Implementing Claims-Based Authentication with SharePoint Server 2010

Implementing Claims-Based Authentication with SharePoint Server 2010

Microsoft Corporation

September 2011

Applies to: Microsoft SharePoint® Server® 2010, Active Directory® Federation Services

Summary: This whitepaper provides guidance for IT professionals and developers about designing and deploying solutions that support claims-based authentication in Microsoft SharePoint® Server® 2010.

Authors and Contributors

David Crawford, Senior Consultant II, Microsoft Corporation
Steve Peschka, Senior Principal Service Engineer, Microsoft Corporation
Bill Baer, Senior Technical Product Manager, Microsoft Corporation
Sesha Mani, Senior Program Manager, Microsoft Corporation
Joe Fuentes, Senior Consultant, Microsoft Corporation
Richard Harrison, Senior Consultant I, Microsoft Corporation
Aaron Isom, Solution Architect, Microsoft Corporation
John Moh, Senior Consultant, Microsoft Corporation
Bert Jansen, MSC Architect, Microsoft Corporation
Brent Groom, Senior Consultant, Microsoft Corporation
Bryan Porter, Senior Consultant, Microsoft Corporation
Daniel Akins, Senior Program Manager, Microsoft Corporation
Ken St. Cyr, Solution Architect, Microsoft Corporation
Paul Learning, Senior Consultant II, Microsoft Corporation
Raju Sakthivel, Delivery Architect, Microsoft Corporation
Todd Foust, Escalation Engineer, Microsoft Corporation
Tom Wisnowski, Delivery Architect, Microsoft Corporation
Gary Martinez, Principal Project Manager, Microsoft Corporation

Contents

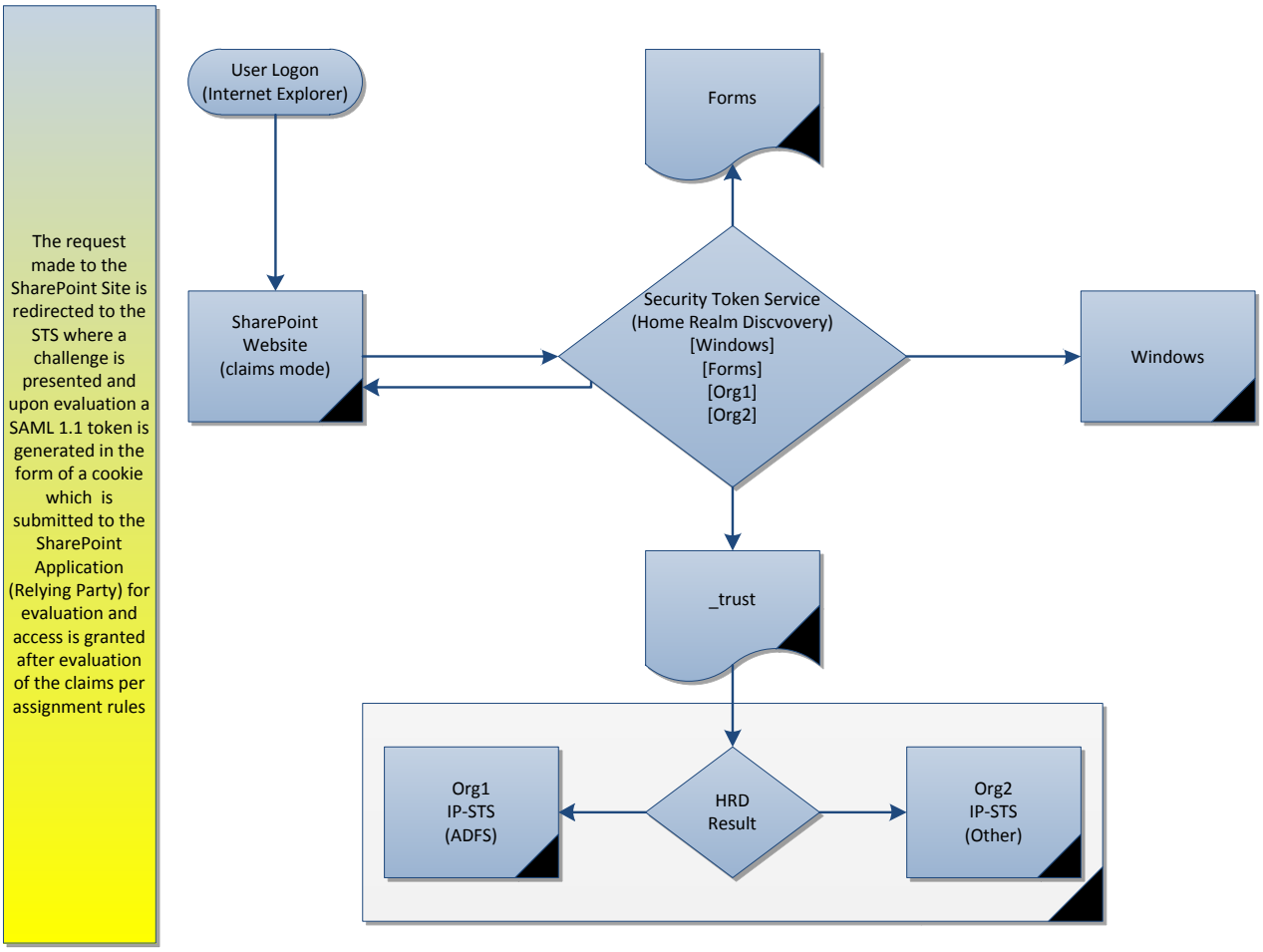
Overview of Claims-based Identity in SharePoint Server 2010

This document is intended for a SharePoint Server 2010 solution team that may have development experience in addition to an infrastructure background. This whitepaper is intended for readers who have some familiarity with claims-based authentication concepts such as WS-Federation and WS-Trust. The terms in many cases will be familiar to those who have implemented solutions with the Windows Identity Foundation. Many terms will be explained in context. However, this whitepaper focuses primarily on features and functionality that are core in producing a SharePoint Server 2010 solution utilizing Claims Mode Authentication Web applications.

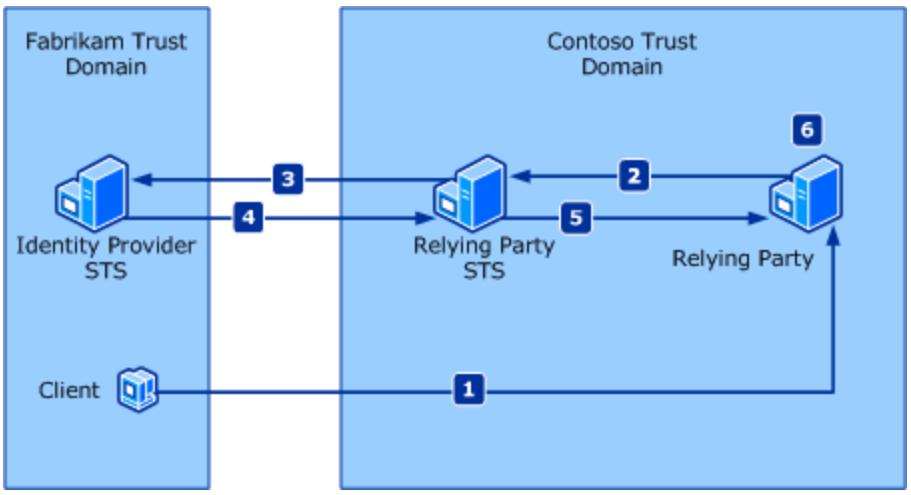
Claims-based Authentication

SharePoint Server 2010 introduces a new capability that enables running in a claims authentication mode. A key benefit to this scenario is the ability to provide authenticated access to entities external to your organization as well as enable multiple authentication types within a single SharePoint zone. Access to SharePoint Server running in Claims Mode Authentication utilizes a Security Token Service (STS) which is essentially an authentication gateway to SharePoint Server that enables access for Windows Integrated Authentication, Form Based Authentication and Trusted Claims Providers (TRUST). This layer or gate, requests that credentials are presented and upon successful evaluation, transitions to claims-based access with WS-Federation. SharePoint Server 2010 adds extensibility points to meet the experience needs presented when running in this mode.

