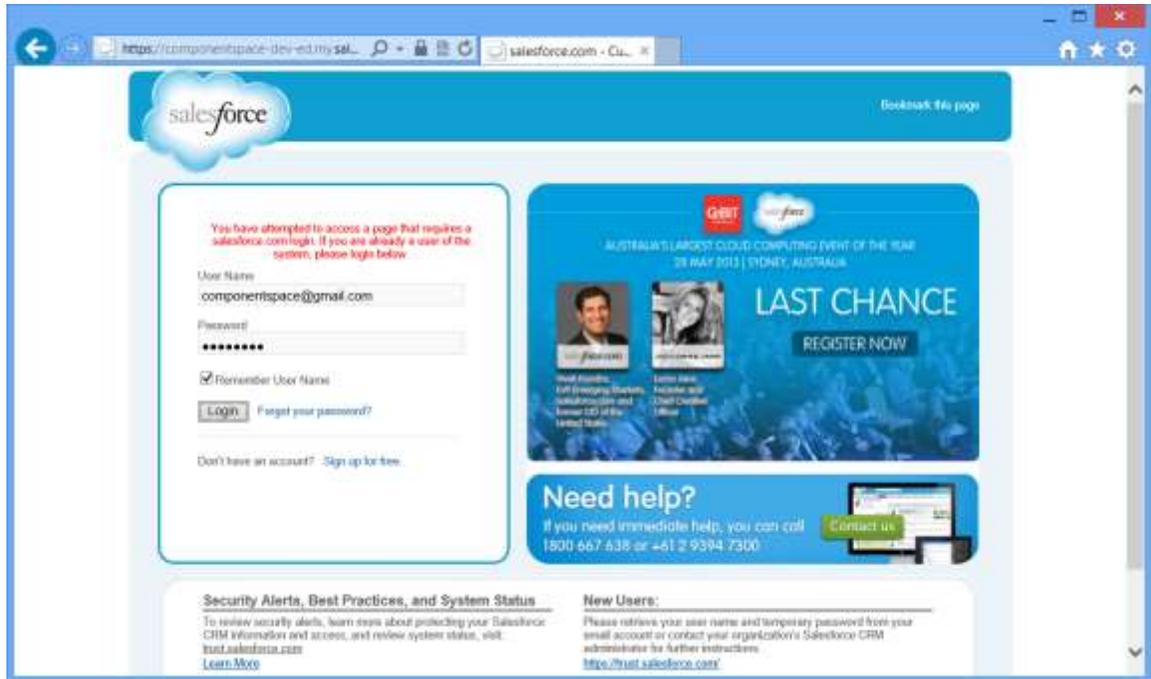
### 10.7.8 Running the Example Service Provider – SP-Initiated SSO

1. Browse to the example service provider. For example:

<http://test.componentspace.com/ExampleServiceProvider>

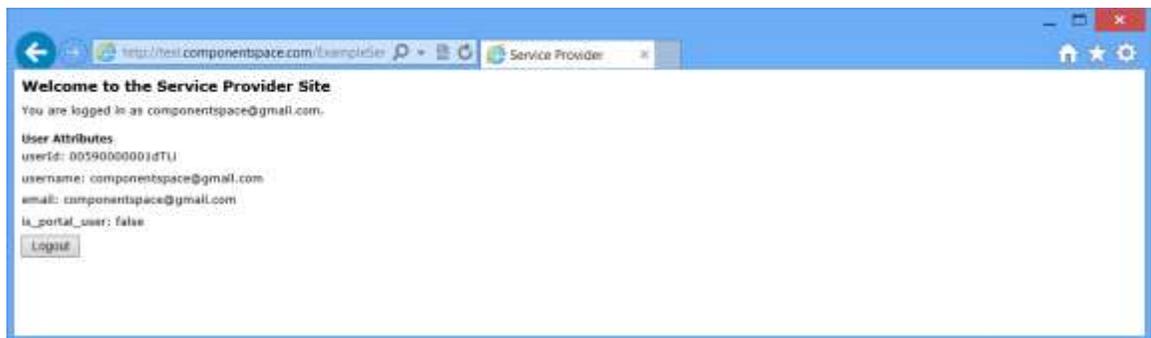
You should then be presented with the Salesforce login prompt.

2. Login using the user name and password of a user known to Salesforce.



3. The example service provider page should be displayed.

This means you've successfully completed a SAML v2.0 SSO and are logged in at the example service provider with your Salesforce user name.



### 10.7.9 Troubleshooting Salesforce SSO

Use Setup > Security Controls > Single Sign-On Settings > SAML Assertion Validator to debug problems with the SAML response.

View the Login History under Setup > Manage Users > Login History.

Salesforce documentation may be found at:

[https://help.salesforce.com/help/pdfs/en/salesforce\\_single\\_sign\\_on.pdf](https://help.salesforce.com/help/pdfs/en/salesforce_single_sign_on.pdf)

<http://www.salesforce.com/us/developer/docs/sso/index.htm>

## 10.8 Shibboleth Interoperability

The Web Forms and MVC example identity providers demonstrate SP initiated single sign-on with Shibboleth.

The following sections describe the configuration for the Web Forms identity service provider but, with the appropriate changes, apply equally to the MVC example identity provider.

Refer to sections 10.1 and 10.1.7 for installing and configuring the Web Forms and MVC example identity providers.

### 10.8.1 Configuring the Identity Provider

The saml.config file identifies the local identity provider. This must match with the entity ID specified in the metadata uploaded to Shibboleth.

```
<IdentityProvider Name="https://test.componentspace.com"/>
```

The saml.config file includes the following entry for the Shibboleth partner service provider.

```
<PartnerServiceProvider Name="https://sp.testshib.org/shibboleth-sp"
    WantAuthnRequestSigned="false"
    SignSAMLResponse="true"
    SignAssertion="false"
    EncryptAssertion="false"
    AssertionConsumerServiceURL=
        "https://sp.testshib.org/Shibboleth.sso/SAML2/POST"/>
```

### 10.8.2 Configuring the Service Provider

The saml.config file identifies the local service provider. This must match with the entity ID specified in the metadata uploaded to Shibboleth.

```
<ServiceProvider Name="https://test.componentspace.com"/>
```

The saml.config file includes the following entry for the Shibboleth partner service provider.

```
<PartnerIdentityProvider Name="https://idp.testshib.org/idp/shibboleth"
    SignAuthnRequest="true"
    WantSAMLResponseSigned="true"
    WantAssertionSigned="false"
    WantAssertionEncrypted="false"
    UseEmbeddedCertificate="true"
    SingleSignOnServiceUrl=
        "https://idp.testshib.org/idp/profile/SAML2/Redirect/SSO"/>
```

The web.config file identifies the partner identity provider. This must specify the Shibboleth identity provider.

```
<add key="PartnerIdP" value="https://idp.testshib.org/idp/shibboleth"/>
```

### 10.8.3 Configuring Shibboleth

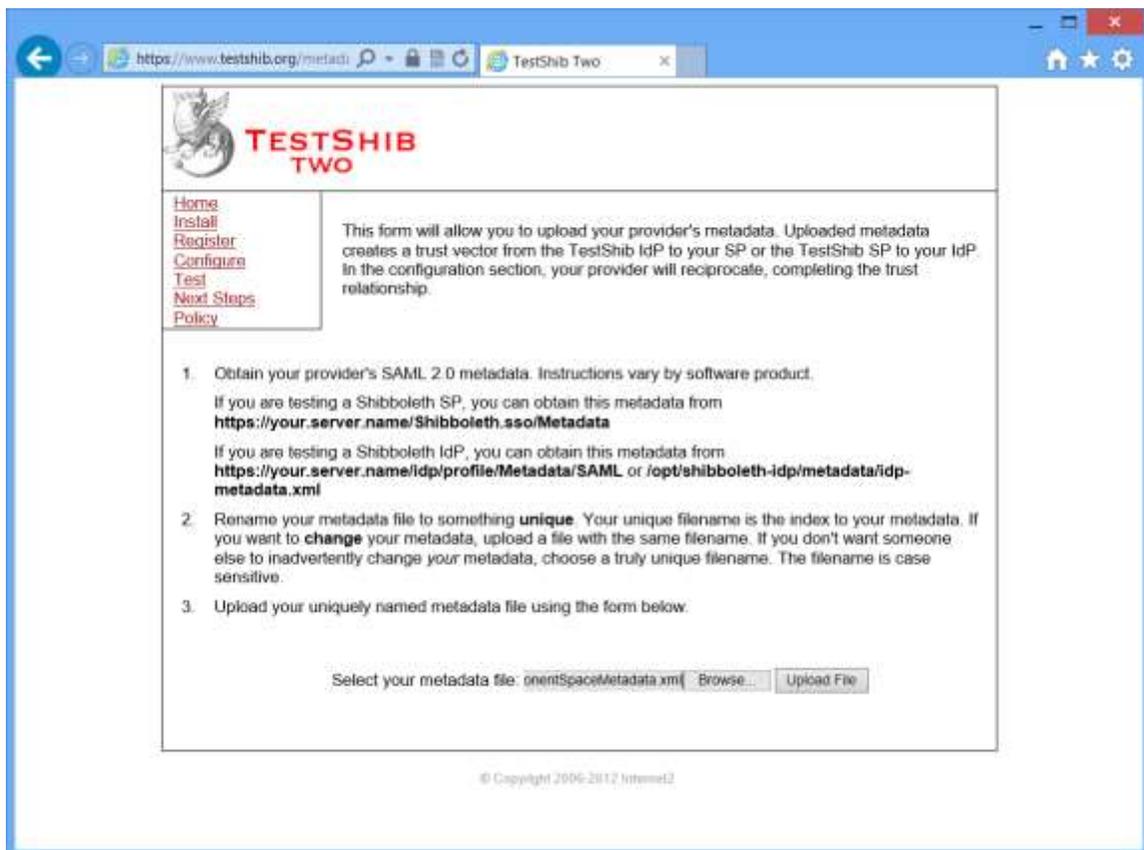
The supplied ComponentSpaceMetadata.xml includes metadata for the example identity provider and the example service provider. The entity ID must uniquely identify your organization. The URLs must be modified to match your configuration.

The metadata to modify may be found at C:\Program Files (x86)\ComponentSpace SAML v2.0 for .NET\Examples\Metadata\Template.

Once the metadata has been updated, navigate to:

<https://www.testshib.org/>

Click the Register button and upload your SAML metadata.



Confirm that the metadata has been uploaded successfully.



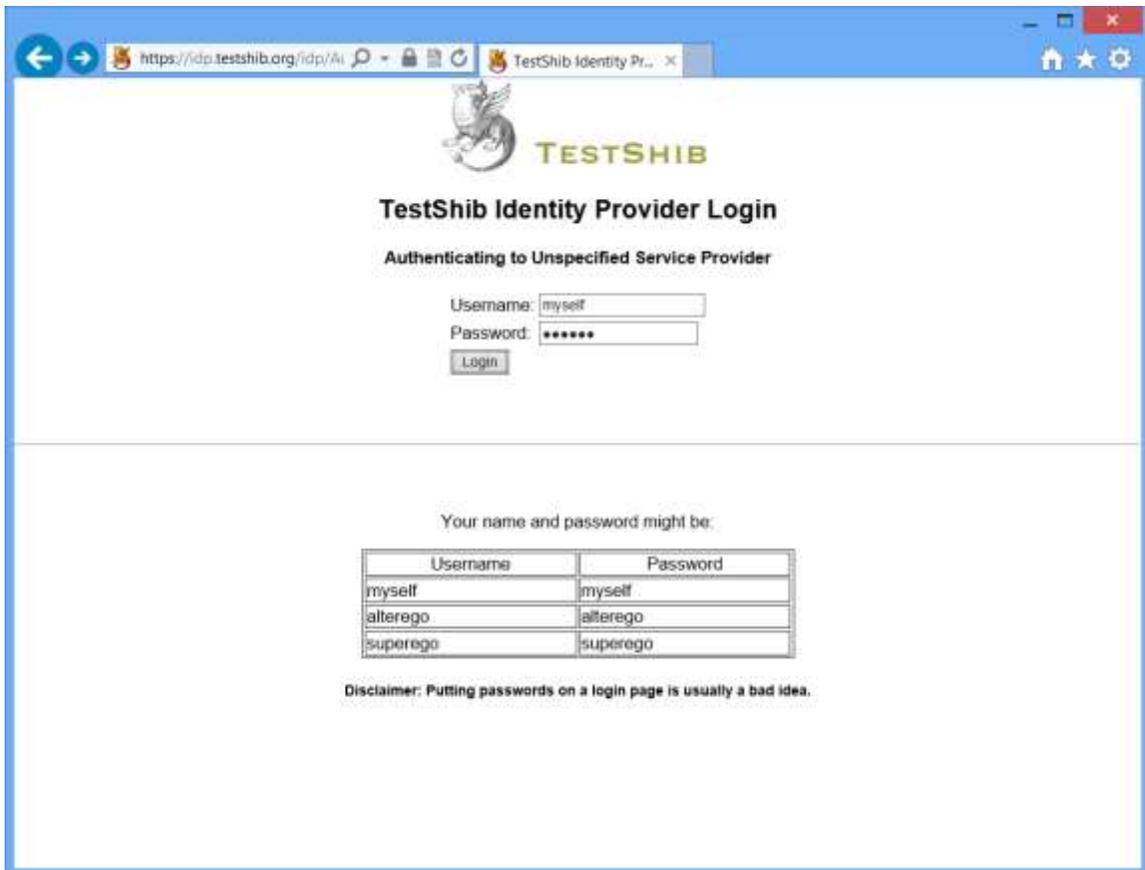


2. You should then be presented with the identity provider login prompt.
3. Login and you should be returned to Shibboleth.

<https://sp.testshib.org/testing/sample.jsp>

This means you've successfully completed a SAML v2.0 SSO and are logged in at Shibboleth with your identity provider user name.





3. Login using one of the listed user names and passwords.
4. The example service provider main page should be displayed.

This means you've successfully completed a SAML v2.0 SSO and are logged in at the example service provider with your Shibboleth user name.



## 10.8.6 Troubleshooting Shibboleth SSO

The Shibboleth identity provider logs may be viewed at:

<https://idp.testshib.org/cgi-bin/idplog.cgi?lines=300&logname=shibd.log>.

The Shibboleth service provider logs may be viewed at:

<https://sp.testshib.org/cgi-bin/splog.cgi?lines=300&logname=shibd.log>.

## 11 Example Applications - Low Level APIs

The example applications must be built and published prior to their use.

The following sections describe the installation and execution of these example applications.

The example applications described in sections 10.1 and 11.2 demonstrate SP initiated SSO and logout using a number of different bindings. These applications are written in VB.NET.

The example applications described in sections 11.3 and 11.4 demonstrate IdP initiated SSO and logout using a number of different bindings. These applications are written in C#.

The example application described in section 11.5 and 11.5 demonstrates interoperability with Active Directory Federation Services (ADFS). This application is written in C#.

The example application described in section 11.6 demonstrates interoperability with Google Apps. This application is written in C#.

The example application described in section 11.7 demonstrates interoperability with Salesforce. This application is written in C#.

The example applications in sections 11.8 and 11.9 demonstrate interoperability with the Shibboleth open source SSO software package. These applications are written in C#.

The example application in section 11.10 demonstrates creating and manipulating SAML assertions. This application is written in VB.NET.

The example applications in section 11.11 demonstrate creating and manipulating SAML metadata. These applications are written in C#.

The example applications in section 11.12 demonstrate generating and verifying XML signatures. These applications are written in C#.

Section 11.13 includes various utility applications.

## **11.1 SP-Initiated SSO – Identity Provider**

The SAML2IdentityProvider web application, in conjunction with the SAML2ServiceProvider web application, demonstrates SP initiated single sign-on and logout.

### **11.1.1 Installing the Identity Provider**

6. Using Visual Studio, build and publish the web application.
7. Open the Internet Information Services management console.
8. Under the default web site for the local computer, create an application with an alias of SAML2IdentityProvider.
9. For the physical path, browse to the directory where SAML2IdentityProvider was built and published.
10. Ensure the web application has been successfully installed and configured by browsing to <http://localhost/SAML2IdentityProvider>.

### **11.1.2 Configuring the Identity Provider**

The identity provider configuration is contained within its web.config file's <appSettings> section and contains the *spArtifactResponderURL* and *spLogoutURL*.

The *spArtifactResponderURL* specifies the URL of the service provider's artifact responder. Its default value is <http://localhost/SAML2ServiceProvider/SAML/ArtifactResponder.aspx>.

The *spLogoutURL* specifies the URL of the service provider's logout service. Its default value is <http://localhost/SAML2ServiceProvider/SAML/SingleLogoutService.aspx>.

Modifying web.config does not require an application restart.

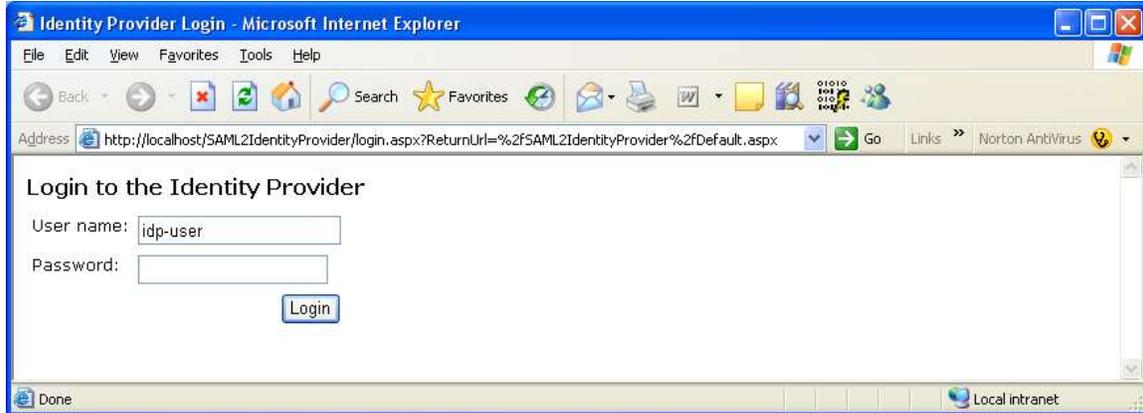
If you use the default installation you won't need to modify this configuration.