This whitepaper provides many of the details related to setting up a trust configuration, an explanation of runtime considerations and management, configuration details, programmatic implementations or use of the extensibility points that turn federated access into a federated SharePoint Server 2010 business solution. The majority of this whitepaper will focus on the trusted claims provider scenario; however, it provides additional insight into the differences related to working with services under the authentication types available when running one or more Claims Mode Authentication Web applications.



The following illustration is an alternative representation with SharePoint Server 2010 as the relying party.



Typical Tasks for Claims-Based Sites

The following are common tasks that implementers will need to follow when deploying and configuring Web applications for Claims Mode Authentication in SharePoint Server 2010 in a trusted configuration.

1. Acquire certificates for each SharePoint Web application, Active Directory Federation Services endpoint, or Security Token Service (STS).

2. A default SharePoint Server 2010 installation results in a Classic Mode Authentication initial Web application. The initial Web application can be deleted and replaced with a Claims Mode Authentication Web application. Optionally, the SPWebApplication.UseClaimsAuthentication property can be set to True by using the SharePoint Management Shell. For more information see Install SharePoint Server 2010 by using Windows PowerShell (http://go.microsoft.com/fwlink/?LinkID=118674).

3. Load the certificate chains utilized with Active Directory Federation Services (AD FS) and SharePoint Server into the respective management stores.

4. Determine the claims utilized for authentication and authorization assignment:

    a) E-mail is typically the user identity of choice as it is common, unique, and has a secondary function of verification through communication. Other claims may be utilized as well and for whatever rationale works best within your organization.

    b) Evaluate use for a User Principal Name (UPN) claim to work with the Claims to Token Windows Service (c2wts).

    c) Claims augmentation may be performed at any issuer. In many cases, Active Directory Federation Services will be used and has a capability to reach into attribute stores such as Active Directory Domain Services, Microsoft® SQL Server, or Active Directory Lightweight Directory Services (AD-LDS) to generate a claim through functionality provided in the AD FS product. Another location that may be used for augmentation is a programmatic extensibility point within the SharePoint Security Token Service.

    d) The User Profile Application provides social functionality. There are configuration selections that define how a URL is formed for My Site public sites. When integrated with Active Directory Domain Services, an example of the rule would be to display only the account name, create the My Site with the domainname_accountname when there is a conflict, or always configure the My Site URL as domain_username. That accountname may be controlled by specifying the "windowsaccountname" claim.

5. Establish relying partner trust for each SharePoint Web application. It may be useful to use a URI such as the following pattern urn:SharePointSiteName:AD FS and with a third-party STS use an extension that matches the type of STS that is being integrated with.

This recommendation is not required; however, it can be useful in troubleshooting. Establish claims provider trust with SharePoint Server as well.

a) Establish the claims that are acceptable by using Identity Provider Security Token Service (IP-STS) and enable appropriate configuration in AD FS. There are a number of administrative models to evaluate when using federated identities. One approach is to use AD FS as a gateway to other partners due to the full featured nature of the product compared to the subset of federation capabilities provided by the Security Token Service in SharePoint Server. In other words, the Security Token Service in SharePoint Server is not designed to be a replacement for AD FS functionality.

b) Establish additional claims provider trusts as necessary.

6. Test assignment with the people/object picker:

   a) Set a user ID in Central Administration for ownership of Web/site collection.

   b) Set a user ID to the critical services and perform role assignments for the site collection administrators, site owners, and members accordingly.

7. Determine custom object picker approach. It will most likely be preferable that the default people picker be replaced with one that utilizes a Windows Identity Foundation abstraction to a claims store. This would be for the user or role assignment functionality to have some validation.

   a) Design the approach

   b) Implement

   c) Test

8. Design and build custom sign-out or sign-in as another user. The default functionality is unaware of the location of the STS that is required to sign-out. For example, the following URL, https://{RP-STS-DNS}/ADFS/ls/?wa=signout1.0, or more concrete example URL, https://sts.contoso.com/ADFS/ls/?wa=signout1.0, would be the location to direct the browser to perform single sign-out (to sign out the user from all of the applications that trust the STS). To perform sign-out for a single relying party, specify the wsignoutcleanup1.0 parameter. For example, https://sts.contoso.com/ADFS/ls/?wa=wsignoutcleanup1.0.

   The primary option to select is a custom sign-out page that redirects to the previously listed URL. This will result in the signing-out of all applications at the STS, and it is important because the sign-out control is not accessible programmatically. Another approach, however, is to expose a link that is based on an audience. Exposing a link that is based on an audience might be confused with the option of using a drop down to logoff/logon as another user. The net result is that the redirect to the RP-STS will destroy the SAML token and cookie. To sign-in as different user, return to the site, redirect to the IP-STS, and authenticate again. Attempting to sign-out from a trust configuration without

performing the federated sign-out will simply redirect the browser right back to the application as if nothing were called.

9. Profile Service Implementation: Two services User Profile Application and User Profile Sync are implemented – identify changes in order to affect desired experience - a couple of key attributes are desired to provide key functionality:

    a) Import the Session Initiated Protocol (SIP) address to support presence (the bubbles that work with Microsoft Lync™ that tell you someone else is online or which may be visible in a Microsoft Word document when tandem editing.

    b) Profile Pictures only show in the Silverlight® organization browser from the SharePoint zone they are uploaded to. It is something to consider when using multiple zones.

10. Design an authentication strategy for search. When you establish a crawl and have multiple authentication methods on a single zone, you will likely have to change the search protocol handler to support the secure connection by adding an "s" as seen here sps3**s**://mysite. However, there may be multizone configurations where the Default Zone uses Windows Integrated security which only requires the default handler as sps3://mysite

    a) Services are evaluated with business requirements to determine whether the required functionality works with claims-based sites. Areas of consideration are covered in this whitepaper.

11. Certificate Revocation List (CRL) – ensure the CRL is configured properly between SharePoint Server and Active Directory Federation Server (AD FS) or significant delays possible in the authentication process may occur due to attempts to reach a CRL.