### **11.1.3 Running the Identity Provider**

As this is SP initiated SSO, you need to run the service provider application rather than the identity provider to initiate SSO.

In this example SSO is not being used. Instead, you should simply perform a local login at the identity provider to ensure it is functioning correctly.

1. Browse to <http://localhost/SAML2IdentityProvider>.
2. Login using the user name *idp-user* and a password of *password* (see Figure 31).
3. Verify that you've been redirected to the identity provider's default page (see Figure 32).



**Figure 31 Identity Provider Login Page**



**Figure 32 Identity Provider Default Page**

#### 11.1.4 Running the Identity Provider in Visual Studio

You may run the identity provider in Visual Studio. The one additional step is to note the port number being used by Visual Studio to run the application. You then need to update the service provider's configuration as described in section 11.2.2 to account for the different port number being used by Visual Studio.

## 11.2 SP-Initiated SSO – Service Provider

The SAML2ServiceProvider web application, in conjunction with the SAML2IdentityProvider web application, demonstrates SP initiated single sign-on and logout.

### 11.2.1 Installing the Service Provider

1. Using Visual Studio, build and publish the web application.
2. Open the Internet Information Services management console.
3. Under the default web site for the local computer, create an application with an alias of SAML2ServiceProvider.
4. For the physical path, browse to the directory where SAML2ServiceProvider was built and published.
5. Ensure the web application has been successfully installed and configured by browsing to <http://localhost/SAML2ServiceProvider>.

### 11.2.2 Configuring the Service Provider

The service provider configuration is contained within its web.config file's <appSettings> section and contains the *idpsssoURL*, *idpArtifactResponderURL* and *idpLogoutURL*.

The *idpsssoURL* specifies the URL of the identity provider's SSO service. Its default value is <http://localhost/SAML2IdentityProvider/SAML/SSOService.aspx>.

The *idpArtifactResponderURL* specifies the URL of the identity provider's artifact responder. Its default value is <http://localhost/SAML2IdentityProvider/SAML/ArtifactResponder.aspx>.

The *idpLogoutURL* specifies the URL of the identity provider's logout service. Its default value is <http://localhost/SAML2IdentityProvider/SAML/SingleLogoutService.aspx>.

Modifying web.config does not require an application restart.

If you use the default installation you won't need to modify this configuration.

### 11.2.3 Running the Service Provider without SSO

In this example SSO is not being used. Instead, you should simply perform a local login at the service provider to ensure it is functioning correctly.

1. Browse to <http://localhost/SAML2ServiceProvider>.
2. You should be presented with the form shown in Figure 35.
3. Select the service provider as the location where login will occur.
4. Login using the user name *sp-user* and a password of *password* (see Figure 33).
5. Verify that you've been redirected to the service provider's default page (see Figure 34).

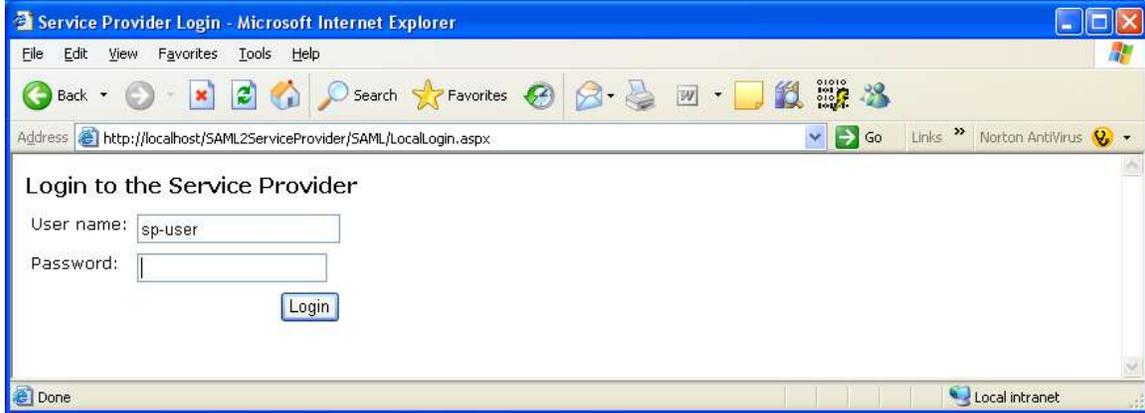


Figure 33 Service Provider Login Page



Figure 34 Service Provider Default Page – Logged in as sp-user

## 11.2.4 Running the Service Provider with SSO

In this example, the user is attempting to access a protected resource on the service provider and, rather than performing a local login at the service provider, SSO is initiated with a local login occurring at the identity provider and the asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the service provider.

8. Browse to <http://localhost/SAML2ServiceProvider>.
9. You should be presented with the form shown in Figure 35.
10. Select the identity provider as the location where login will occur.

Selecting the identity provider will initiate a SAML v2.0 SSO. Selecting the service provider will initiate a local login at the service provider.

11. Select the binding to use when communicating between the service provider and identity provider.

The user experience should be the same regardless of the binding selected. The

only time when this isn't the case is if the HTTP POST binding is selected and Javascript is disabled in which case the user will be presented with an intermediate form and a button they need to click.

12. Select the binding to use when communicating between the identity provider and service provider.

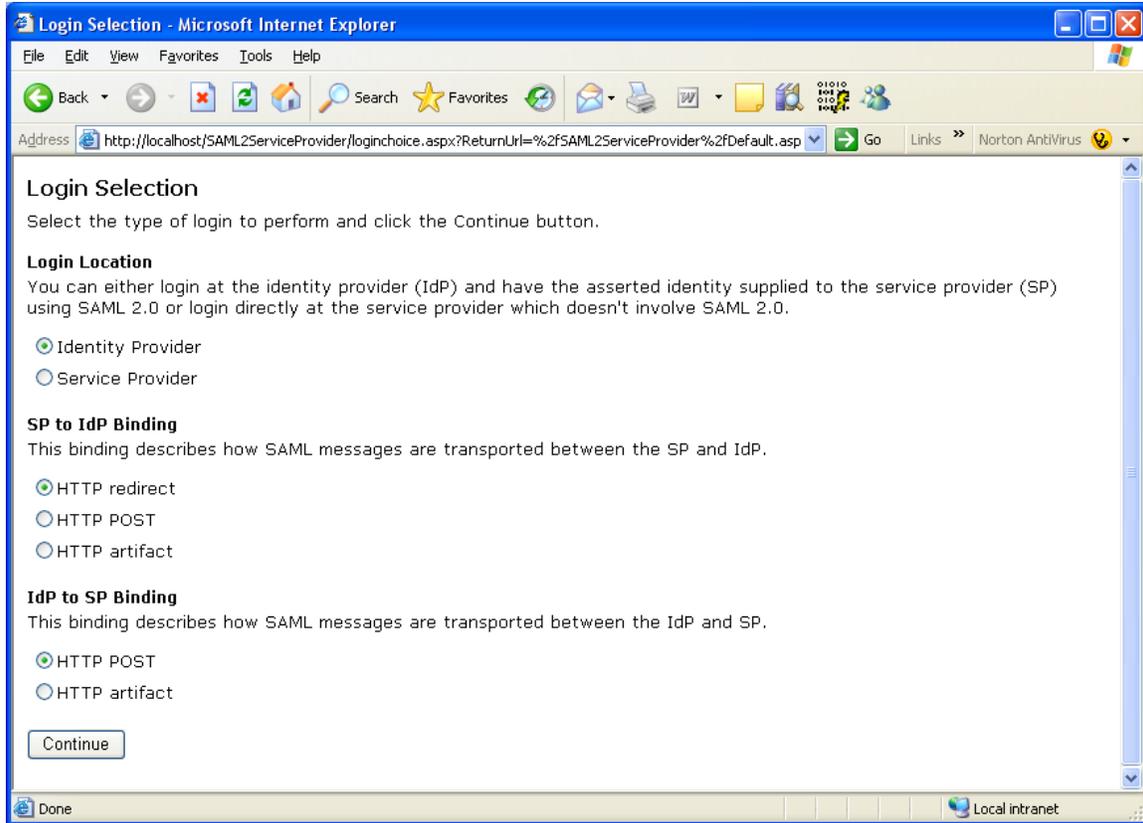
The user experience should be the same regardless of the binding selected. The only time when this isn't the case is if the HTTP POST binding is selected and Javascript is disabled in which case the user will be presented with an intermediate form and a button they need to click.

13. Click the Continue button.
14. You should then be presented with the identity provider login page (see Figure 31) as you will be logging in at the identity provider, not the service provider.

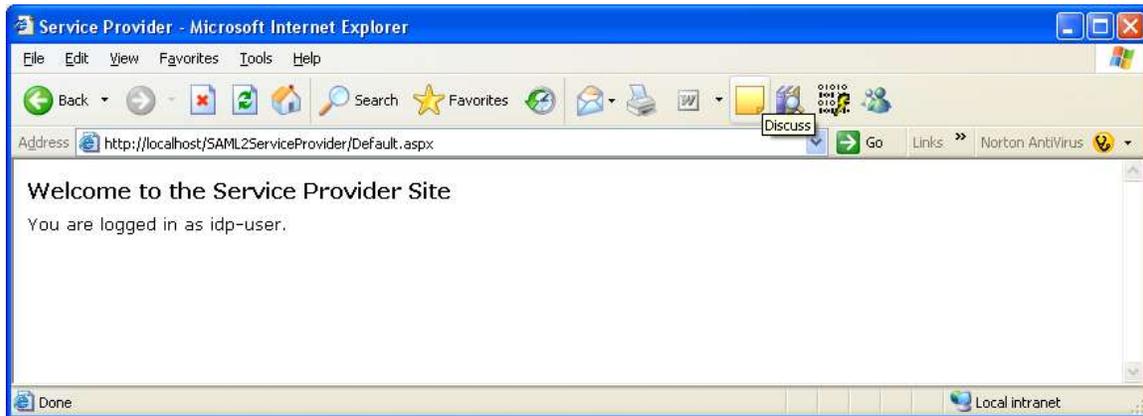
If you are not then you must already have logged in at the identity provider. To force a login, close the browser and start again.

15. Login using the user name *idp-user* and a password of *password*.
16. You should then be presented with the service provider's default page (see Figure 36).

This means you've successfully completed a SAML v2.0 SSO and are logged in at the service provider with your identity provider user name.



**Figure 35 Service Provider Login Selection Page**



**Figure 36 Service Provider Default Page – Logged in as idp-user**

The service provider is coded so as not to force authentication at the identity provider. To demonstrate this login at the identity provider and, using the same browser session, browse to the service provider and select the login location as the identity provider. No login at the identity provider will be required as you have already logged in.

1. Login at the identity provider by following the steps described in section 11.1.3.

- Using the same browser session, follow the steps described in section 11.2.4.

You should not have to login again at the identity provider.

### 11.2.5 Running the Service Provider in Visual Studio

You may run the service provider in Visual Studio. The one additional step is to note the port number being used by Visual Studio to run the application. You then need to update the identity provider's configuration as described in section 11.1.2 to account for the different port number being used by Visual Studio.

### 11.2.6 Service Provider SSO Execution Flow

Figure 37 illustrates the execution flow between the example service provider and identity provider.

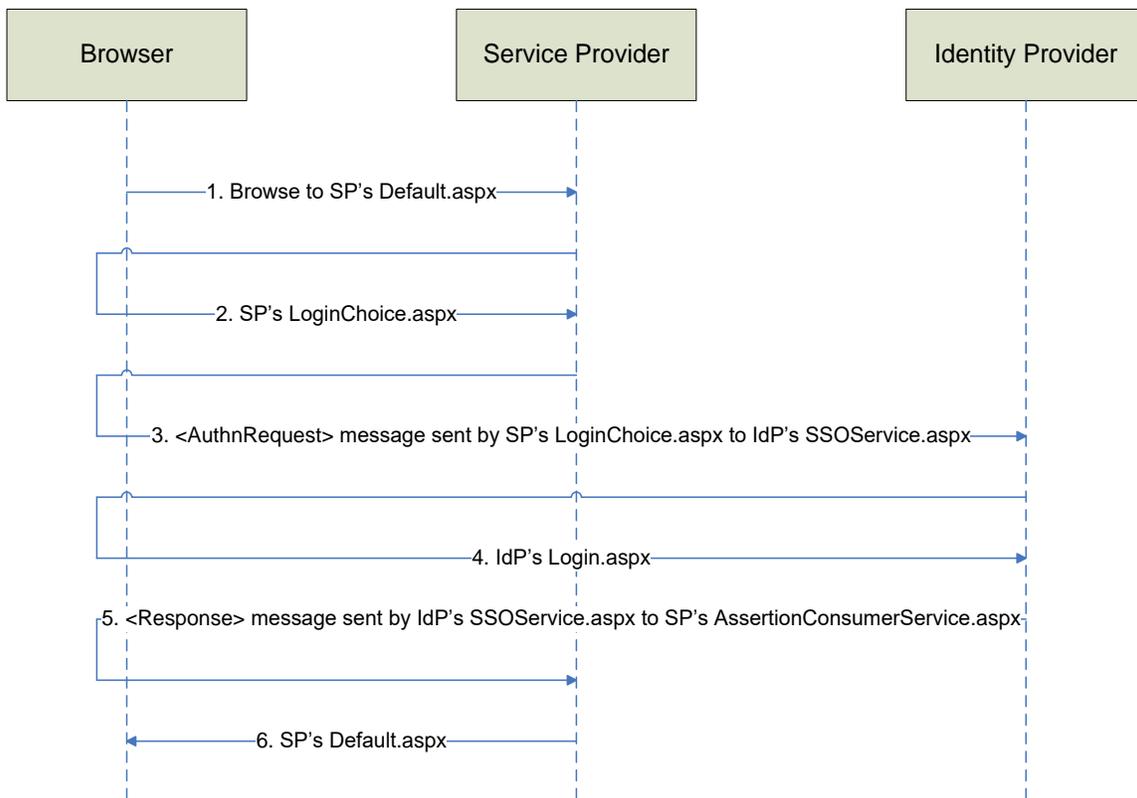


Figure 37 SSO Execution Flow

- The user browses to <http://localhost/SAML2ServiceProvider.aspx>.
- As this is configured in web.config as a protected page, the user is redirected by ASP.NET to the configured login page, LoginChoice.aspx.
- The user selects from the LoginChoice.aspx to login at the identity provider.

The LoginChoice.aspx page sends a SAML v2.0 AuthnRequest protocol message to the identity provider's SSOService.aspx using the selected SP to IdP binding.

4. If the user isn't logged in at the identity provider, the identity provider redirects the user to the identity provider's Login.aspx page.

Once logged in, ASP.NET redirects the user back to the SSOService.aspx page.

5. The SSOService.aspx page returns the asserted identity in a SAML assertion contained in a SAML v2.0 Request protocol message that's sent to the service provider's AssertionConsumerService.aspx page using the specified IdP to SP binding.
6. The service provider performs an automatic login using the asserted identity and redirects the user to the original service provider page (Default.aspx).

### **11.3 IdP-Initiated SSO – Service Provider**

The SAML2SP web application, in conjunction with the SAML2IdP web application, demonstrates IdP initiated single sign-on.

#### **11.3.1 Installing the Service Provider**

1. Using Visual Studio, build and publish the web application.
2. Open the Internet Information Services management console.
3. Under the default web site for the local computer, create an application with an alias of SAML2SP.
4. For the physical path, browse to the directory where SAML2SP was built and published.
5. Ensure the web application has been successfully installed and configured by browsing to <http://localhost/SAML2SP>.

#### **11.3.2 Configuring the Service Provider**

There is no service provider configuration.

#### **11.3.3 Running the Service Provider**

As this is IdP initiated SSO, you need to run the identity provider application rather than the service provider to initiate SSO.

In this example SSO is not being used. Instead, you should simply perform a local login at the service provider to ensure it is functioning correctly.

1. Browse to <http://localhost/SAML2SP>.
2. Login using the user name *sp-user* and a password of *password*.
3. Verify that you've been redirected to the service provider's default page.

### 11.3.4 Running the Service Provider in Visual Studio

You may run the identity provider in Visual Studio. No configuration changes are required.

### 11.4 IdP-Initiated SSO – Identity Provider

The SAML2IdP web application, in conjunction with the SAML2SP web application, demonstrates IdP initiated single sign-on.

#### 11.4.1 Installing the Identity Provider

1. Using Visual Studio, build and publish the web application.
2. Open the Internet Information Services management console.
3. Under the default web site for the local computer, create an application with an alias of SAML2IdP.
4. For the physical path, browse to the directory where SAML2IdP was built and published.
5. Ensure the web application has been successfully installed and configured by browsing to <http://localhost/SAML2IdP>.

#### 11.4.2 Configuring the Identity Provider

The identity provider configuration is contained within its web.config file's <appSettings> section.

The *AssertionConsumerServiceURL* specifies the URL of the service provider's assertion consumer service.

The *SPTargetURL* specifies the target URL of the service provider.

Modifying web.config does not require an application restart.

If you use the default installation you won't need to modify this configuration.

#### 11.4.3 Running the Identity Provider

In this example, the user is attempting to access a protected resource on the service provider and, rather than performing a local login at the service provider, SSO is initiated with a local login occurring at the identity provider and the asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the service provider.

1. Browse to <http://localhost/SAML2IdP>.
2. You should be presented with the identity provider login page as you will be logging in at the identity provider, not the service provider.

If you are not then you must already have logged in at the identity provider. To force a login, close the browser and start again.

3. Login using the user name *idp-user* and a password of *password*.

4. You should then be presented with the identity provider's default page.
5. Click the link to access the service provider.
6. You should then be presented with the service provider's default page.

This means you've successfully completed a SAML v2.0 SSO and are logged in at the service provider with your identity provider user name.

#### **11.4.4 Running the Identity Provider in Visual Studio**

You may run the service provider in Visual Studio. The one additional step is to note the port number being used by Visual Studio to run the application. You then need to update the identity provider's configuration as described in section 11.4.2 to account for the different port number being used by Visual Studio.

### **11.5 ADFS Interoperability – Service Provider**

The ADFSSP web application demonstrates SP initiated single sign-on with Windows Active Directory Federation Services (ADFS).

#### **11.5.1 Installing the Service Provider**

1. Using Visual Studio, build and publish the web application.
2. Open the Internet Information Services management console.
3. Under the default web site for the local computer, create an application with an alias of ADFSSP.
4. For the physical path, browse to the directory where ADFSSP was built and published.
5. Ensure the web application has been successfully installed and configured by browsing to <http://localhost/ADFSSP>.

#### **11.5.2 Configuring the Service Provider**

The service provider configuration is contained within its web.config file's <appSettings> section.

The *SingleSignOnServiceBinding* specifies the binding to use when communicating to the identity provider. The options are:

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

The *HttpPostSingleSignOnServiceURL* specifies the URL of the identity provider's single sign-on service when using the HTTP POST binding.

The *HttpRedirectSingleSignOnServiceURL* specifies the URL of the identity provider's single sign-on service when using the HTTP Redirect binding.

Modifying web.config does not require an application restart.

If you use the default installation you won't need to modify this configuration.

### 11.5.3 Miscellaneous Configuration

For the purposes of this example, the host name of the service provider is [www.sp.com](http://www.sp.com) and the host name of the ADFS identity provider is [www.idp.com](http://www.idp.com).

Update the Windows\System32\drivers\etc\hosts file on the identity provider and server provider to include entries for [www.idp.com](http://www.idp.com) and [www.sp.com](http://www.sp.com). For example:

```
192.168.1.20      www.idp.com
192.168.1.21      www.sp.com
```

### 11.5.4 Configuring ADFS

In the ADFS terminology, the service provider is a relying party. Using the ADFS management console, add a relying party trust for the service provider.

Select the option to enter the relying party information manually.

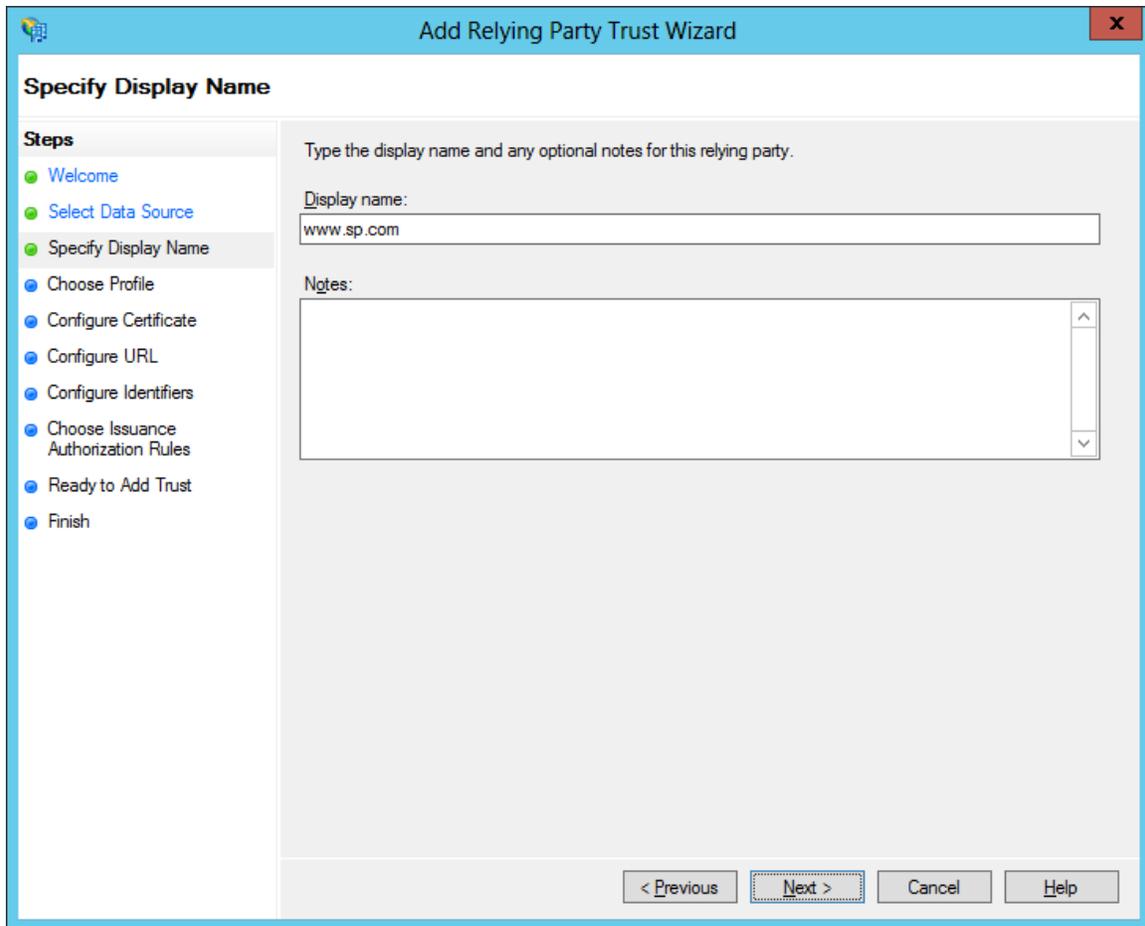
The screenshot shows the 'Add Relying Party Trust Wizard' dialog box. The title bar reads 'Add Relying Party Trust Wizard'. The main area is titled 'Select Data Source'. On the left, a 'Steps' pane lists the following steps: Welcome, Select Data Source (highlighted), Specify Display Name, Choose Profile, Configure Certificate, Configure URL, Configure Identifiers, Choose Issuance Authorization Rules, Ready to Add Trust, and Finish. The main content area contains the following text and options:

Select an option that this wizard will use to obtain data about this relying party:

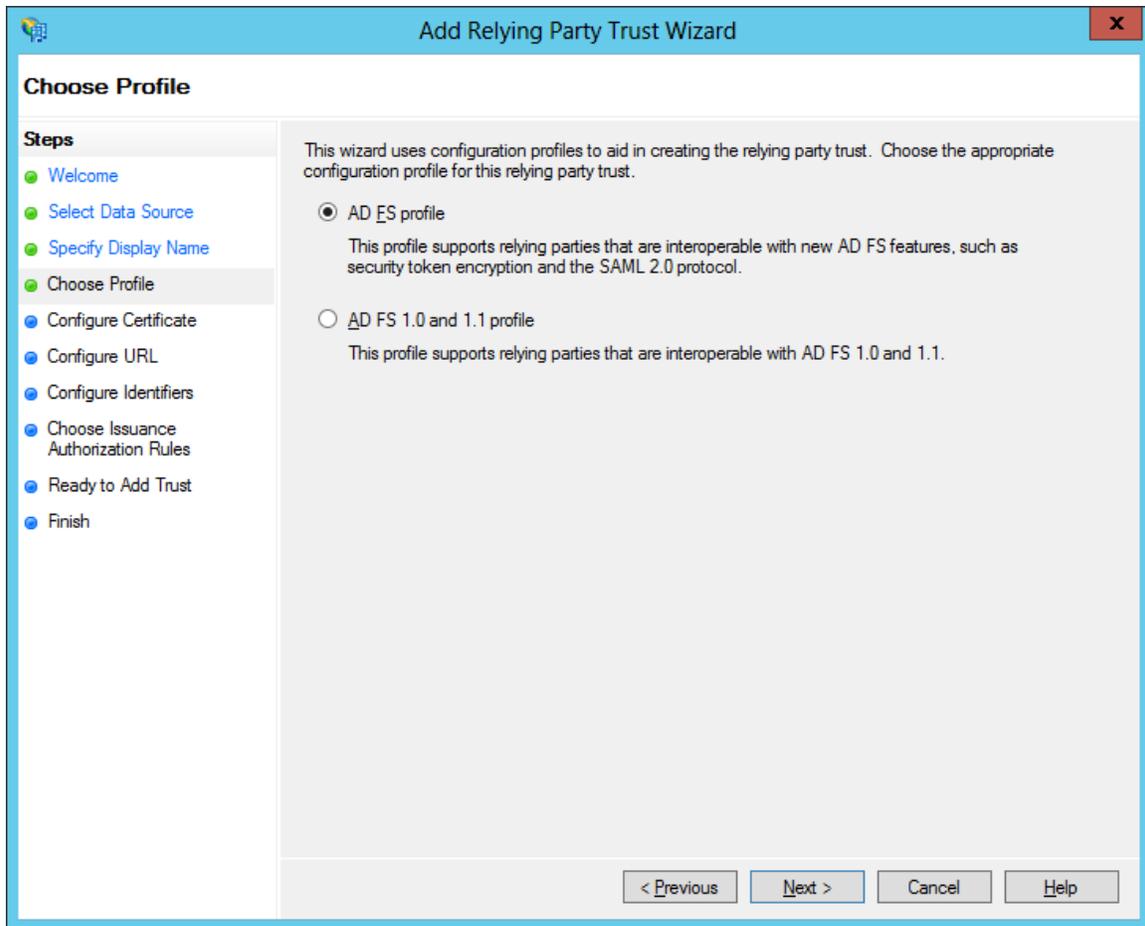
- Import data about the relying party published online or on a local network  
Use this option to import the necessary data and certificates from a relying party organization that publishes its federation metadata online or on a local network.  
Federation metadata address (host name or URL):  
  
Example: fs.contoso.com or https://www.contoso.com/app
- Import data about the relying party from a file  
Use this option to import the necessary data and certificates from a relying party organization that has exported its federation metadata to a file. Ensure that this file is from a trusted source. This wizard will not validate the source of the file.  
Federation metadata file location:
- Enter data about the relying party manually  
Use this option to manually input the necessary data about this relying party organization.

At the bottom of the dialog, there are four buttons: '< Previous', 'Next >', 'Cancel', and 'Help'.

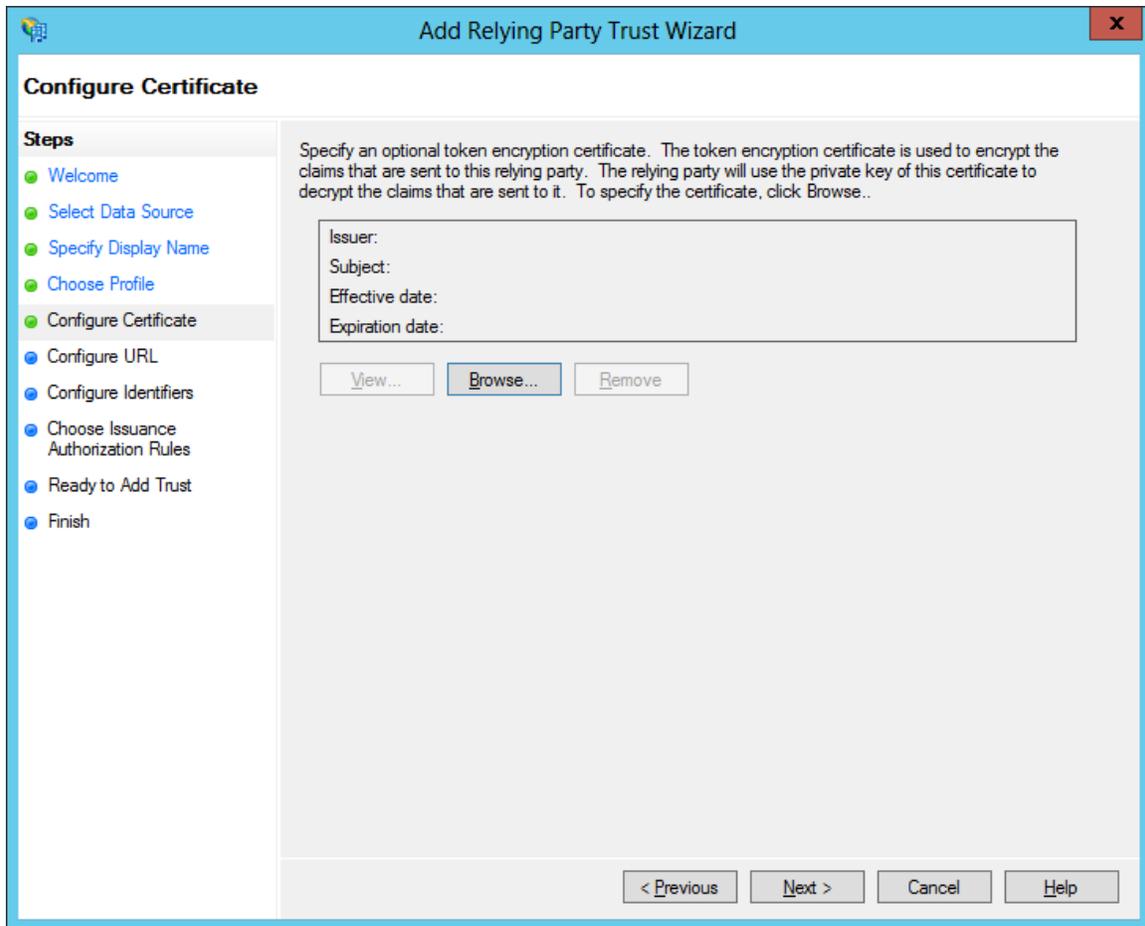
Specify a display name.



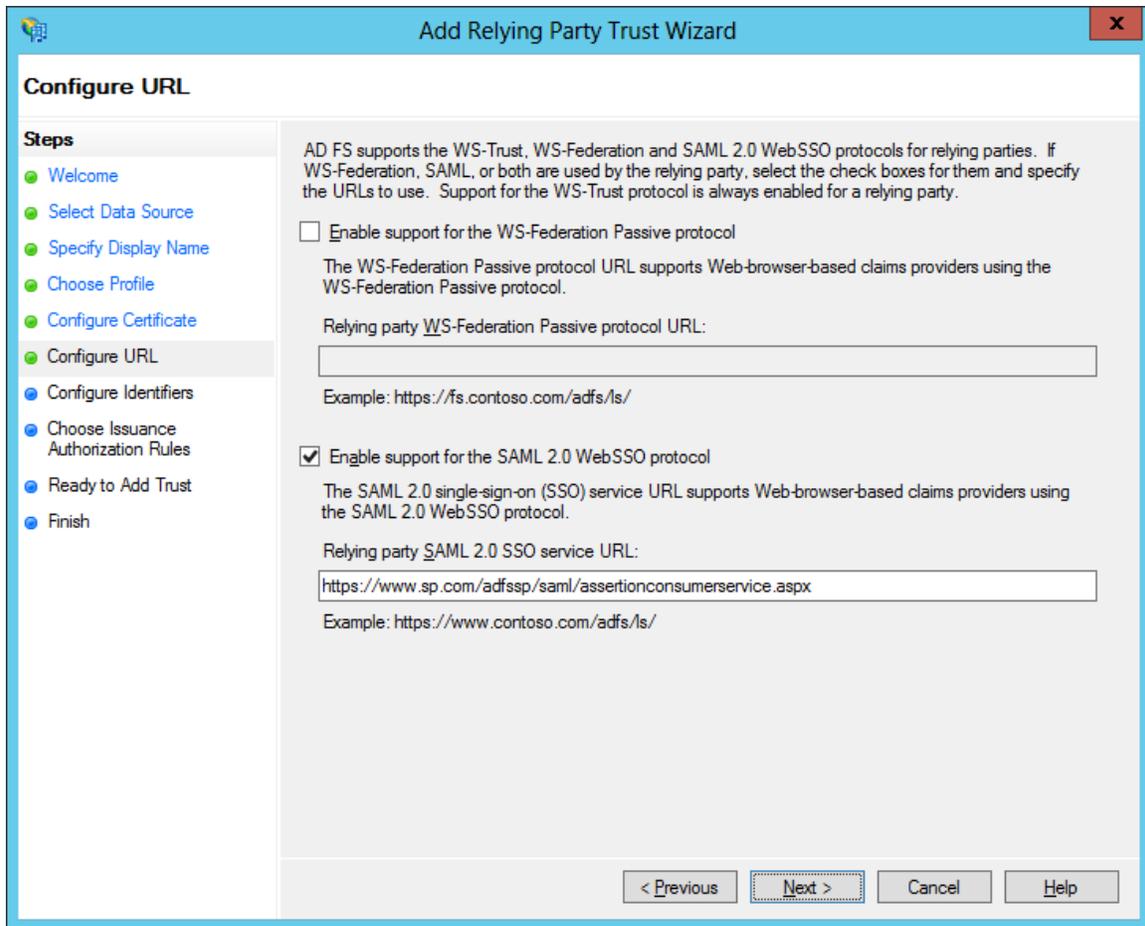
Choose the ADFS profile.