Other considerations:

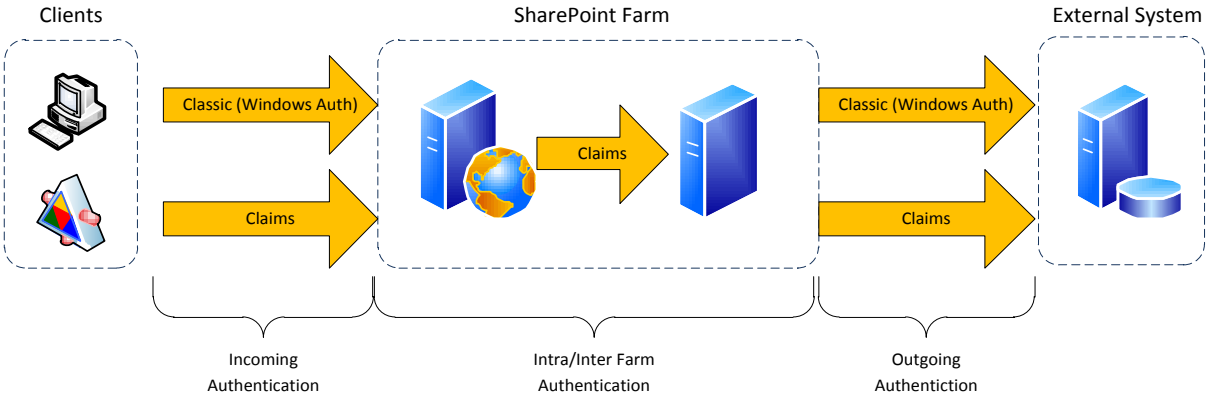
- Web applications will run under secure sockets layer (SSL) and images do not automatically show up in search without prior authentication to the picture store. WS-Federation does not cause the browser to perform a 401 prompt such as with Windows integrated security.
- Ensure that your identity attribute is available for your claims. For example, if you use an email address as the identity claims, that email address needs to be populated in Active Directory Domain Services for AD FS to generate a claim for it. Attempting to reach an STS without the expected identity claims will result in a rather unfriendly error.
- Ensure that at minimum Service Pack 1 for SharePoint 2010 products is applied.
- Set the logon token cache properly. A common symptom is the logon process going in a browser redirect loop.
- Smart Card Authentication is not supported directly with SharePoint Server at the time of this publication; however, you may use Smart Cards with AD FS 2.0.
- FAST Search Server 2010 for SharePoint document preview does not work with SharePoint Server 2010 claims-based Web applications.

Overview Concepts and Terminology

Identity Scenarios in SharePoint Server 2010 Products

When learning about identity in the context of authentication in SharePoint Server 2010, you can conceptually look at how the platform handles identity in three key scenarios:

- Incoming Authentication
- Inter/Intra Farm Authentication
- Outgoing authentication.



Incoming Identity

The incoming authentication scenario represents the means with which a client presents its identity to the platform, or in other words "authenticates" with the Web application or Web Service. SharePoint Server will use the client's identity to authorize the client to access SharePoint secured resources such as Web pages, documents, and so on.

SharePoint Server 2010 support two modes in which a client can authenticate with the platform: Classic and Claims Mode Authentication.

Claims-Based Authentication

Support for claims-based authentication is a new feature in SharePoint Server 2010 built on the Microsoft Windows Identity Foundation (WIF). In a claims model, SharePoint Server will accept one or more "claims" about an authenticating client to identify and authorize the client. The claims come in the form of SAML tokens and are simply "facts" about the client stated by a "trusted" authority. For example, a claim could state "Bob is a member of the "enterprise admins" group for the domain Contoso.com". If this claim came from a provider that SharePoint Server trusts, the platform could use this information to authenticate Bob and to authorize him to access SharePoint resources. For more information about claims authentication, see A Guide to Claims-based Identity and Access Control (http://go.microsoft.com/fwlink/?LinkID=187911).

The types of claims authentication modes that SharePoint Server 2010 supports for incoming authentication are:

Windows Claims

In the Windows claims mode sign in, SharePoint Server authenticates the client using standard Integrated Windows authentication (NTLM/Kerberos) and then translate the resulting Windows Identity into a claims identity.

Forms-Based Authentication Claims

In forms-based authentication claims mode, SharePoint Server redirects the client to a login page hosting the standard ASP.NET login controls. The page authenticates the client using the ASP.NET membership provider, similar to the way in which forms-based authentication functions in Office SharePoint Server 2007. After the identity object that represents the user is created, SharePoint Server will then translate this identity to a claims identity object.

SAML-Claims

In SAML claims mode, SharePoint Server accepts SAML tokens from a trusted external Security Token Provider (STS) often known as a claims provider trust. A user who attempts to login is directed to an external claims provider (for example, Windows Live ID claims provider) which authenticates the user and produce a SAML token. SharePoint Server accepts and processes this token, augmenting the claims and creating a claims identity object for the user.

For more information about claims-based authentication in SharePoint Server 2010 refer to SharePoint Claims-Based Identity (http://go.microsoft.com/fwlink/?LinkId=196647).

Claims Authentication and the Claims-to-Windows Token Service (C2WTS)

Some service applications require the use of the Windows Identity Foundation (WIF) Claims-to-Windows Token Service (C2WTS) to translate claims within the farm to Windows credentials for outbound authentication. It is important to understand that Service Applications that come with SharePoint Server can leverage the C2WTS only if the incoming authentication method is either Classic mode or Windows claims. Service applications that are accessed through Web applications that leverage SAML claims or forms-based authentication claims do not use the C2WTS, and therefore they are not able to translate claims to Windows credentials.

Identity within a SharePoint Server 2010 Environment

SharePoint Server 2010 environments use claims-based authentication for intra- and inter-farm communications with most SharePoint service applications and SharePoint integrated products regardless of the incoming authentication mechanism. This means that even in situations where Classic authentication is used to authenticate with a particular Web application, SharePoint Server will convert the incoming identity into a claims identity to authenticate with SharePoint service applications and products that are claims-aware. By standardizing on the claims model for intra/inter farm communications, the platform can abstract itself from the incoming protocols.

NOTE

Some SharePoint integrated products, such as SQL Server Reporting Services, are not claims-aware by default and do not leverage the intra-farm claims-based authentication architecture. SharePoint Server may also rely on classic Kerberos delegation as well as claims in other scenarios, for example when the RSS viewer Web part is configured to consume an authenticated feed. Refer to each product or service application's documentation to determine if it can support claims-based authentication and/or identity delegation.

SharePoint Server 2010 with Active Directory Federation Services 2.0

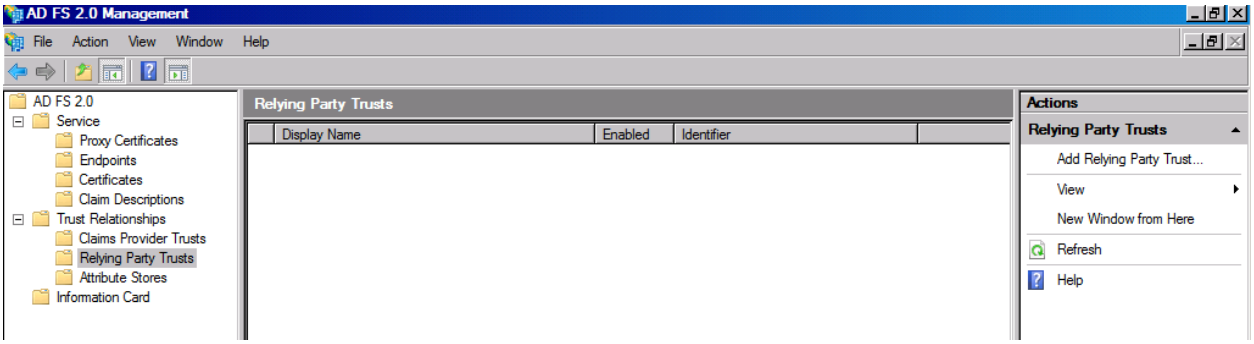
Configuration

This section provides an end-to-end walkthrough that configures SharePoint Server 2010 and AD FS 2.0 to utilize SAML claims authentication. Both the necessary manual steps and corresponding Windows PowerShell scripts are included.