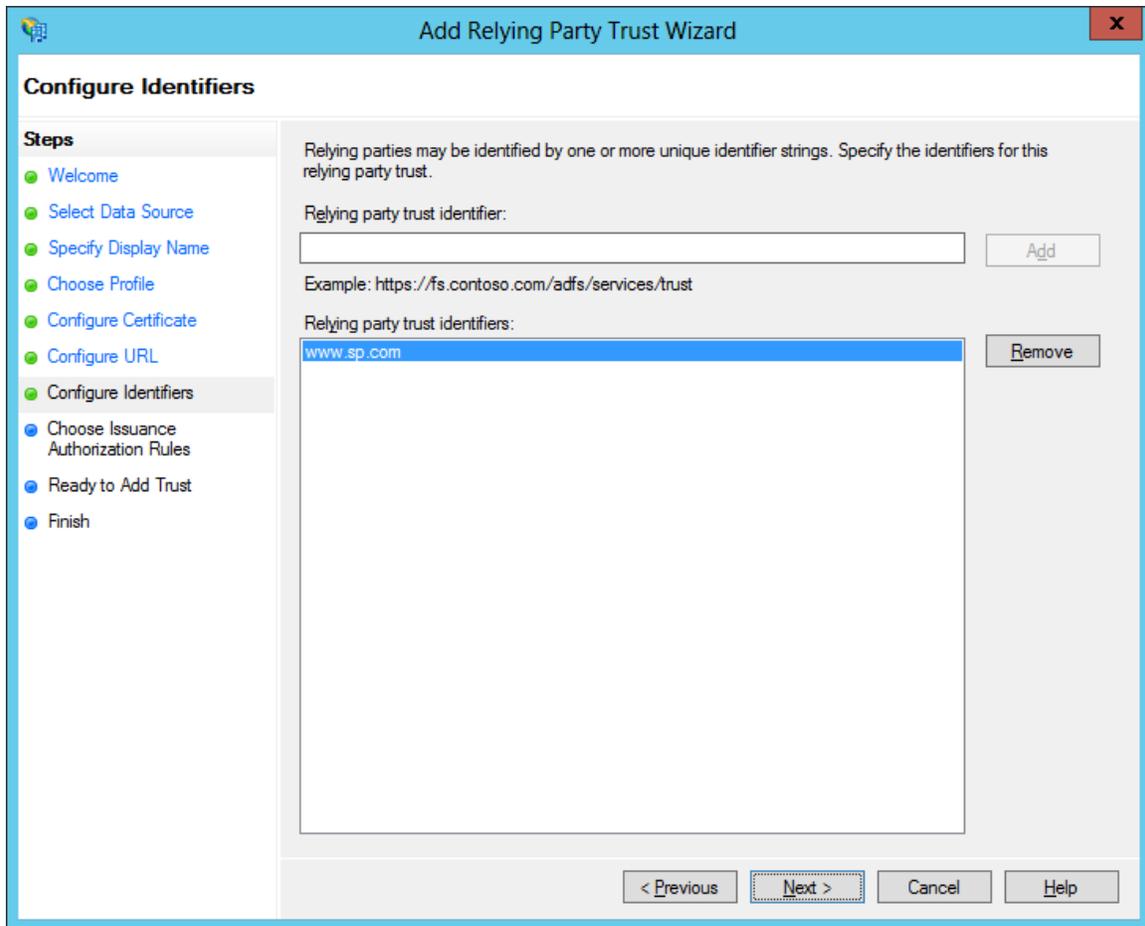
If you wish to have the SAML assertion returned by ADFS encrypted, browse to sp.cer to specify it as the token encryption certificate.



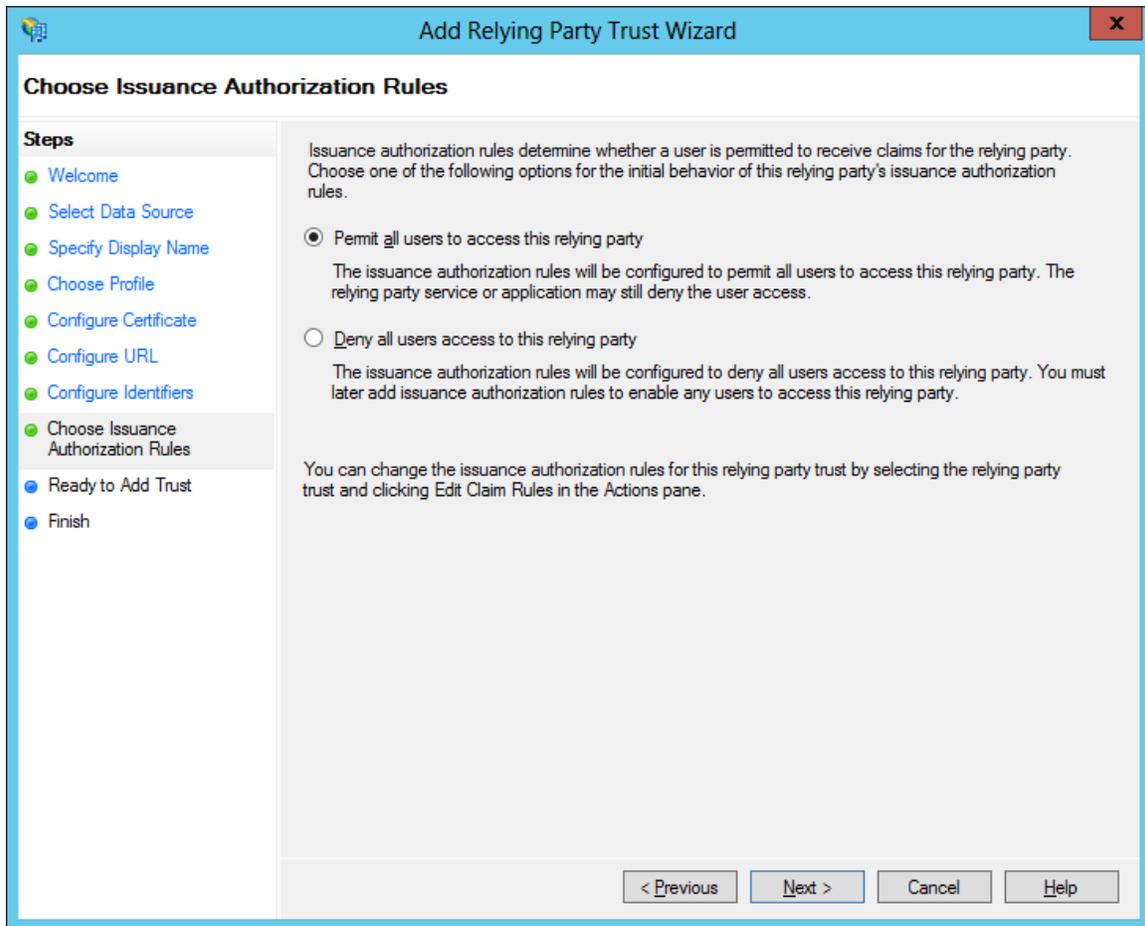
Enable support for SAML v2.0 and specify the service provider's assertion consumer service URL.



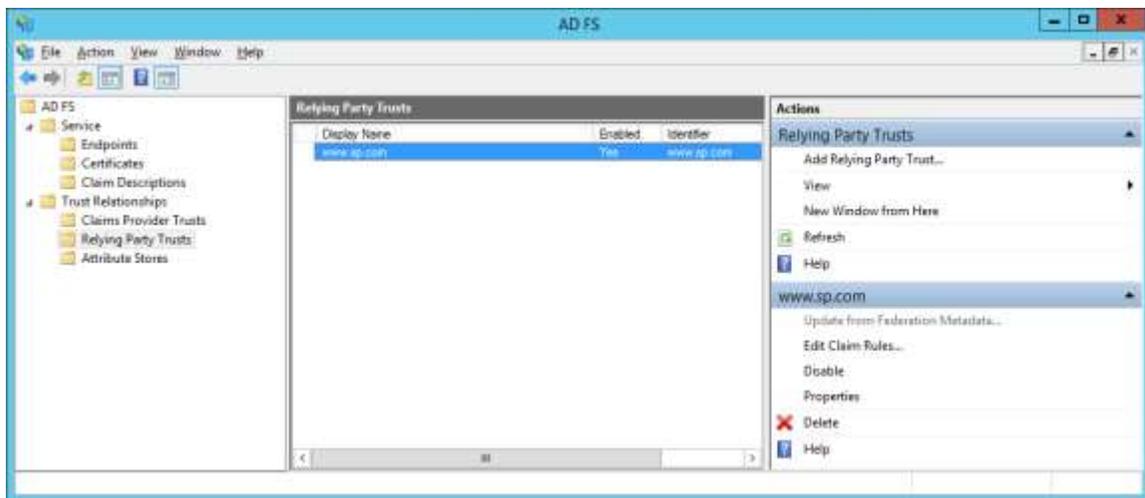
Specify the relying party trust identifier.



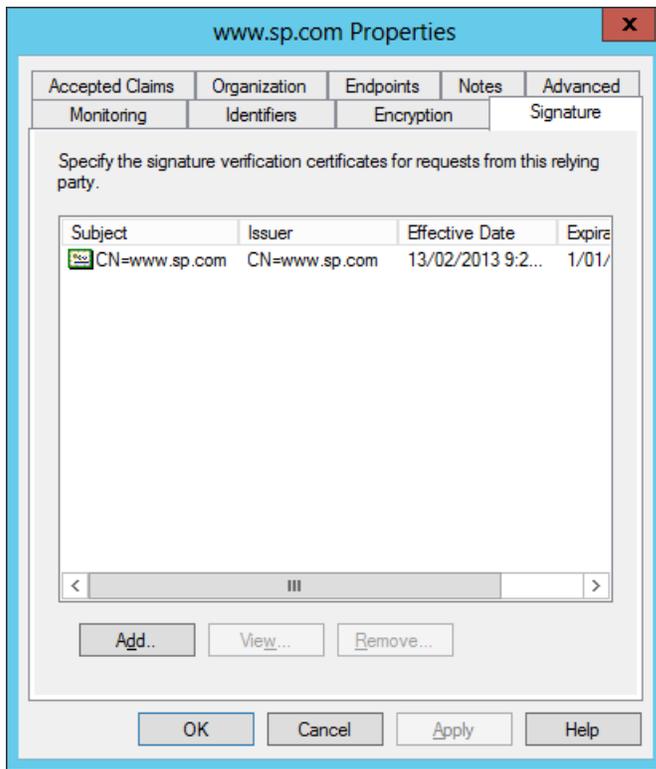
Permit all users access to this relying party.



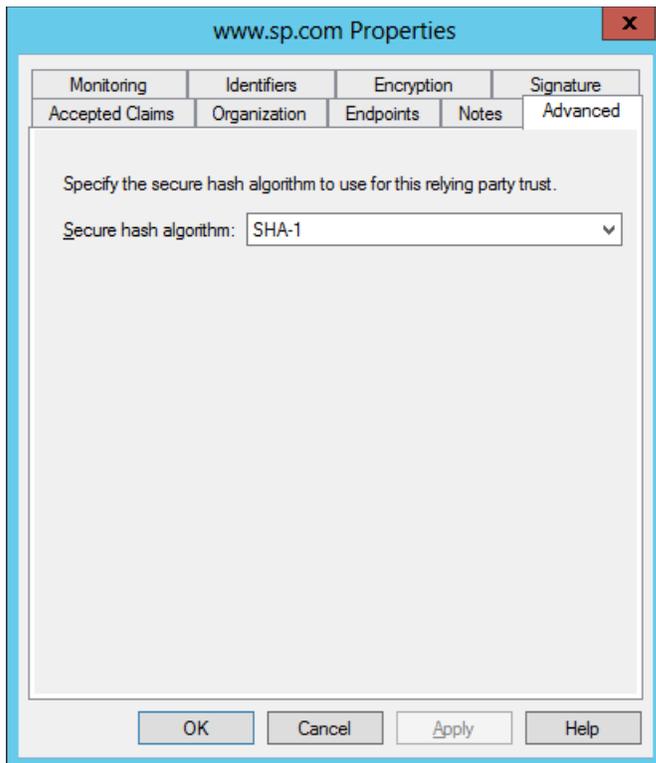
Review the configuration and click Finish. The service provider should be included in the list of relying party trusts.



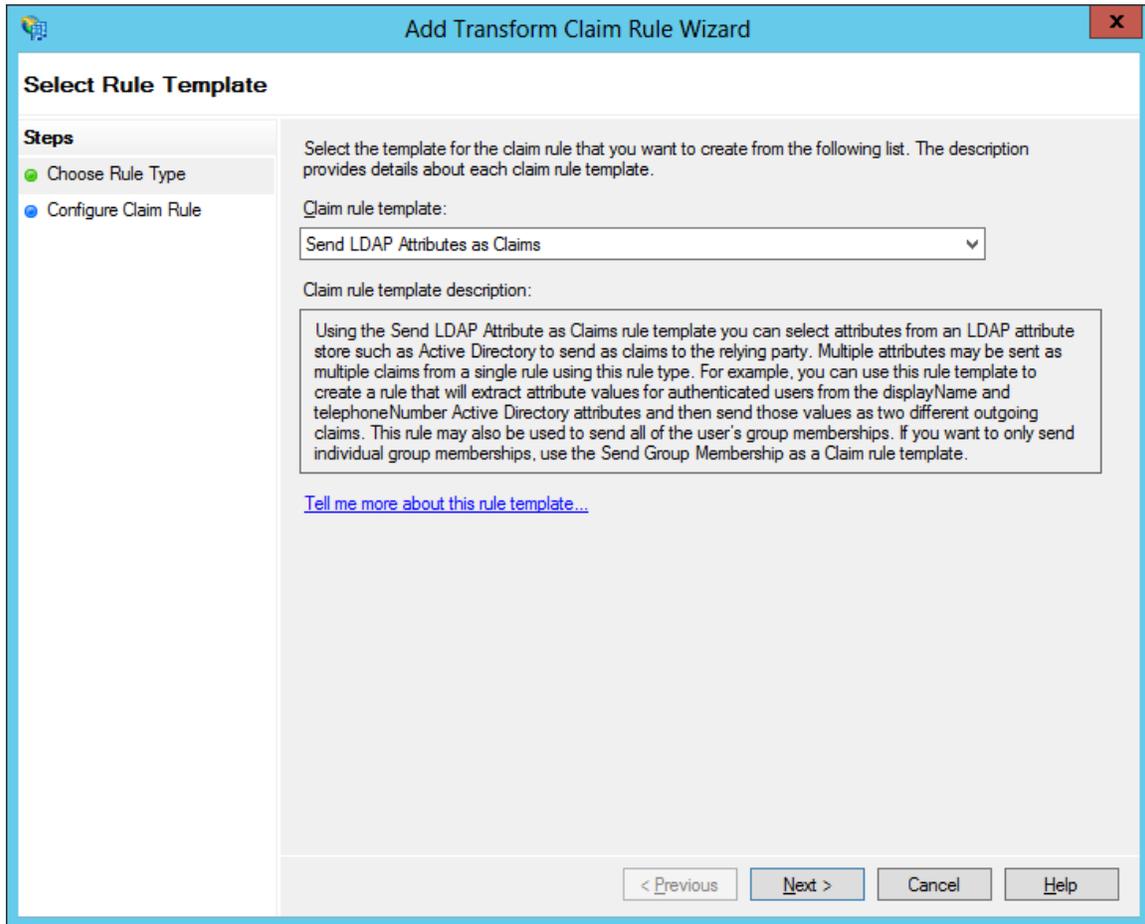
The authn request sent by the service provider is signed. To specify the certificate to use to verify the signature, bring up the reply party trusts properties and under the Signature tab add the service provider certificate.



Although the SAML v2.0 component supports SHA-256 signatures, for this example SHA-1 is used. To specify this, under the Advanced tab select SHA-1.



Edit the claim rules and add a rule.



Map the Active Directory user principal name to the outgoing Name ID.

**Add Transform Claim Rule Wizard**

---

**Configure Rule**

**Steps**

- Choose Rule Type
- **Configure Claim Rule**

You can configure this rule to send the values of LDAP attributes as claims. Select an attribute store from which to extract LDAP attributes. Specify how the attributes will map to the outgoing claim types that will be issued from the rule.

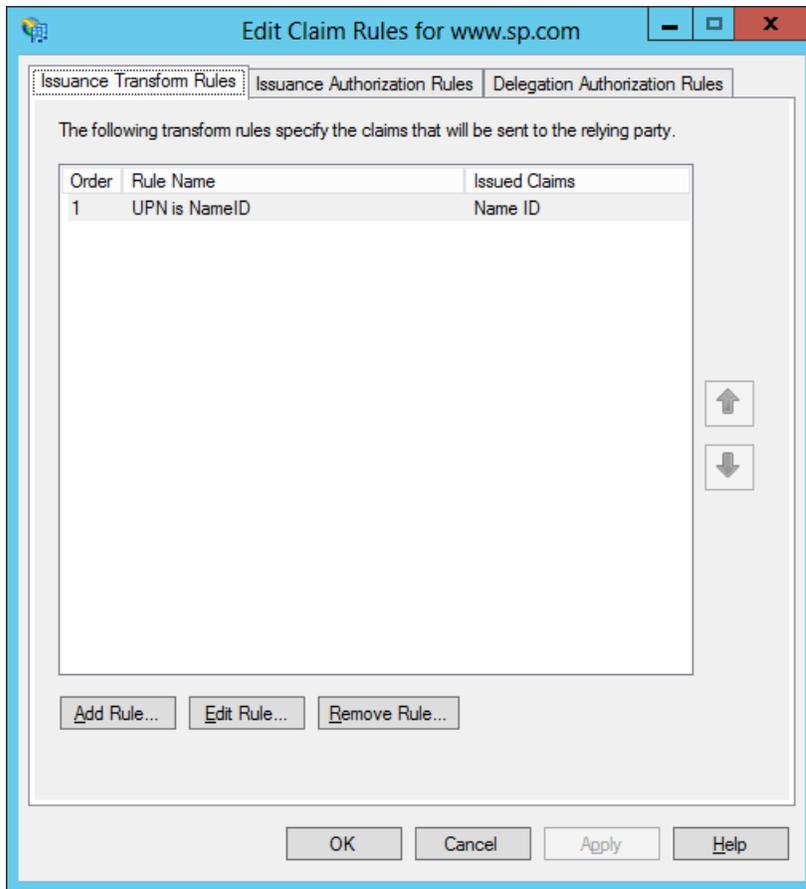
Claim rule name:

Rule template: Send LDAP Attributes as Claims

Attribute store:

Mapping of LDAP attributes to outgoing claim types:

	LDAP Attribute (Select or type to add more)	Outgoing Claim Type (Select or type to add more)
▶	<input type="text" value="User-Principal-Name"/>	<input type="text" value="Name ID"/>
*	<input type="text"/>	<input type="text"/>



ADFS should now be ready to communicate with the example service provider.