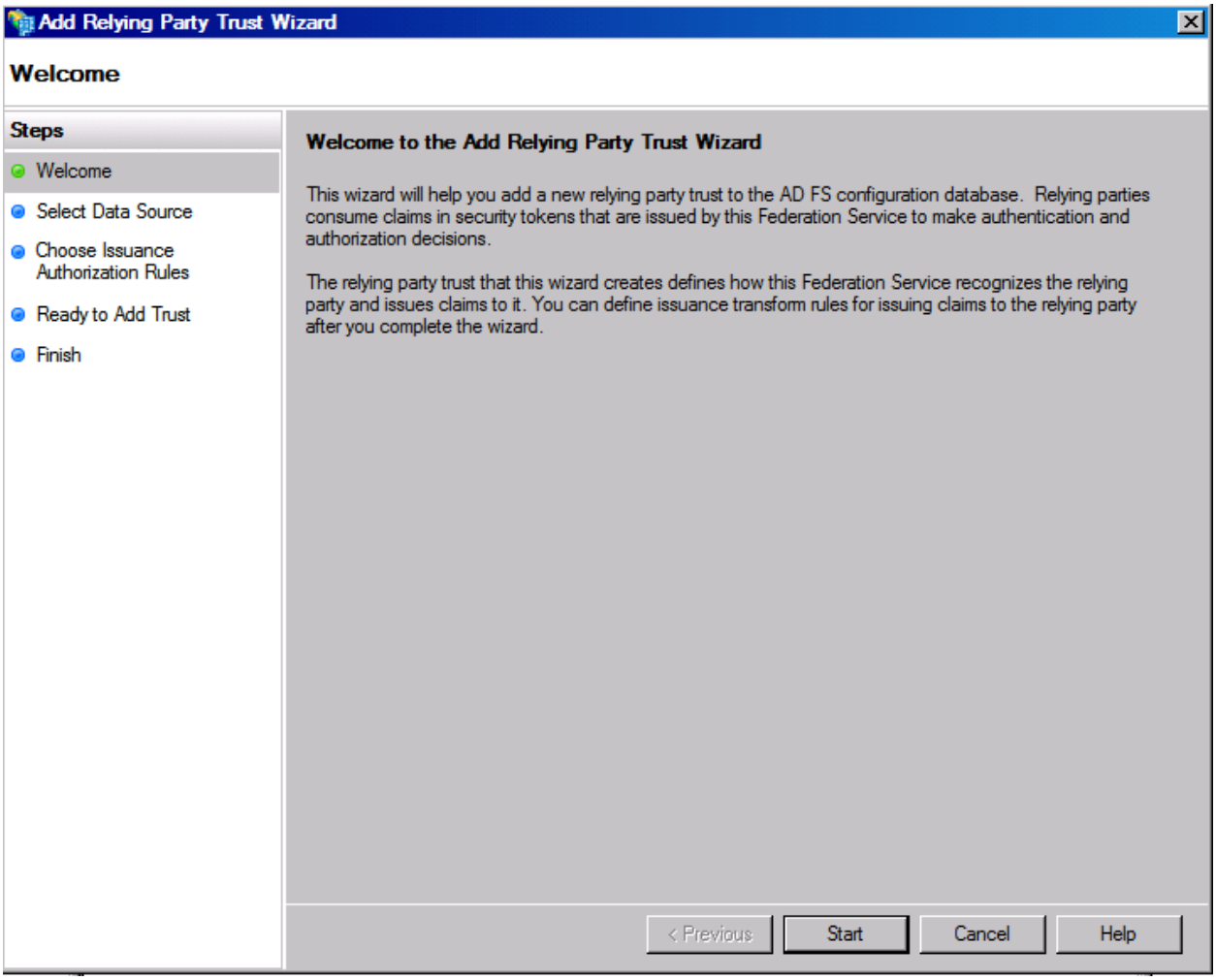
In this scenario AD FS 2.0 is implemented at the Identity Provider, also known as an IP-STS (Security Token Service). AD FS should be configured with information about the relying party. In this scenario, SharePoint Server 2010 is configured as the relying party because it depends on AD FS to perform authentication and to provide the claims. From the SharePoint Server perspective, it must be configured to trust the IP-STS that is sending claims with a Web application configured for Claims Mode Authentication to consume the received claims.

Create the Relying Party

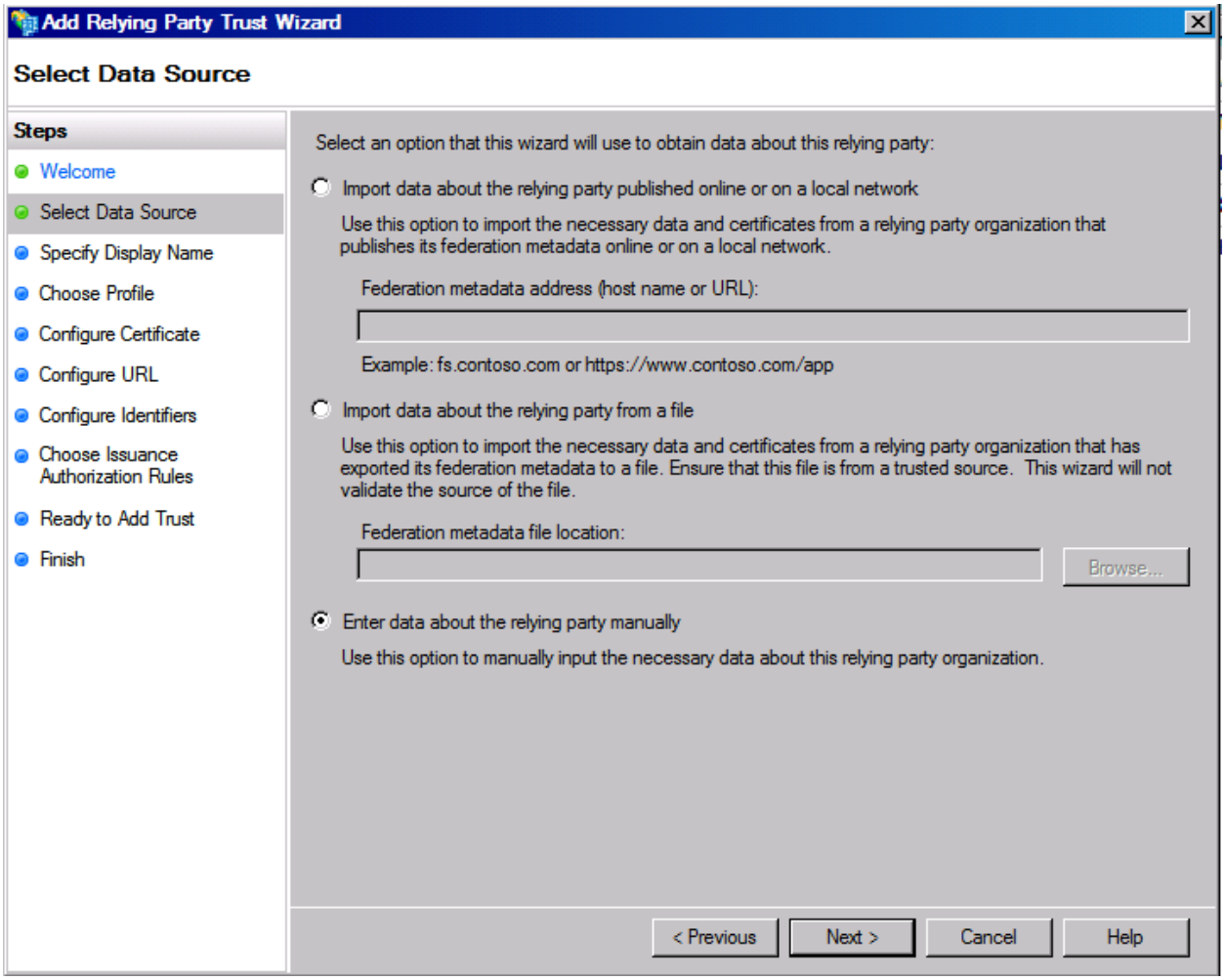
On the server on which AD FS is installed, open the AD FS 2.0 Management application and expand the **Trust Relationships** node, then click on the **Relying Party Trusts** node.



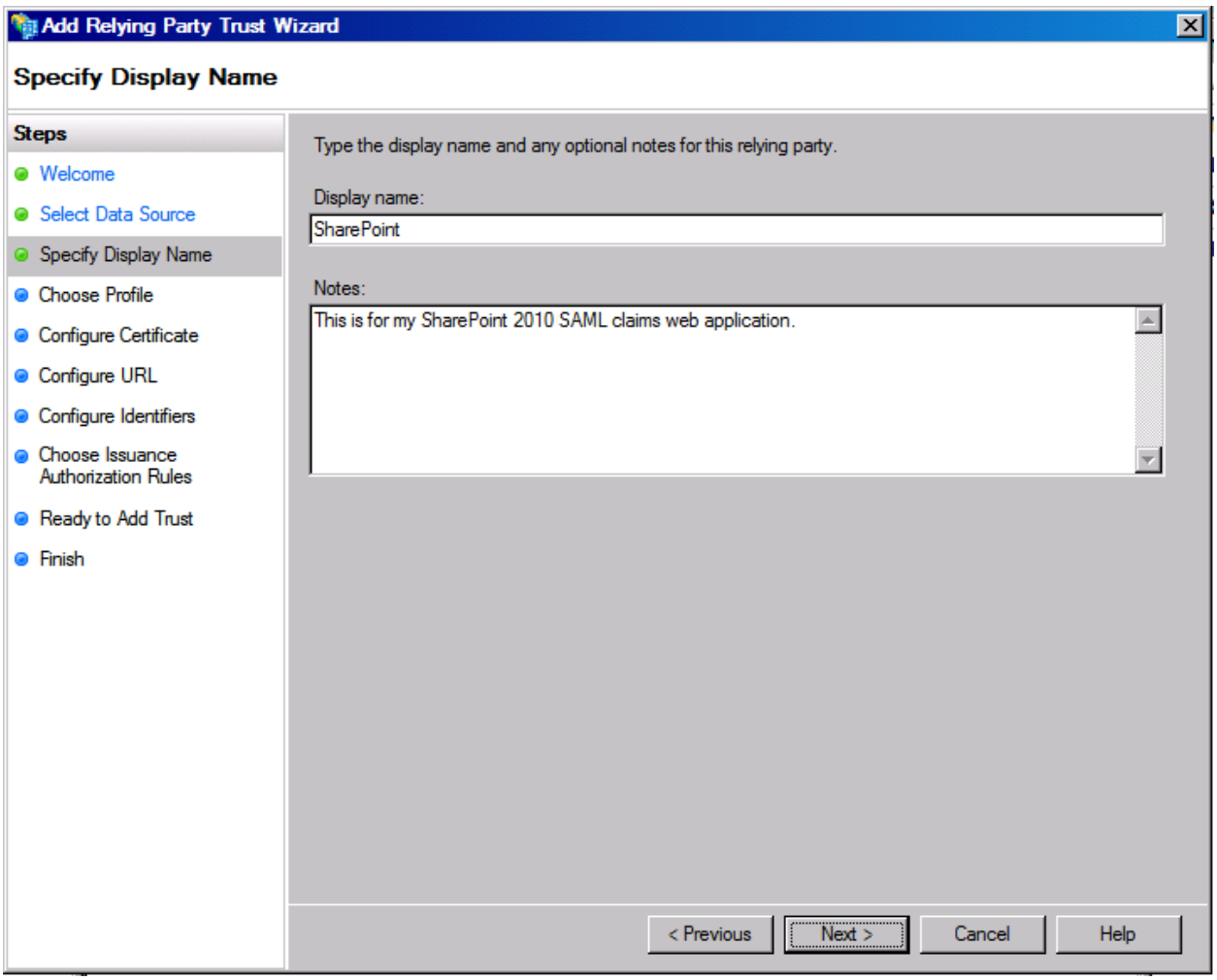
Click **Add Relying Party Trust** in the right pane to start the Add Relying Party Trust Wizard.