To review the metadata published by ADFS browse to:

<https://www.idp.com/FederationMetadata/2007-06/FederationMetadata.xml>

### 11.5.5 Running the Service Provider without SSO

In this example SSO is not being used. Instead, you should simply perform a local login at the service provider to ensure it is functioning correctly.

1. Browse to <https://www.sp.com/ADFSSP>, ignoring any browser certificate warnings.
2. Select the service provider as the location where login will occur.
3. Login using the user name *sp-user* and a password of *password*.
4. Verify that you've been redirected to the service provider's default page.

### 11.5.6 Running the Service Provider with SSO

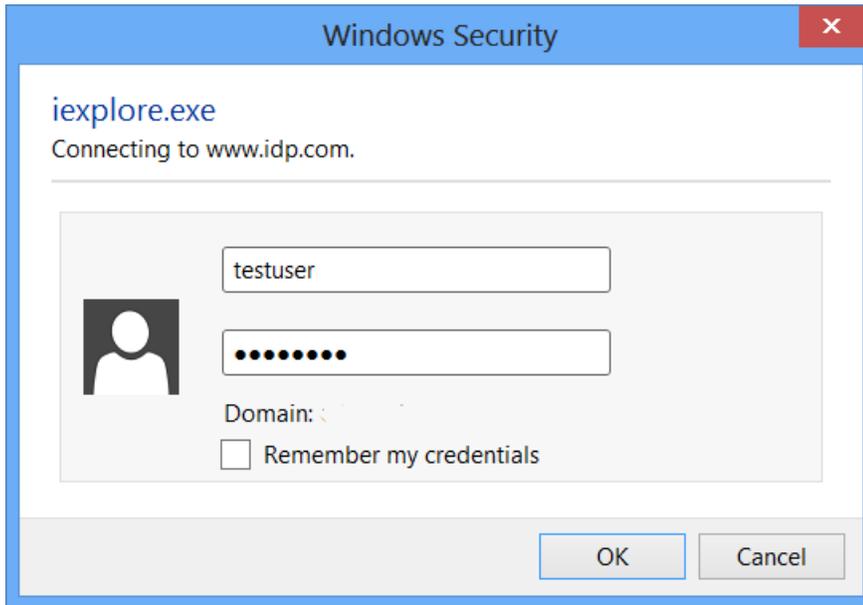
In this example, the user is attempting to access a protected resource on the service provider and, rather than performing a local login at the service provider, SSO is initiated with a local login occurring at the ADFS identity provider and the asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the service provider.

1. Browse to <http://www.sp.com/ADFSSP>, ignoring any browser certificate warnings.
2. Select the identity provider as the location where login will occur.

Selecting the identity provider will initiate a SAML v2.0 SSO to ADFS. Selecting the service provider will initiate a local login at the service provider.



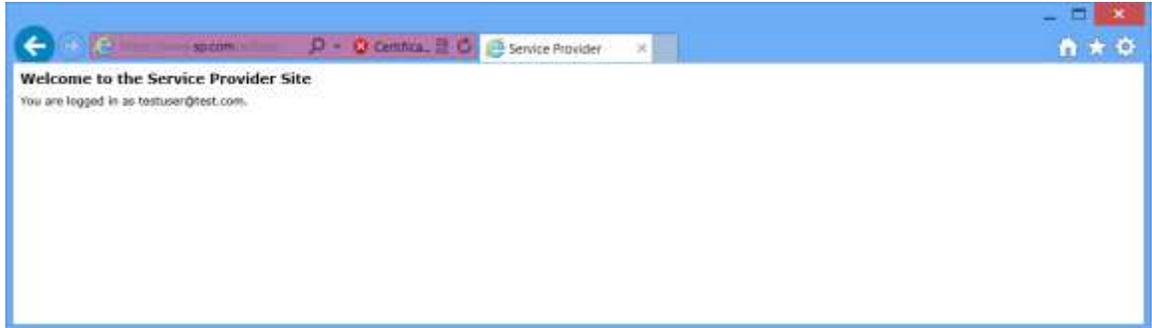
3. You should then be presented with the identity provider login prompt as you will be logging in at the identity provider, not the service provider.



4. Login using the user name and password of a user defined in Active Directory.

5. You should then be presented with the service provider's default page.

This means you've successfully completed a SAML v2.0 SSO and are logged in at the service provider with your identity provider user name.



## 11.6 Google Apps Interoperability – Identity Provider

The GoogleIdP web application demonstrates single sign-on with Google Apps. Google Apps is the service provider and GoogleIdP acts as the identity provider in an SP-initiated single sign-on. This allows single sign-on to Google web applications such as Gmail.

### 11.6.1 Installing the Identity Provider

1. Using Visual Studio, build and publish the web application.
2. Open the Internet Information Services management console.
3. Under the default web site for the local computer, create an application with an alias of GoogleIdP.
4. For the physical path, browse to the directory where GoogleIdP was built and published.
5. Ensure the web application has been successfully installed and configured by browsing to <http://localhost/GoogleIdP>.

### 11.6.2 Configuring the Identity Provider

The only identity provider configuration you may need to change is the list of user credentials specified in `web.config`. The user name that you log into at the identity provider must match with an account name in Google Apps.

Modifying `web.config` does not require an application restart.

### 11.6.3 Configuring Google Apps

Refer to the Google Apps documentation for instructions on enabling and configuring single sign-on.

When using the GoogleIdP application, the following URLs should be specified. Replace the example host name with the appropriate host name or IP address.

The sign-in page is: <http://www.yourdomain.com/GoogleIdP/SAML/SSOService.aspx>.

The sign-out page is: <http://www.yourdomain.com/GoogleIdP/SAML/Logout.aspx>.

The change password page is:

<http://www.yourdomain.com/GoogleIdP/ChangePassword.aspx>.

Upload the certificate contained in the idp.cer file in the project's directory.

### 11.6.4 Running Google Apps

1. Browse to a Google application (e.g. <http://mail.google.com/a/yourdomain.com>).
2. Login using the user name *google* and a password of *password*. This assumes there's a Google App user called *google*. You may have to modify the user credentials in the GoogleIdP web.config to include a Google App user name.
3. Verify that you've been redirected and signed into the Google app.
4. Sign out from the Google App.
5. Verify that you've been redirected to the GoogleIdP logout page.

## 11.7 Salesforce Interoperability – Identity Provider

The SalesforceIdP web application demonstrates single sign-on with Salesforce. Salesforce is the service provider and SalesforceIdP acts as the identity provider in an IdP-initiated single sign-on. This allows single sign-on to Salesforce.

### 11.7.1 Installing the Identity Provider

1. Using Visual Studio, build and publish the web application.
2. Open the Internet Information Services management console.
3. Under the default web site for the local computer, create an application with an alias of SalesforceIdP.
4. For the physical path, browse to the directory where SalesforceIdP was built and published.
5. Ensure the web application has been successfully installed and configured by browsing to <http://localhost/SalesforceIdP>.

### 11.7.2 Configuring the Identity Provider

The only identity provider configuration you may need to change is specified in web.config. The Issuer setting must match with the issuer specified in the Salesforce SSO configuration. The SalesforceLoginID is the Salesforce user name used to single sign-on to Salesforce.

Modifying web.config does not require an application restart.

### 11.7.3 Configuring Salesforce

Refer to the Salesforce documentation for instructions on enabling and configuring single sign-on.

The SAML version is 2.0. The issuer is [www.idp.com](http://www.idp.com). This must match the Issuer setting specified in web.config. Upload the certificate contained in the idp.cer file in the project's directory.

The user ID type should be set to specify that the assertion contains the user's Salesforce user name and the user ID is in the subject's name identifier. The Salesforce user name is configured with the SalesforceLoginID setting in web.config.

The Salesforce Login History under Manage Users provides a useful log for debugging single sign-on problems.

#### **11.7.4 Running Salesforce**

1. Browse to SalesforceIdP (e.g. <http://localhost/SalesforceIdP>).
2. Login using the user name *idp-user* and a password of *password*.
3. Click the link to navigate to Salesforce.

#### **11.7.5 Validating SAML Responses in Salesforce**

The Salesforce SAML Assertion Validator may be used to track down SAML assertion validation errors.

1. Capture and copy the SAML response XML making sure not to modify it in any way. For example, turn on trace within web.config and copy the SAML response XML from the generated log file.
2. Paste the SAML response XML into the Salesforce SAML Assertion Validator screen and click Validate.
3. Review the validation check list.

### ***11.8 Shibboleth Interoperability – Identity Provider***

The ShibbolethIdP web application, in conjunction with the ShibbolethSP web application, demonstrates SP initiated single sign-on.

These applications may also be used to demonstrate interoperability with Shibboleth. Shibboleth ([shibboleth.internet2.edu](http://shibboleth.internet2.edu)) is an open source SSO software package using Java and C++ technologies. Installation and configuration of the Shibboleth software is beyond the scope of this document and is not required for this demonstration.

#### **11.8.1 Installing the Identity Provider**

1. Using Visual Studio, build and publish the web application.
2. Open the Internet Information Services management console.
3. Under the default web site for the local computer, create an application with an alias of ShibbolethIdP.
4. For the physical path, browse to the directory where ShibbolethIdP was built and published.

5. Ensure the web application has been successfully installed and configured by browsing to <http://localhost/ShibbolethIdP>.

### 11.8.2 Configuring the Identity Provider

The identity provider configuration is contained within its web.config file's <appSettings> section.

The *AssertionConsumerServiceBinding* specifies the binding to use when communicating to the service provider. The options are:

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

The *HttpPostAssertionConsumerServiceURL* specifies the URL of the service provider's assertion consumer service when using the HTTP POST binding.

The *HttpArtifactAssertionConsumerServiceURL* specifies the URL of the service provider's assertion consumer service when using the HTTP Artifact binding.

The *ArtifactResolutionServiceURL* specifies the URL of the service provider's artifact resolution service.

Modifying web.config does not require an application restart.

If you use the default installation you won't need to modify this configuration.

### 11.8.3 Running the Identity Provider

As this is SP initiated SSO, you need to run the service provider application rather than the identity provider to initiate SSO.

In this example SSO is not being used. Instead, you should simply perform a local login at the identity provider to ensure it is functioning correctly.

1. Browse to <http://localhost/ShibbolethIdP>.
2. Login using the user name *idp-user* and a password of *password*.
3. Verify that you've been redirected to the identity provider's default page.

### 11.8.4 Running the Identity Provider in Visual Studio

You may run the identity provider in Visual Studio. The one additional step is to note the port number being used by Visual Studio to run the application. You then need to update the service provider's configuration as described in section 11.8.2 to account for the different port number being used by Visual Studio.

## 11.9 Shibboleth Interoperability – Service Provider

The ShibbolethSP web application, in conjunction with the ShibbolethIdP web application, demonstrates SP initiated single sign-on.

These applications may also be used to demonstrate interoperability with Shibboleth. Shibboleth ([shibboleth.internet2.edu](http://shibboleth.internet2.edu)) is an open source SSO software package using Java

and C++ technologies. Installation and configuration of the Shibboleth software is beyond the scope of this document and is not required for this demonstration.

### 11.9.1 Installing the Service Provider

6. Using Visual Studio, build and publish the web application.
7. Open the Internet Information Services management console.
8. Under the default web site for the local computer, create an application with an alias of ShibbolethSP.
9. For the physical path, browse to the directory where ShibbolethSP was built and published.
10. Ensure the web application has been successfully installed and configured by browsing to `http://localhost/ShibbolethSP`.

### 11.9.2 Configuring the Service Provider

The service provider configuration is contained within its `web.config` file's `<appSettings>` section.

The *SingleSignOnServiceBinding* specifies the binding to use when communicating to the identity provider. The options are:

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

The *HttpPostSingleSignOnServiceURL* specifies the URL of the identity provider's single sign-on service when using the HTTP POST binding.

The *HttpRedirectSingleSignOnServiceURL* specifies the URL of the identity provider's single sign-on service when using the HTTP Redirect binding.

The *HttpArtifactSingleSignOnServiceURL* specifies the URL of the identity provider's single sign-on service when using the HTTP Artifact binding.

The *ArtifactResolutionServiceURL* specifies the URL of the identity provider's artifact resolution service.

Modifying `web.config` does not require an application restart.

If you use the default installation you won't need to modify this configuration.

### 11.9.3 Running the Service Provider without SSO

In this example SSO is not being used. Instead, you should simply perform a local login at the service provider to ensure it is functioning correctly.

5. Browse to `http://localhost/ShibbolethSP`
6. Select the service provider as the location where login will occur.
7. Login using the user name *sp-user* and a password of *password*.

8. Verify that you've been redirected to the service provider's default page.

### 11.9.4 Running the Service Provider with SSO

In this example, the user is attempting to access a protected resource on the service provider and, rather than performing a local login at the service provider, SSO is initiated with a local login occurring at the identity provider and the asserted identity, passed to the service provider in a SAML assertion, is used to perform an automatic login at the service provider.

6. Browse to <http://localhost/ShibbolethSP>.
7. Select the identity provider as the location where login will occur.

Selecting the identity provider will initiate a SAML v2.0 SSO. Selecting the service provider will initiate a local login at the service provider.

8. You should then be presented with the identity provider login page as you will be logging in at the identity provider, not the service provider.

If you are not then you must already have logged in at the identity provider. To force a login, close the browser and start again.

9. Login using the user name *idp-user* and a password of *password*.
10. You should then be presented with the service provider's default page.

This means you've successfully completed a SAML v2.0 SSO and are logged in at the service provider with your identity provider user name.

### 11.9.5 Running the Service Provider in Visual Studio

You may run the service provider in Visual Studio. The one additional step is to note the port number being used by Visual Studio to run the application. You then need to update the identity provider's configuration as described in section 11.9.2 to account for the different port number being used by Visual Studio.

## 11.10 Assertion Examples

Assertion examples may be found under the examples directory (e.g. C:\Program Files (x86)\ComponentSpace\SAML2\Examples\Assertion).

### 11.10.1 SAML Assertion Example Application

The AssertionExample application demonstrates:

- Creating a SAML response message and accessing its contents
- Converting a SAML response message to and from XML
- Signing a SAML response message and verifying the signature
- Creating a SAML assertion and accessing its contents
- Converting a SAML assertion to and from XML
- Signing a SAML assertion and verifying the signature

- Encrypting and decrypting a SAML assertion
- Encrypting and decrypting a SAML attribute

### **11.11 Metadata Examples**

Metadata examples may be found under the examples directory (e.g. C:\Program Files (x86)\ComponentSpace\SAML2\Examples\Metadata).

See section 14 for SAML metadata templates and examples.

#### **11.11.1 Import Metadata Example Application**

The ImportMetadata application imports a SAML metadata file into the high-level API SAML configuration (saml.config).

Usage:

```
ImportMetadata.exe <metadata-filename>
```

where the file contains the SAML entities descriptor or entity descriptor metadata to be imported into saml.config.

For example, the following imports IdP metadata into saml.config:

```
ImportMetadata.exe idp-metadata.xml
```

The saml.config file, if any, is assumed to be in the current directory.

If it doesn't exist, a saml.config file is created. Otherwise, metadata is merged into the existing saml.config.

A saml.config partner provider entry is created for each entity descriptor in the metadata.

The updated saml.config includes "TODO" instructions where additional information is required or needs review.

#### **11.11.2 Export Metadata Example Application**

The ExportMetadata application exports the high-level API SAML configuration (saml.config) to SAML metadata.

Usage:

```
ExportMetadata.exe <partner-name> [<certificate-filename>] <metadata-filename>
```

where the partner name specifies the partner provider in the SAML configuration, the certificate file contains the local provider's X.509 certificate, and a metadata file containing the SAML entity descriptor metadata is created.

For example, the following exports saml.config to a metadata file:

```
ExportMetadata.exe urn:componentspace:ExampleIdentityProvider sp.cer sp-metadata.xml
```

The saml.config file is assumed to be in the current directory.

A single metadata entity descriptor is created for the local provider configured in `saml.config`.

The generated metadata includes “TODO” instructions where additional information is required or needs review.

### 11.11.3 SAML Metadata Example Application

The `MetadataExample` application demonstrates creating and manipulating SAML metadata including:

- Creating and reading an IdP entity descriptor
- Creating and reading an SP entity descriptor

### 11.11.4 ReadMetadata

`ReadMetadata` reads SAML metadata. It is a useful utility for confirming the metadata syntax is valid.

Usage:

```
ReadMetadata.exe <filename>
```

where the file contains the SAML entities descriptor or entity descriptor metadata.

For example, the following parses IdP metadata:

```
ReadMetadata.exe idp-metadata.xml
```

## 11.12 Signature Examples

XML signature examples may be found under the examples directory (e.g. `C:\Program Files (x86)\ComponentSpace\SAML2\Examples\Signature`).

### 11.12.1 SHA-256 Signature Example Application

The `SHA256Signature` application demonstrates generating and verifying XML signatures using the SHA-256, SHA-384 and SHA-512 digest and signature algorithms.

SHA-256, SHA-384 and SHA-512 XML signature support requires a .NET CLR security update or the use of .NET 4.5 or above. Refer to section 16.3 for more details.

### 11.12.2 SignSAML

`SignSAML` demonstrates signing SAML assertions and protocol messages. It is also a useful utility for debugging signature verification errors.