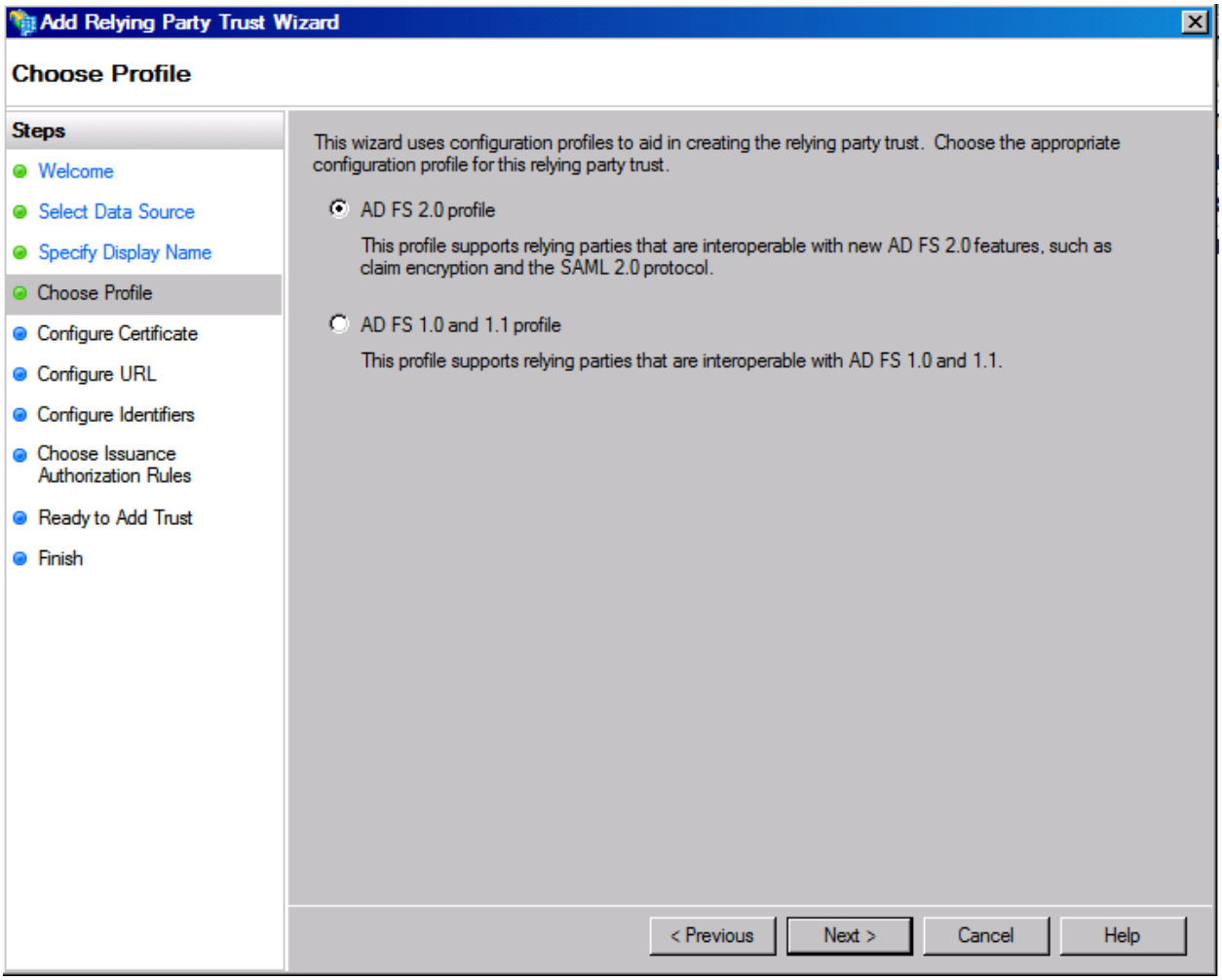
In the Add Relying Party Trust Wizard dialog box, click **Start**.



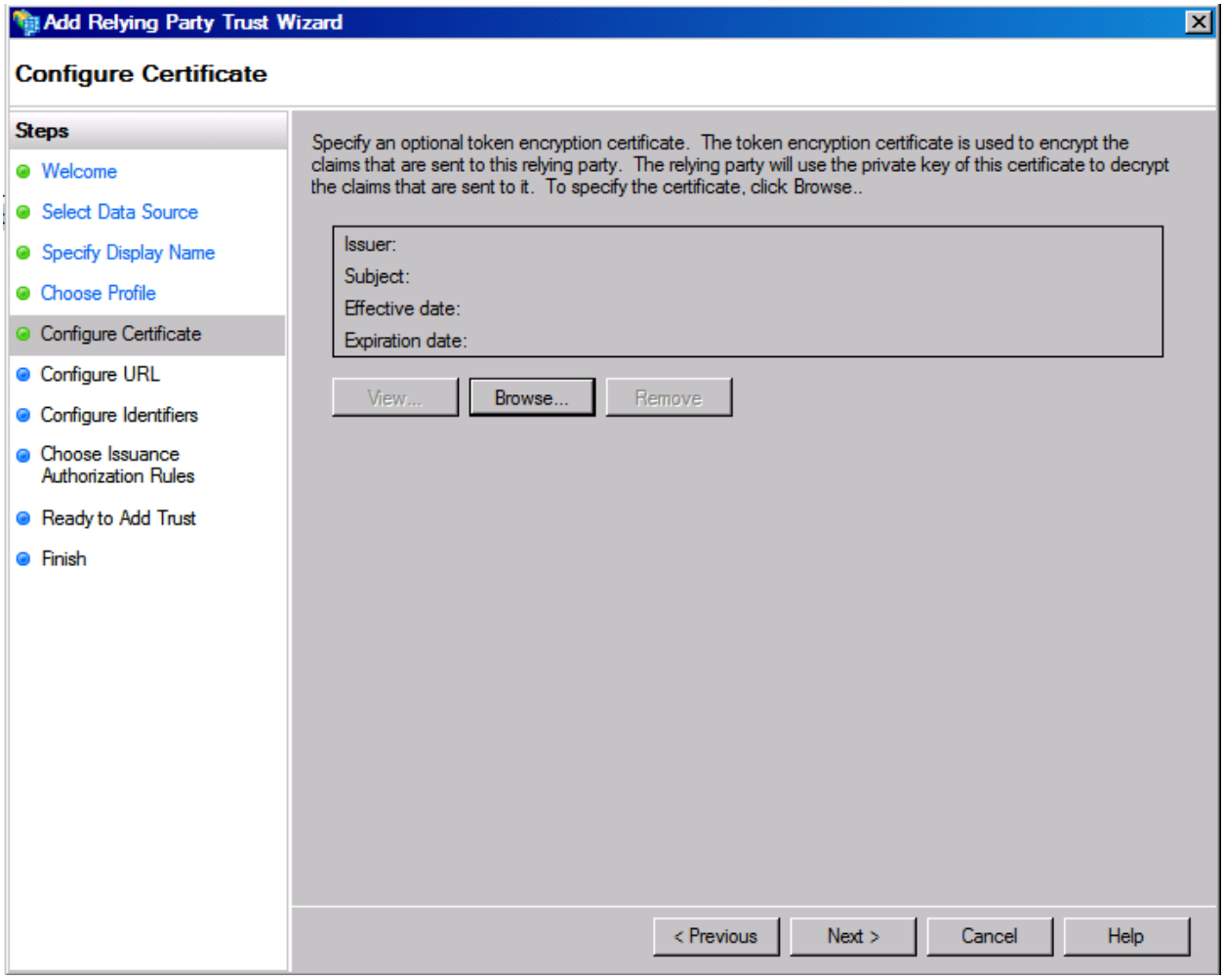
In the Add Relying Party Trust Wizard, select the **Enter data about the relying party manually** option, and then click **Next**.



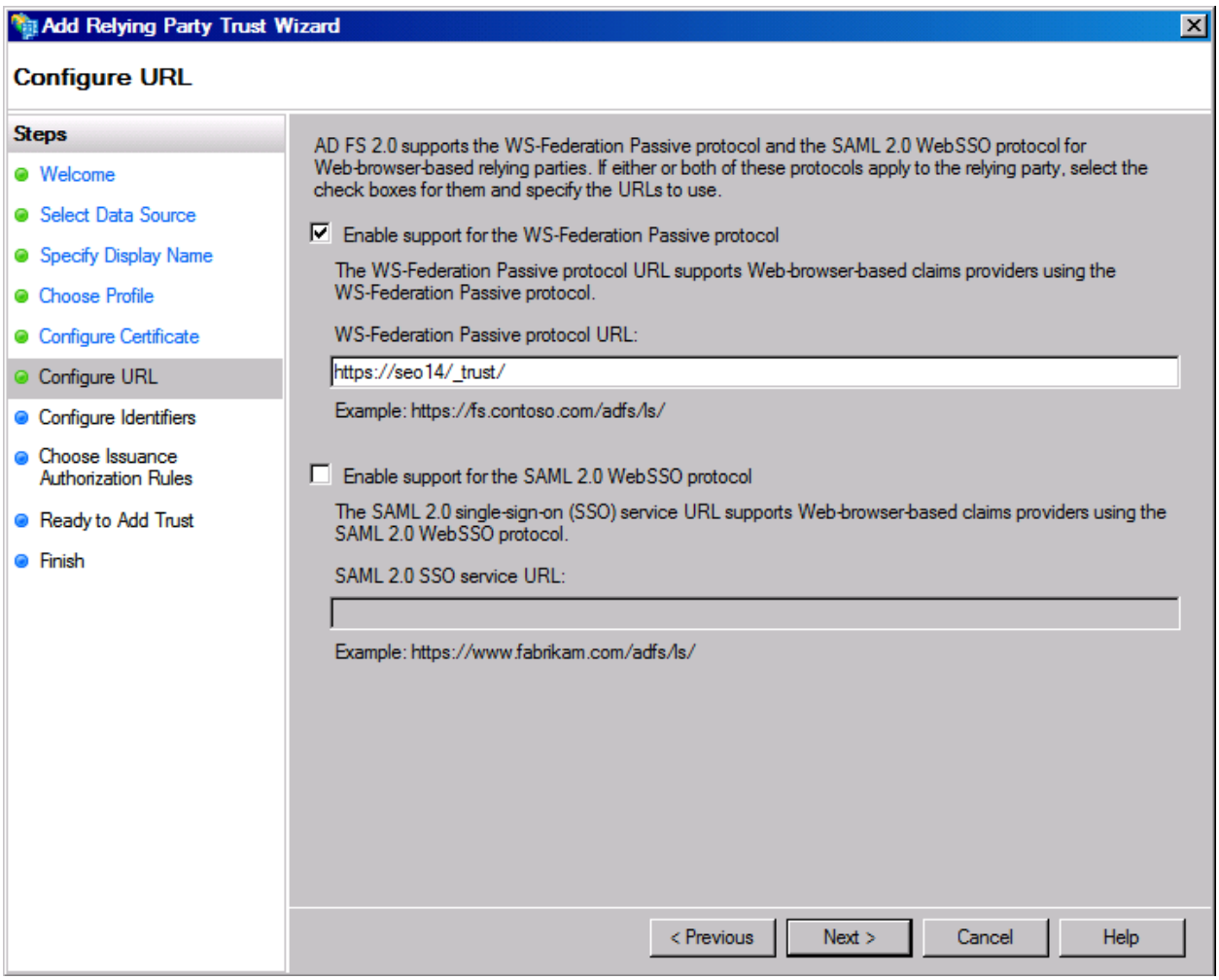
In the Add Relying Party Trust Wizard, enter a display name in the **Display name:** field and optionally a description for the relying party, and then click **Next**.



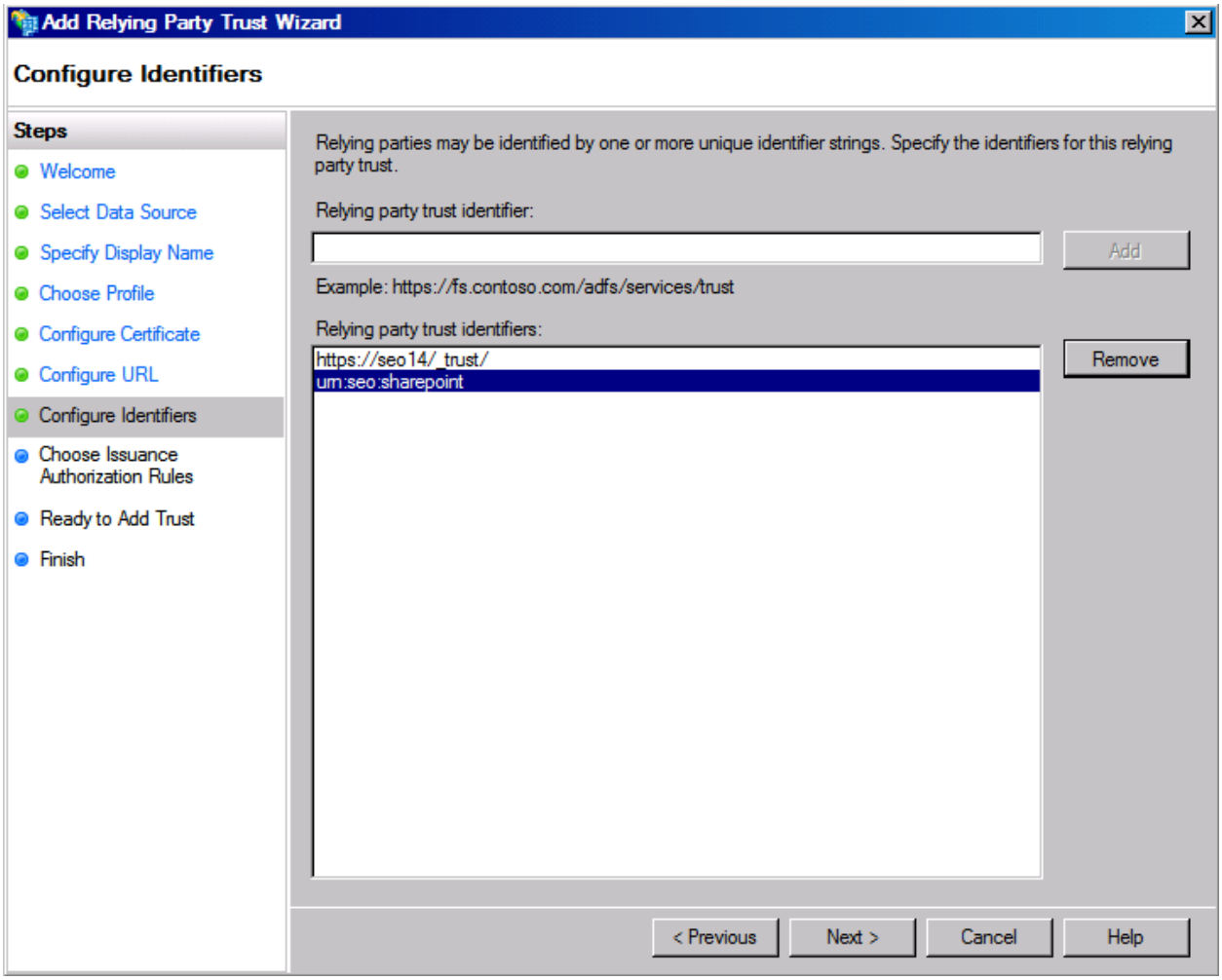
In the Add Relying Party Trust Wizard, select the **AD FS 2.0 profile** option, and then click **Next**.