Usage:

```
SignSaml.exe -k <keystore> -p <password> <filename>
```

where the keystore is a PFX file containing a key, the password is the password to the keystore, and the file contains the XML to be signed.

For example, the following signs a SAML response:

```
SignSaml.exe -k test.pfx -p password SAMLResponse.xml
```

### **11.12.3 VerifySAML**

VerifySAML demonstrates verifying signatures on SAML assertions and protocol messages. It is also a useful utility for debugging signature verification errors.

Usage:

```
VerifySaml.exe [-c <certificateFileName>] <filename>
```

where the certificateFileName is a CER file containing the certificate to use to verify the signature, and the file contains the signed XML to be verified.

If no certificateFileName is specified then the certificate contained in the signed XML is used.

For example, the following verifies a SAML response:

```
VerifySaml.exe -c test.cer SAMLResponse.xml
```

## **11.13 Utility Applications**

Utility applications may be found under the examples directory (e.g. C:\Program Files (x86)\ComponentSpace\SAML2\Examples\Utility).

### **11.13.1 ValidateConfig**

ValidateConfig validates the SAML configuration file against its schema. See section 6 for a description of the SAML configuration schema.

ValidateConfig is a useful utility for debugging SAML configuration file errors.

Usage:

```
ValidateConfig.exe <filename >
```

where the file contains the SAML configuration.

For example, the following validates a SAML response:

```
ValidateConfig.exe saml.config
```

### **11.13.2 ValidateXML**

ValidateXML demonstrates validating SAML assertions, protocol messages and metadata against the SAML, XML signature and XML encryption schemas. It is also a useful utility for debugging invalid SAML XML errors.

Usage:

```
ValidateXml.exe <filename>
```

where the file contains the SAML assertion, protocol or metadata XML to be validated.

For example, the following validates a SAML response:

```
ValidateXml.exe SAMLResponse.xml
```

### 11.13.3 EncryptSAML

EncryptSAML demonstrates encrypting SAML assertions. It is also a useful utility for debugging encryption errors.

Usage:

```
EncryptSaml.exe -c <certificateFileName> <filename>
```

where the certificateFileName is a CER file containing the certificate to use to encrypt the assertion, and the file contains the SAML assertion XML to be encrypted.

For example, the following encrypts a SAML assertion:

```
EncryptSaml.exe -c test.cer SAMLAssertion.xml
```

### 11.13.4 DecryptSAML

DecryptSAML demonstrates decrypting encrypted SAML assertions. It is also a useful utility for debugging decryption errors.

Usage:

```
DecryptSaml.exe -k <keystore> -p <password> <filename>
```

where the keystore is a PFX file containing a key, the password is the password to the keystore, and the file contains the encrypted SAML assertion XML to decrypt.

For example, the following decrypts an encrypted SAML assertion:

```
DecryptSaml.exe -k test.pfx -p password EncryptedAssertion.xml
```

### 11.13.5 ParseHttpRequestUrl

ParseHttpRequestUrl decodes the query string parameters in an HTTP redirect URL and verifies their signature. It is a useful utility for debugging signature errors.

Usage:

```
ParseHttpRequestUrl.exe -c <certificateFileName> <filename>
```

where the certificateFileName is a CER file containing the certificate to use to verify the signature, and the file contains the redirect URL including query string.

For example, the following parses a redirect URL:

```
ParseHttpRequestUrl.exe -c test.cer RedirectURL.txt
```

### 11.13.6 Java Utilities

A Java application may be found under the examples directory (e.g. C:\Program Files (x86)\ComponentSpace\SAML2\Examples\Java).

This application may be used to independently generate and verify signatures.

Refer to the readme.txt in the Examples\Java directory for instructions on running this application.

## 12 Creating your own SSO Application

The following steps describe the process for enabling SAML single sign-on in your application. Refer to section 3 if you are not familiar with SAML single sign-on.

1. Determine whether your application will be an identity provider (IdP) or service provider (SP).
2. Determine whether your application will support IdP-initiated SSO and/or SP-initiated SSO.
3. Exchange SAML configuration information (e.g. X.509 certificates, URLs) with the partner organization or site (see section 14).
4. Add a reference to the SAML class library in your application (see section 4.1).
5. Add endpoint pages to your application to receive SAML protocols messages, if required.
6. Call into the SAML class library API to enable SAML single sign-on.

The example applications described in section 5 are a good starting point for understanding the SAML class library API.

Refer to the class library reference for help using the API (see section 21).

7. Once completed, distribute the SAML class library DLL with your application (see section 4.2).

### 12.1 Considerations

The example applications described in section 5 are a good starting point for adding SSO support to your application.

However, these are only example applications and have been kept as simple as possible to assist you in understanding them.

For production applications you will need to consider a number of requirements not covered by the example applications.

#### 12.1.1 Error Handling

The example applications include minimal error handling.

The example applications display the specific error on the browser.

In a production environment you should not display specific error information but instead should log this information and display a generic error message or error page.

#### 12.1.2 Configuration

Some of the example applications allow the user to select the SAML bindings.

The bindings to use should be negotiated between the identity provider and service provider and not exposed to the user.

### 12.1.3 Key Management

SAML assertions and protocol message may be signed to ensure their integrity and origin.

Also, SAML assertions may be encrypted to ensure the privacy of their data.

The example applications sign messages using private keys stored in PFX files and verify signatures using public keys stored in CER files.

You may wish to store keys and certificates in one of the Windows certificate stores and access them using the .NET framework's X509Store class.

### 12.1.4 Security

Some of the example applications do not use HTTPS.

You should follow the security recommendations described in the SAML specification.

Typically this means using HTTPS.

## 13 Test Certificates and Keys

Test certificate and key files are supplied that may also be used during the development of your identity provider or service provider applications.

For example, the SP initiated identity provider includes an idp.pfx and sp.cer in its root directory. The SP initiated service provider includes an sp.pfx and idp.cer in its root directory. The password for these PFX files is *password*.

The identity provider uses the secret key stored in idp.pfx to sign messages. The service provider uses the public key contained in idp.cer to verify signatures in messages received from the identity provider.

Similarly, the service provider uses the secret key stored in sp.pfx to sign messages. The identity provider uses the public key contained in sp.cer to verify signatures in messages received from the service provider.

Certificates may also be embedded in the XML signature that's included with the signed message. These certificates may be used rather than separately stored certificates although you need to consider any security ramifications.

The following sections outline a number of alternatives for generating your own test certificates and keys.

Test certificate and keys should not be used in a production environment. You should purchase these from a certificate issuing service. Standard SSL certificates may be used.

### 13.1 Makecert

Use the makecert and pvk2pfx tools that ships with Visual Studio (e.g. C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\Bin\makecert.exe) to create certificate files and private key files.

```
makecert -r -pe -sky exchange -n "cn=www.idp.com" -sv idp.pvk idp.cer
```

You then need to convert the PVK file to a PFX file so it can be loaded with the .NET framework classes.

```
pvk2pfx -pvk idp.pvk -spc idp.cer -pfx idp.pfx -po password -f
```

Refer to the Microsoft help for additional options.

### **13.1.1 Makecert and SHA-256 XML Signatures**

When using makecert to create a self-signed certificate to generate SHA-256, SHA-384 and SHA-512 XML signatures (see section 16.3), the correct cryptographic provider type must be specified.

The Microsoft Enhanced RSA and AES Cryptographic Provider is required to support SHA-256, SHA-384 and SHA-512 signatures. This provider's type is twenty-four.

For example:

```
makecert -r -pe -sky exchange -n "cn=www.idp.com" -ss My -sy 24
```

Rather than saving the certificate and private key to files, they're saved to the Windows certificate store. This is required to work around an issue in makecert or pvk2pfx where the provider type information is lost if the certificate and private key are directly saved to files.

The Microsoft Management Console's Windows Certificate snap-in should be used to export the certificate and private key to a PFX file.

The default 1024 bit key length may be used for SHA-256, SHA-384 and SHA-512 XML signature generation.

If you wish to create a 2048 bit key, specify the length parameter.

For example:

```
makecert -r -pe -sky exchange -n "cn=www.idp.com" -ss My -sy 24 -len 2048
```

The default signature algorithm used to sign the certificate using the issuer's private key is SHA-1. This is independent of the certificate's support for SHA-256, SHA-384 and SHA-512 XML signatures. For example, a SHA-1 signed certificate may be used to generate SHA-256 XML signatures.

If you wish to sign the certificate using SHA-256, specify the algorithm parameter.

For example:

```
makecert -r -pe -sky exchange -n "cn=www.idp.com" -ss My -sy 24 -len 2048 -a sha256
```

## **13.2 Microsoft Certificate Server**

1. If not already done, install the Windows 2003 Certificate Services Windows component. This installs a certification authority (CA) to issue certificates.
2. Navigate to the certificate service (e.g. <http://localhost/certsrv>) and request a certificate. Select the "advanced certificate request" and then "Create and submit a request to this CA". Fill in the certificate request details, specifying the certificate

- type as server authentication certificate and make sure "Mark keys as exportable" is checked.
3. Using the Certification Authority MMC snap-in, view the pending requests and issue a certificate.
  4. Back at the certificate service click, view the status of the pending certificate request and click the link to install the certificate.
  5. Using the Certificates MMC snap-in, view the certificate to confirm that it has been installed. If required you can export the certificate and private key to a PFX file but make sure to check "Include all certificates in the certification path if possible". You can also export the certificate only if required.

### **13.3 Keytool**

Use the Java keytool that comes with the JDK to create certificate files and private key files.

```
keytool -genkey -dname "cn=www.idp.com" -alias idp -keypass password -keyalg RSA -validity 3650 -keystore idp.pfx -storepass password -storetype pkcs12
```

You then need to generate a certificate file.

```
keytool.exe -export -alias idp -keystore idp.pfx -storepass password -storetype pkcs12 -rfc -file idp.cer
```

## **14 SAML Metadata**

The "Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0" specification defines a format for exchanging SAML configuration information. This exchange occurs out-of-band (e.g. by downloading from web sites or by email) between partner organizations as part of establishing a single sign-on environment.

The use of SAML metadata is entirely optional. Information, including endpoint URLs etc, may be exchanged in any manner convenient to the partner organizations.

### **14.1 Metadata Production**

The SAML library supports the generation of SAML metadata programmatically. Section 11.11 outlines example projects demonstrating how to generate identity provider and service provider metadata.

Alternatively, metadata may be created using your preferred XML editor and some knowledge of the SAML metadata XML schema. Templates and examples are included in C:\Program Files (x86)\ComponentSpace SAML v2.0 for .NET\Examples\Metadata\Templates.

The IdP-template.xml is a suitable starting point for creating identity provider metadata and includes comments outlining what has to be edited. The IdP-example.xml is an example of identity provider metadata.

The SP-template.xml is a suitable starting point for creating service provider metadata and includes comments outlining what has to be edited. The SP-example.xml is an example of identity provider metadata.

## 14.2 Metadata Consumption

The SAML library supports the consumption of SAML metadata programmatically. Section 11.11 outlines example projects demonstrating how to read identity provider and service provider metadata.

Alternatively, metadata may be read using your preferred XML editor and with some knowledge of the SAML metadata XML schema. Information including URLs etc may then be extracted and included with your application's configuration.

## 14.3 Importing and Exporting Metadata

Section 11.11 describes example applications for importing and exporting SAML metadata to and from SAML configuration (saml.config).

# 15 Troubleshooting

## 15.1 Tracing

To help resolve problems, tracing internal to the product may be enabled. If you are experiencing a problem you may be asked to enable tracing.

### 15.1.1 Diagnostic Tracing in Web Applications

To enable diagnostic tracing, update your application's web.config to include a <system.diagnostics> section as shown in the example configuration below.

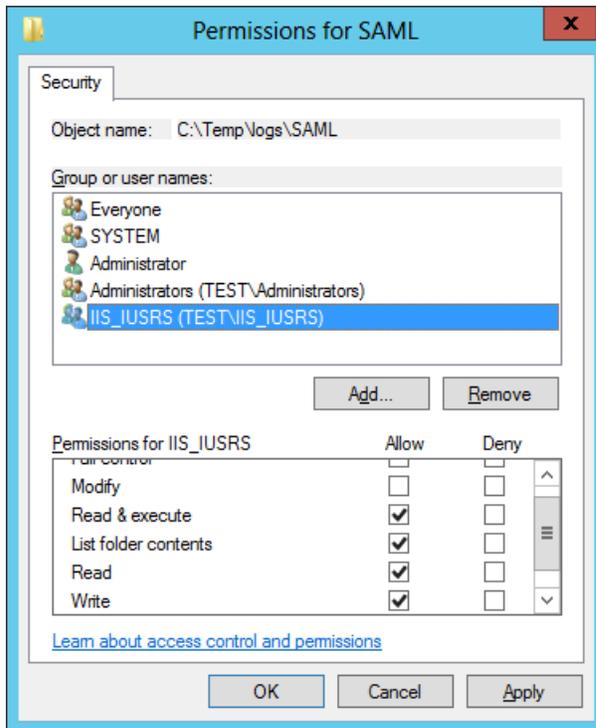
```
<system.diagnostics>
  <trace autoflush="true">
    <listeners>
      <add name="TextWriter"/>
    </listeners>
  </trace>
  <sources>
    <source name="ComponentSpace.SAML2" switchValue="Verbose">
      <listeners>
        <add name="TextWriter"/>
      </listeners>
    </source>
  </sources>
  <sharedListeners>
    <add name="TextWriter"
      type="System.Diagnostics.TextWriterTraceListener"
      initializeData="c:\temp\logs\saml\idp.log"/>
  </sharedListeners>
</system.diagnostics>
```

Most of the example applications include a diagnostic tracing section in their web.config files.

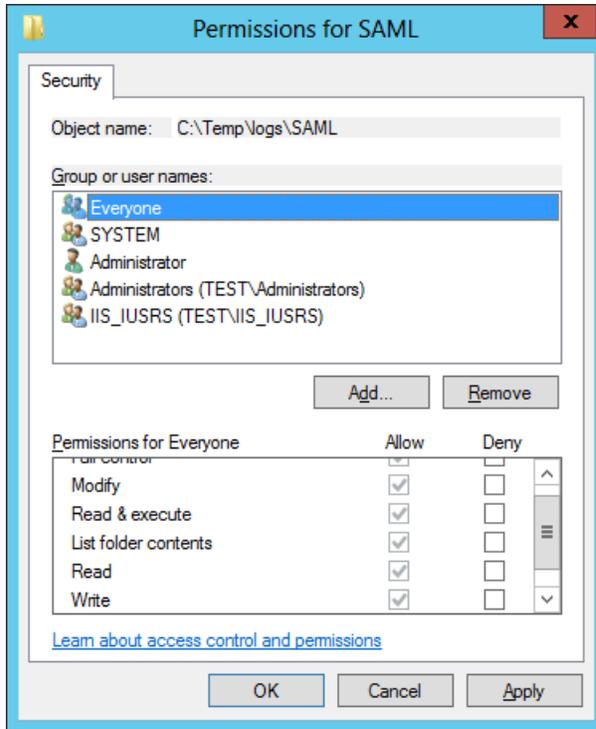
### 15.1.1.1 Setting Log File Permissions

You must ensure the directory where the log file will be written exists. In this example, the directory `c:\temp` must exist in order for the `SAML2.log` file to be created.

The user account running the web application must have write permission to this directory. For example, give the `IIS_USERS` group write permission to the directory.



Alternatively, in a development environment you may give the `Everyone` group write permission to the directory. This should not be done in a production environment.



### 15.1.2 Diagnostic Tracing in Non-Web Applications

Create a standard .NET configuration file for your application, if one doesn't already exist. The configuration file name consists of your application file name and `.config`. For example, if your application is called `myapplication.exe` then its configuration file should be named `myapplication.exe.config`. The configuration file must be located in the same directory as the application executable.

Include in the `<system.diagnostics>` section a switch and listeners as shown in the example configuration below.

```
<system.diagnostics>
  <trace autoflush="true">
    <listeners>
      <add name="TextWriter"/>
    </listeners>
  </trace>
  <sources>
    <source name="ComponentSpace.SAML2" switchValue="Verbose">
      <listeners>
        <add name="TextWriter"/>
      </listeners>
    </source>
  </sources>
  <sharedListeners>
    <add name="TextWriter"
      type="System.Diagnostics.TextWriterTraceListener"
      initializeData="c:\temp\logs\idp.log"/>
  </sharedListeners>
</system.diagnostics>
```

```
</sharedListeners>
</system.diagnostics>
```

## 15.2 Troubleshooting XML Signatures

SAML assertions and protocol messages may include an XML signature. If a signature fails to verify then either the signed XML has been altered in some way or the wrong certificate has been used to perform the verification.

One common problem when manipulating signed XML is to be careful to preserve white-space characters as these characters are significant when generating signatures. Internally, the SAML v2.0 class library ensures white-space is preserved. If you load or manipulate the signed XML ensure the XML is not modified prior to verifying its signature and specifically ensure white-spaces are preserved.

If signature verification is failing ensure the correct certificate is being used. The certificate is either contained in the XML signature or is loaded from a certificate file or store. If you are using a separately loaded certificate rather than a certificate contained within the XML signature, ensure the certificate is the correct one.

### 15.2.1 VerifySAML

If you believe the correct certificate is being used you can run a supplied utility, VerifySAML, to see whether the signature can be verified (see section 11.11.4).

For example, to verify the signature on a SAML protocol response you would capture the response to file, ensuring the XML is not altered in any way, and run:

```
VerifySaml.exe [-c <certificateFileName>] <filename>
```

The certificateFileName is a CER file containing the certificate to use to verify the signature. Only specify this parameter if the certificate is being loaded from a certificate file or store. If the certificate is included in the XML signature, then do not specify this parameter.

The filename is the file containing the SAML protocol response as XML.

This utility may be used to verify signatures on SAML requests, responses and assertions.

### 15.2.2 VerifySAML Log File

The VerifySAML utility generates a VerifySAML.log0020file in the working directory. This includes log entries generated by the .NET framework during signature verification.

If signature verification is successful, the log will include entries like:

```
System.Security.Cryptography.Xml.SignedXml Verbose: 13 : [SignedMessage#00245fb7,
VerifyReference] Reference Reference#003917f2 hashed with
"http://www.w3.org/2000/09/xmlsig#sha1" (SHA1CryptoServiceProvider) has hash value
a3503180ce819de2efc3a66f9b29b7d2687033ec, expected hash value
a3503180ce819de2efc3a66f9b29b7d2687033ec.
```

System.Security.Cryptography.Xml.SignedXml Information: 9 : [SignedMessage#00245fb7, SignatureVerificationResult] Verification with key RSACryptoServiceProvider#039490e2 was successful

This shows that the calculated hash of the canonicalized XML matches the hash or digest value contained in the XML signature. This confirms that the XML has not been modified.

It also shows that the calculated signature value matches the signature value in the XML signature. This confirms that the XML was signed by the owner or subject of the certificate.

If the XML has been modified after signing, the log will include entries like:

```
System.Security.Cryptography.Xml.SignedXml Verbose: 13 : [SignedMessage#00245fb7, VerifyReference] Reference Reference#003917f2 hashed with "http://www.w3.org/2000/09/xmldsig#sha1" (SHA1CryptoServiceProvider) has hash value 604bfa74922eb89c25d061a0e9eda3d3f1967d9c, expected hash value a3503180ce819de2efc3a66f9b29b7d2687033ec.
System.Security.Cryptography.Xml.SignedXml Information: 12 : [SignedMessage#00245fb7, VerificationFailure] Verification failed checking references.
```

This shows that the calculated hash of the canonicalized XML does not match the hash or digest value contained in the XML signature. This means that the XML has been modified.

If the wrong certificate is used to verify the signature, the log will include entries like:

```
System.Security.Cryptography.Xml.SignedXml Information: 12 : [SignedMessage#00245fb7, VerificationFailure] Verification failed checking SignedInfo.
```

### 15.2.3 Java VerifyXMLSignature

A Java application is supplied for independently checking XML signatures. This application may be run using the provided VerifyXMLSignature.bat script file.

The VerifyXMLSignature.bat uses Java 1.6 or better to verify XML signatures.

### 15.2.4 XML Signatures and Prefixes

Consider the following section of XML:

```
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol">
```

The element name is Response; it's prefixed with samlp; and it's declared in the urn:oasis:names:tc:SAML:2.0:protocol namespace.

What uniquely identifies this element is its name and namespace. The prefix is a mechanism for linking an element with its namespace declaration.

By conventions samlp is used for the urn:oasis:names:tc:SAML:2.0:protocol namespace although any other prefix would be equally valid.

Java and some other implementations often use the ds prefix for the XML signature namespace. For example:

```
<ds:Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
```

The .NET framework treats the XML signature namespace as the default namespace and consequently doesn't use a prefix. For example:

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
```

Both forms, with and without a prefix, are equally valid.

The class library supports verifying XML signatures that use no prefix, the ds prefix or some other prefix.

XML signatures generated by the class library will no include a prefix.

When confronted with XML signature verification issues, some third parties who are familiar with the use of the ds prefix will assume that the missing prefix is the issue. This is not the case and, if it ever were the case, this would indicate poorly implemented XML signature verification at the third party.

### **15.3 Troubleshooting Loading Certificates**

When loading X.509 certificates using the .NET X509Certificate2 class, always specify the X509KeyStorageFlags.MachineKeySet flag. This ensures the certificate may be accessed from within IIS. For example:

```
X509Certificate2 x509Certificate = new X509Certificate2("idp.pfx", "password",  
X509KeyStorageFlags.MachineKeySet);
```

#### **15.3.1 Certificates Stored in Files**

When loading a certificate from file, if an access denied exception occurs, this generally indicates a file permissions error.

Firstly, make sure the process the application is running under has permission to read the certificate file. For example, ensure the IIS\_USERS group has read permission to the idp.pfx file.

If the permissions are correct for the pfx file, then the issue lies with the private key.

Private keys are stored in containers on the file system. The location of the private key container on Windows 7/Windows 8/Windows 2008 and Windows 2012 is:

C:\ProgramData\Microsoft\Crypto\RSA\MachineKeys

The FindPrivateKey.exe Windows SDK utility may be run to locate the private key container.

<http://msdn.microsoft.com/en-us/library/aa717039.aspx>

The application process must have permission to create a file in the private key container folder.

The following code dumps out a certificate:

```
X509Certificate2 x509Certificate = new X509Certificate2("idp.pfx", "password",  
X509KeyStorageFlags.MachineKeySet);
```

```
Trace.Write(x509Certificate.ToString(true));
```

Its output is:

[Version]  
V1

[Subject]  
CN=www.idp.com  
Simple Name: www.idp.com  
DNS Name: www.idp.com

[Issuer]  
CN=www.idp.com  
Simple Name: www.idp.com  
DNS Name: www.idp.com

[Serial Number]  
46D399D0

[Not Before]  
8/28/2007 1:43:12 PM

[Not After]  
8/25/2017 1:43:12 PM

[Thumbprint]  
4E387A0C0B695DB05F5DFD70D2572BB0FE98BA

[Signature Algorithm]

md5RSA(1.2.840.113549.1.1.4)

[Public Key]

Algorithm: RSA

Length: 1024

Key Blob: 30 81 89 02 81 81 00 a3 7d 6a de 62 59 6b 25 df 66 42 c3 b8 b7 27 6a 77 3f 28 6f 91  
0c 55 be 3a 56 03 3f e4 6e 6e f5 a7 b7 c5 f9 8e d8 94 4d ca 7c 21 0e 3a 4c d1 14 16 c9 26 b2 89  
d4 6f 90 27 1b ec ce 09 c6 b0 6f 67 49 af c9 01 b4 23 61 7e 2f d3 b9 f6 46 05 03 63 b8 0c 4d 32  
2d f8 c8 88 11 74 68 a7 6b 39 c7 81 c9 6b 00 82 19 4f 22 9e ad 0a 98 8c f2 f5 c5 10 ec 14 6a 73  
a8 61 a2 ff 6a 29 cc df 27 57 99 02 03 01 00 01

Parameters: 05 00

[Private Key]

Key Store: Machine

Provider Name: Microsoft Base Cryptographic Provider v1.0

Provider type: 1

Key Spec: Exchange

Key Container Name: {7D7021F3-C4E9-44C2-BB68-ECD0517EF5FE}

Unique Key Container Name: 1cff1ca21ad134bb7e6e87ee27ff71d3\_cddd8d16-473b-40eb-8c9f-  
9cb4b8d44d33

Hardware Device: False

Removable: False

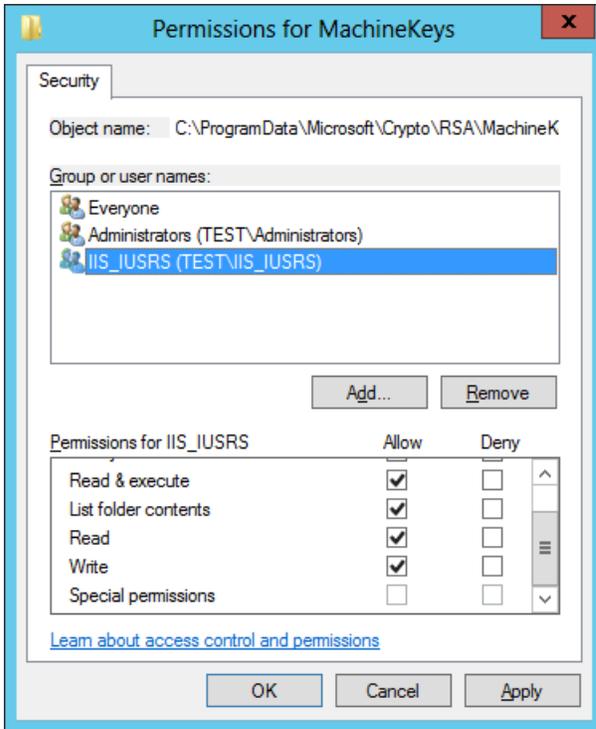
Protected: False

Note the unique key container name. This name is different each time the certificate is loaded.

At the time the certificate is loaded, a file with the private key container name is created in the private key container (e.g.

C:\ProgramData\Microsoft\Crypto\RSA\MachineKeys\7D7021F3-C4E9-44C2-BB68-ECD0517EF5FE).

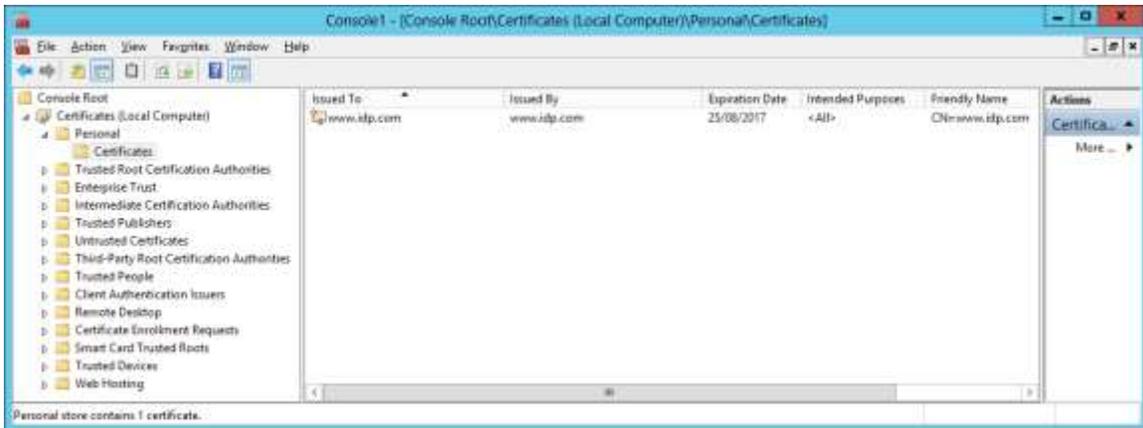
The application process must have permission to create files in this container.



### 15.3.2 Certificates Stored in the Windows Certificate Store

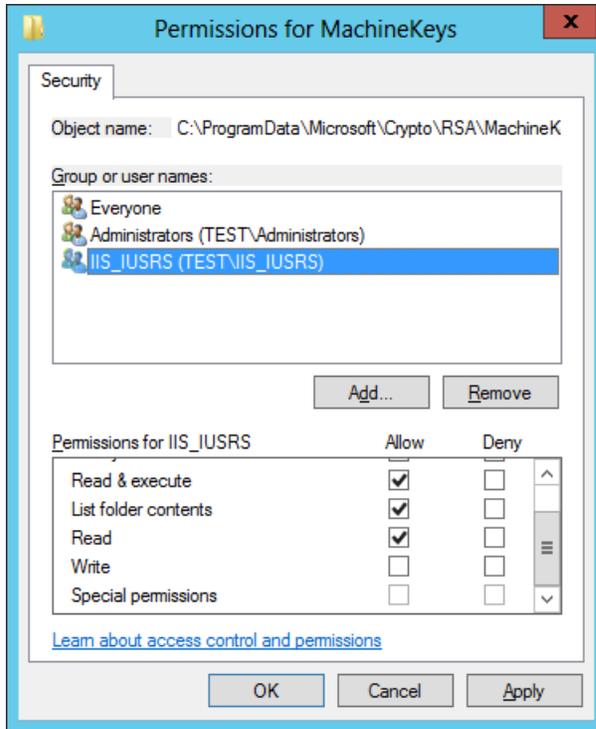
When loading a certificate from the Windows Certificate Store, if an access denied exception occurs, this generally indicates a permissions error.

Firstly, make sure the certificate is stored in the local computer store rather than the current user's store. This ensures the certificate may be accessed from within IIS.



Select the certificate, and click the menu Action > All Tasks > Manage Private Keys.

Make sure the application process (e.g the IIS\_IUSRS group) has read access.



## 16 Generating and Verifying Signatures

The following data types may be signed:

- ❖ SAML request and response messages
- ❖ SAML assertions
- ❖ SAML metadata

To generate or verify signatures on SAML request or response messages use the methods contained in the `ComponentSpace.SAML2.Protocols.SAMLMessageSignature` class.

To generate or verify signatures on SAML assertions use the methods contained in the `ComponentSpace.SAML2.Assertions.SAMLAssertionSignature` class.

To generate or verify signatures on SAML metadata use the methods contained in the `ComponentSpace.SAML2.Metadata.SAMLMetadataSignature` class.

Supplying the wrong data type to a signature class will result in errors. For example, trying to verify the signature on a SAML assertion using the `SAMLMessageSignature` class is an error.

Refer to the class library documentation described in section 21 for details regarding these classes and their methods. Also, review the example code which demonstrates signing and verifying SAML messages, assertions and metadata.

### 16.1 Signature Generation

To generate a signature on an object it must first be converted to XML. For example, to sign a `SAMLResponse` object it must first be converted to XML by calling its `ToXml`

method. Similarly, to sign a SAMLAssertion object it must first be converted to XML by calling its ToXml method. The returned XmlElement may then be passed into SAMLMessageSignature.Generate, in the case of a SAML response, or SAMLAssertionSignature.Generate, in the case of a SAML assertion, to generate the signature and store it in the XML.

The sequence for constructing and signing a SAML response is:

1. Construct the SAMLResponse object and use its properties and methods to create the SAML response.
2. Call SAMLResponse.ToXml to convert the SAML response to an XmlElement.
3. Call SAMLMessageSignature.Generate, passing in the SAML response XmlElement, to generate a signature.

The sequence for constructing and signing a SAML assertion is:

1. Construct the SAMLAssertion object and use its properties and methods to create the SAML assertion.
2. Call SAMLAssertion.ToXml to convert the SAML assertion to an XmlElement.
3. Call SAMLAssertionSignature.Generate, passing in the SAML assertion XmlElement, to generate a signature.

## **16.2 Signature Verification**

Signature verification must occur on the XML prior to converting it to a SAML object. For example, given a SAML response as an XmlElement, call SAMLMessageSignature.Verify to verify the signature. Once verified, construct as SAMLResponse object using the XmlElement. Similarly, given a SAML assertion as an XmlElement object, call SAMLAssertionSignature.Verify to verify the signature. Once verified, construct a SAMLAssertion object using the XmlElement.

Constructing a SAML object from an XmlElement and then converting it back to XML using the ToXml method will cause the signature verification to fail.

The sequence for constructing and verifying a SAML response is:

1. Call SAMLMessageSignature.Verify, passing in the SAML response XmlElement, to verify the signature.
2. Construct the SAMLResponse object from the SAML response XmlElement.
3. Call the SAMLResponse properties and methods to access the contents of the SAML response.

The sequence for constructing and verifying a SAML assertion is:

1. Call SAMLAssertionSignature.Verify, passing in the SAML assertion XmlElement, to verify the signature.
2. Construct the SAMLAssertion object from the SAML assertion XmlElement.
3. Call the SAMLAssertion properties and methods to access the contents of the SAML assertion.

### **16.3 SHA-256, SHA-384 and SHA-512 Support**

By default, SHA-1 signatures are supported and are perfectly suitable for the majority of use cases. However, SHA-256, SHA-384 and SHA-512 (also referred to as SHA-2) signatures are also supported for those use cases requiring additional security.

The SHA256Signature example project demonstrates SHA-256, SHA-384 and SHA-512 signature generation and verification. Successfully running this example project confirms that SHA-256, SHA-384 and SHA-512 support is enabled.

There are a number of options for supporting SHA-256, SHA-384 and SHA-512 XML signatures depending on the target .NET framework level.

#### **16.3.1 .NET 4.0 Framework Support**

For .NET 4.0 and above, SHA-256, SHA-384 and SHA-512 support is, for the most part, built into the .NET framework. The only requirement is to register the cryptographic algorithm. This is done automatically when using the .NET 4.0 version of the ComponentSpace.SAML2 DLL.

#### **16.3.2 CLR Security Update**

For .NET 3.5 and above, 256, SHA-384 and SHA-512 support in XML signatures requires the use of the CLR security update.

Version 1.6.0.0 of the CLR security update is recommended as a strong named version of the CLR security update DLL is required for installation into the GAC.

Download the CLR security update from:

<http://clrsecurity.codeplex.com/releases/view/47764>

Installation instructions may be found at:

<http://clrsecurity.codeplex.com/wikipage?title=Security.Cryptography.RSAPKCS1SHA256SignatureDescription&referringTitle=Home&ProjectName=clrsecurity>

1. Extract the Security.Cryptography DLL from the CLR security zip.
2. Run `gacutil.exe /i Security.Cryptography.dll` to add the assembly to the GAC.
3. View the assembly (e.g. `C:\Windows\assembly`) and note the version number (e.g. 1.6.0.0). Alternatively, the version number may be found by running:

```
gacutil.exe /l Security.Cryptography
```

4. Update machine.config (e.g. in `C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config` and

C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config) ensuring the version number of the assembly is correct. The <mscorlib> should be inserted after the <system.web> section in <configuration>. See below for an example configuration.

5. Certificates and keys should be generated using the “Microsoft Enhanced RSA and AES Cryptographic Provider”.

The following is an example SHA-256 configuration for insertion into machine.config.

```
<mscorlib>
  <!-- ... -->
  <cryptographySettings>
    <cryptoNameMapping>
      <cryptoClasses>
        <cryptoClass
RSASHA256SignatureDescription="Security.Cryptography.RSAPKCS1SHA256SignatureDescription, Security.Cryptography, Version=1.6.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
        </cryptoClass>
      </cryptoClasses>
      <nameEntry name="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" class="RSASHA256SignatureDescription" />
    </cryptoNameMapping>
  </cryptographySettings>
</mscorlib>
```

To generate SHA-256 signatures, use one of the overloaded signature Generate methods that take as parameters the digest and signature methods.

The default digest method is <http://www.w3.org/2000/09/xmldsig#sha1>.

The default signature method is <http://www.w3.org/2000/09/xmldsig#rsa-sha1>.

Instead of the defaults, specify <http://www.w3.org/2001/04/xmlenc#sha256> as the digest method and <http://www.w3.org/2001/04/xmldsig-more#rsa-sha256> as the signature method.

For example:

```
SAMLMessageSignature.Generate(
    samlResponseElement,
    x509Certificate.PrivateKey,
    x509Certificate,
    null,
    "http://www.w3.org/2001/04/xmlenc#sha256",
    "http://www.w3.org/2001/04/xmldsig-more#rsa-sha256");
```

No code changes are required to verify SHA-256 signatures.

## 17 Extracting SAML Assertions from SAML Responses

SAML responses may contain one or more of the following:

- ❖ Encrypted SAML assertions
- ❖ Signed SAML assertions
- ❖ SAML assertions that are neither encrypted nor signed

The `ComponentSpace.SAML2.Protocols.SAMLResponse` class encapsulates a SAML response message. To access the various types of SAML assertions contained within it use of the following properties or methods from this class:

- ❖ `Assertions`
- ❖ `GetEncryptedAssertion`
- ❖ `GetSignedAssertion`
- ❖ `GetAssertion`
- ❖ `GetEncryptedAssertions`
- ❖ `GetSignedAssertions`
- ❖ `GetAssertions`

The `Assertions` property returns all assertions including encrypted and signed assertions. Encrypted assertions are returned as `EncryptedAssertion` objects. Signed assertions are returned as `XmlElement` objects. Unencrypted, unsigned assertions are returned as `SAMLAssertion` objects.

The `GetEncryptedAssertion` method returns the encrypted assertion as an `EncryptedAssertion` object. Section 18 describes how to process encrypted assertions.

The `GetSignedAssertion` method returns the signed assertion as a `SAMLAssertion` object after verifying the signature.

The `GetAssertion` method returns the unencrypted and unsigned assertion as a `SAMLAssertion` object.

Typically a SAML response contains one SAML assertion. The following methods support retrieving multiple SAML assertions.

The `GetEncryptedAssertions` method only returns the encrypted assertions as `EncryptedAssertion` objects. Section 18 describes how to process encrypted assertions.

The `GetSignedAssertions` method only returns the signed assertions as `XmlElement` objects. Signed assertions are returned as `XmlElement` objects as this is the format required for signature verification. Section 15.3.2 describes how to verify a signature. Once the signature is verified a `SAMLAssertion` object may be constructed from the `XmlElement`.

The `GetAssertions` method only returns the unencrypted and unsigned assertions as `SAMLAssertion` objects.

The list of objects returned by the `Assertions` property is equivalent to combining the three lists returned by the `GetEncryptedAssertions`, `GetSignedAssertions` and `GetAssertions` methods.

### **17.1 Extracting a SAML Assertion**

The following section of code returns the unsigned and unencrypted SAML assertion from the SAML response.

```
SAMLAssertion samlAssertion = samlResponse.GetAssertion();
```

### **17.2 Extracting a Signed SAML Assertion**

The following section of code returns the signed SAML assertion from the SAML response after having verified its signature. The supplied X.509 certificate is used to perform the signature verification.

```
SAMLAssertion samlAssertion =  
    samlResponse.GetSignedAssertion(x509Certificate);
```

### **17.3 Extracting an Encrypted Assertion**

The following section of code extracts encrypted SAML assertion from the SAML response. The supplied X.509 certificate is used to perform the decryption.

```
EncryptedAssertion encryptedAssertion =  
    samlResponse.GetEncryptedAssertion();  
  
SAMLAssertion samlAssertion =  
    encryptedAssertion.Decrypt(x509Certificate);
```

## **18 Encrypted Assertions**

For additional security, SAML assertions may be encrypted.

To encrypt a SAML assertion construct an `EncryptedAssertion` object supplying the `SAMLAssertion` to be encrypted, an X.509 certificate, and specifying the type of encryption to perform.

A random symmetric session key is generated from the public key contained within the X.509 certificate. The symmetric key is used to encrypt the data. The encryption method for the encrypted symmetric key is [http://www.w3.org/2001/04/xmlenc#rsa-1\\_5](http://www.w3.org/2001/04/xmlenc#rsa-1_5). The encryption method for the data is that specified in the `EncryptedAssertion` constructor (e.g. <http://www.w3.org/2001/04/xmlenc#aes256-cbc>).

To decrypt an `EncryptedAssertion` call either the `Decrypt` or `DecryptToXml` method. The `decrypt` method returns a `SAMLAssertion` object. The `DecryptToXml` method returns an `XmlElement` and should be used if the encrypted assertion is also signed. Both methods accept an asymmetric key decrypting key and an optional data encryption method.

The asymmetric key is used to decrypt the symmetric key contained within the encrypted data. The symmetric key is used to decrypt the data.

The encryption method for the encrypted symmetric key is expected to be [http://www.w3.org/2001/04/xmlenc#rsa-1\\_5](http://www.w3.org/2001/04/xmlenc#rsa-1_5) or <http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p>.

If the optional data encryption method is not specified then the encryption method for the data supplied in the encrypted data is used.

The example project described in section 11.10 demonstrates encrypting and decrypting SAML assertions.

## 19 Extracting Statements from SAML Assertions

SAML assertions may contain one or more of the following:

- ❖ Authentication statements
- ❖ Authorization decision statements
- ❖ Attribute statements

The `ComponentSpace.SAML2.Assertions.SAMLAssertion` class encapsulates a SAML assertion. To access the various types of statements contained within it use of the following properties or methods from this class:

- ❖ `Statements`
- ❖ `GetAuthenticationStatements`
- ❖ `GetAuthorizationDecisionStatements`
- ❖ `GetAttributeStatements`

The `Statements` property returns all statements including authentication, authorization decision and attribute statements. Authentication statements are returned as `AuthnStatement` objects. Authorization decision statements are returned as `AuthzDecisionStatement` objects. Attribute statements are returned as `AttributeStatement` objects.

The `GetAuthenticationStatements` method only returns the `AuthnStatement` objects. The `GetAuthorizationDecisionStatements` method only returns the `AuthzDecisionStatement` objects. The `GetAttributeStatements` method only returns the `AttributeStatement` objects.

The list of objects returned by the `Statements` property is equivalent to combining the three lists returned by the `GetAuthenticationStatements`, `GetAuthorizationDecisionStatements` and `GetAttributeStatements` methods.

## 20 Extracting SAML Attributes

As well as the methods described in section 19 for accessing attribute statements contained in SAML assertions, the `ComponentSpace.SAML2.Assertions.SAMLAssertion` class includes the following convenience methods to access SAML attributes:

- ❖ GetAttributes
- ❖ GetAttributeValue

The GetAttributes method returns the list of all unencrypted SAML attributes with the specified name.

The GetAttributeValue method returns the value of the SAML attribute with the specified name. This method assumes that only one attribute exists with the specified name, the attribute only has one value, and the value is a string. If this isn't the case then use one of the other attribute related methods.

## 21 Class Library Reference

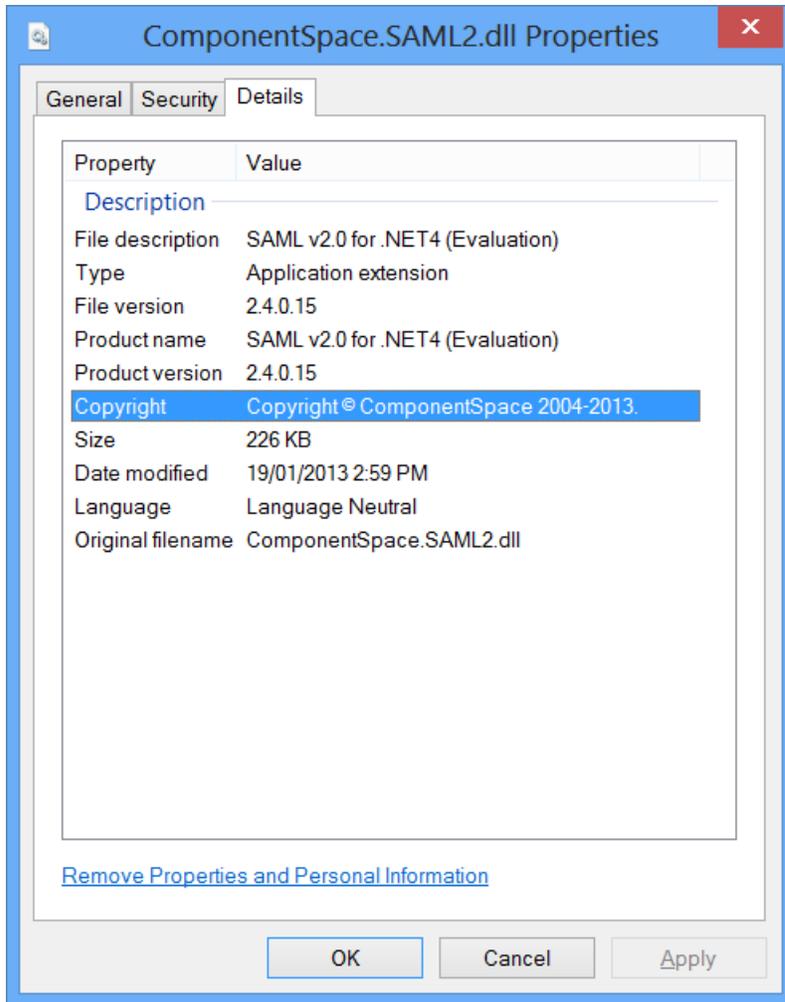
The reference section is contained within a separate help file. You can find the reference help file in the documentation directory or navigate to it using the Start menu.

## 22 Class Library Version

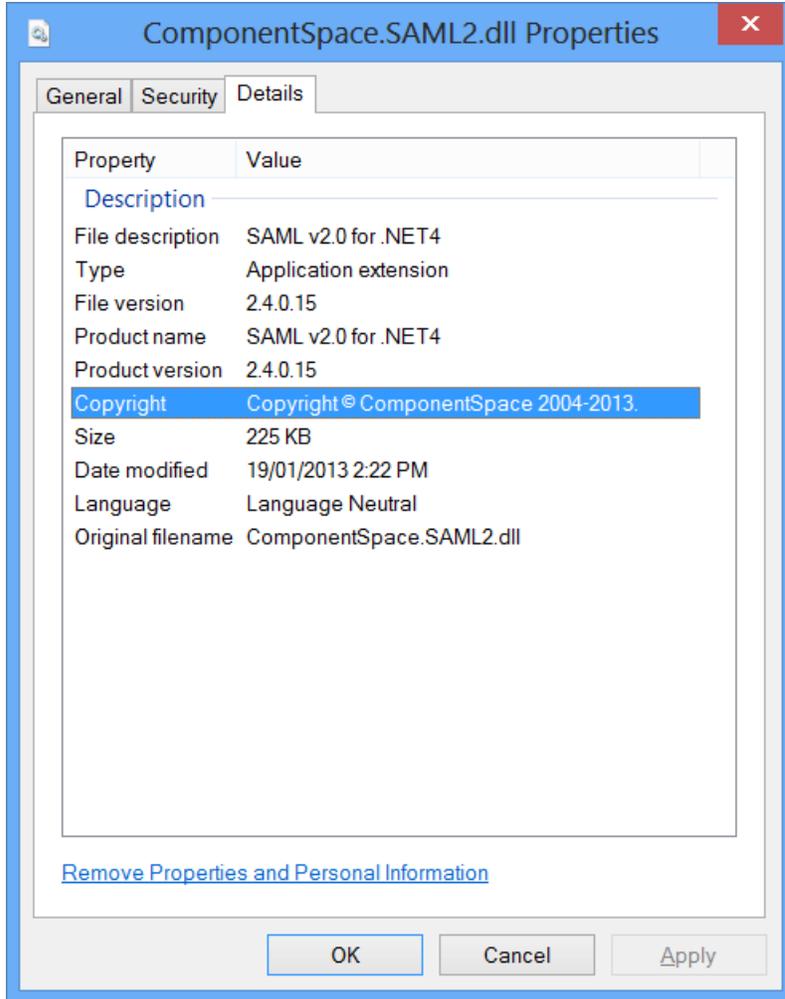
The License.IsLicensed method may be used to determine whether the class library is a licensed or evaluation version. The version information may be retrieved by calling the License.GetVersion method.

Alternatively, using Windows Explorer, select the class library DLL and bring up its file properties. Under the Details tab the version number may be found. If the DLL is an evaluation version then this also will be indicated.

The following properties are for the evaluation version of the DLL.



The following properties are for the licensed version of the DLL.



## 23 Frequently Asked Questions

### 1. *What is SAML?*

The Security Assertion Markup Language (SAML) consists of a set of standards published by the OASIS organization ([www.oasis-open.org](http://www.oasis-open.org)). Refer to their web site for more details.

### 2. *What version of SAML is supported?*

We support SAML v1.1 and v2.0. Please note that these are packaged as separate products.

### 3. *I can't build the examples. Why can't the ComponentSpace namespace be found?*

The assembly containing the namespace or the project referencing it has been moved. The simplest way to fix this is to re-add the reference to the project by browsing to where the DLL is located (typically the installation directory).

4. *I keep receiving an error message saying the trial period has expired. What does this mean?*

You are using an evaluation version of the component and the trial period has ended. If you need to extend the evaluation period, please [contact us](#).

5. *How do I tell if I'm using an evaluation version?*

Navigate to the component DLL and right click it to bring up the file properties. Under the Version tab, the description will specify whether or not it's an evaluation version.

6. *I'm not sure how to use the product. What can I do?*

Whether you're evaluating the product or are an existing customer, please feel free to [contact us](#) with any questions you might have.

7. *The product is missing a feature I really need. What can I do?*

Please [contact us](#) if there's additional functionality you would like to see. Your feedback is most welcome and will be given careful consideration.

8. *Does the product support SAML single sign-on?*

Yes. The product includes the necessary functionality for enabling SAML single sign-on at either the identity provider or service provider web site. You need to integrate this functionality with your existing web applications.

9. *Can I generate and verify XML digital signatures on SAML assertions and protocol messages?*

Yes. Refer to the Reference section and the Examples for more information.

10. *Is the product compatible with Product X?*

It should be but if you have any questions please [contact us](#). A simple test is to integrate the example identity provider or service provider application with the product in question.

11. *How do I create a SAML assertion?*

Use the ComponentSpace.SAML2.Assertions.SAMLAssertion class to create a SAML assertion.

The example applications demonstrate creating and manipulating SAML assertions.

12. *How do I convert a SAML assertion to and from XML?*

The ComponentSpace.SAML2.Assertions.SAMLAssertion class includes a

constructor that creates a SAML assertion from an XML element. This class also includes a ToXml method that converts the SAML assertion object to XML.

13. *How do I sign a SAML assertion?*

The ComponentSpace.SAML2.Assertions.SAMLAssertionSignature class has methods for generating and verifying signatures on SAML assertions that are serialized to XML.

Typically you create a SAML assertion using the ComponentSpace.SAML2.Assertions.SAMLAssertion class. Once complete, you convert it to XML using the ToXml method of this class. Then you pass this XML to the Generate method of the SAMLAssertionSignature class.

14. *How do I verify a signed SAML assertion?*

The ComponentSpace.SAML2.Assertions.AssertionSignature class has methods for generating and verifying signatures on SAML assertions that are serialized to XML.

Typically you verify the signature by passing the XML to the Verify method of the SAMLAssertionSignature class. Once verified, you create a SAML assertion from the XML using the ComponentSpace.SAML2.Assertions.Assertion class.

15. *How do I create a SAML protocol message?*

Use the classes under the ComponentSpace.SAML2.Protocols namespace for creating protocol messages. For example, to create an authentication request you would use the AuthnRequest class.

16. *How do I convert a SAML protocol message to and from XML?*

The protocol message classes include a constructor that creates a SAML protocol message from an XML element. These classes also include a ToXml method that converts the SAML protocol message to XML.

17. *How do I sign a SAML protocol message?*

The ComponentSpace.SAML2.Protocols.SAMLMessageSignature class has methods for generating and verifying signatures on SAML protocol messages that are serialized to XML.

Typically you create a SAML message using the corresponding class in the ComponentSpace.SAML2.Protocols namespace. Once complete, you convert it to XML using the ToXml method of this class. Then you pass this XML to the Generate method of the SAMLMessageSignature class.

18. *How do I verify a signed SAML protocol message?*

The ComponentSpace.SAML2.Protocols.SAMLMessageSignature class has

methods for generating and verifying signatures on SAML protocol messages that are serialized to XML.

Typically you verify the signature by passing the XML to the Verify method of the SAMLMessageSignature class. Once verified, you create a SAML protocol message from the XML using the appropriate constructor for the protocol message's corresponding class in the ComponentSpace.SAML2.Protocol namespace.

19. *What's the difference between CurrentUser and LocalMachine when referring to certificate store locations?*

Please refer to the Microsoft knowledge base article Q322371. If running in an ASP.NET application then keys may need to be loaded from LocalMachine key stores.

20. *How do I use HTTPS to secure the connection between identity provider and the service provider?*

The MSDN has numerous articles on configuring HTTPS within an ASP.NET environment. A good starting point is the article *Building Secure ASP.NET Applications: Authentication, Authorization, and Secure*.

## 24 Support

For further information, visit us at [www.componentspace.com](http://www.componentspace.com) or send email to [info@componentspace.com](mailto:info@componentspace.com).

If you need assistance, have a bug to report, or a product enhancement request, send email to [support@componentspace.com](mailto:support@componentspace.com).