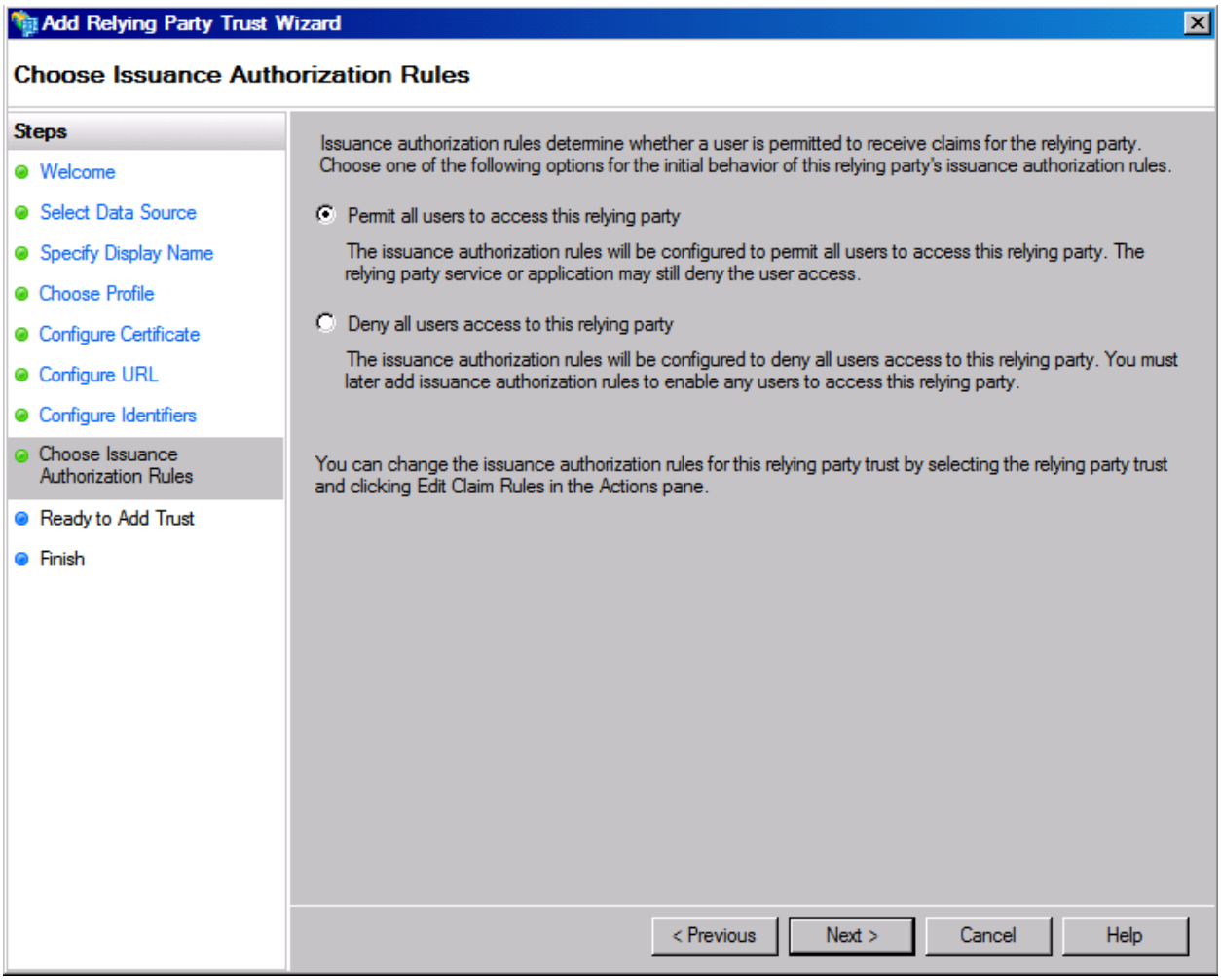
In the Add Relying Party Trust Wizard, you can select a certificate to encrypt the SAML token itself. However, AD FS requires a connection to SharePoint Server over SSL, so the channel the token is sent over is encrypted by default. Click **Next**.



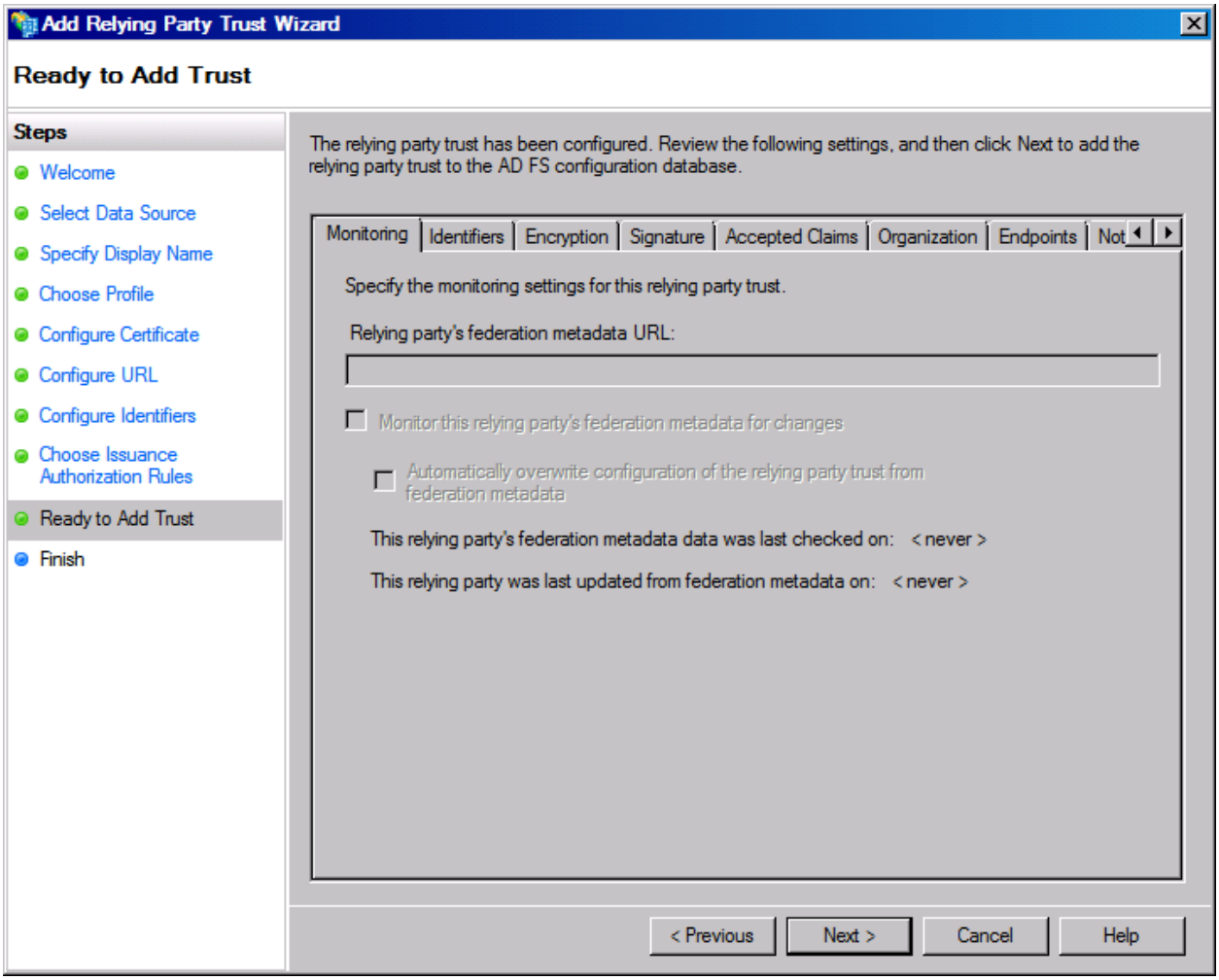
In the Add Relying Party Trust Wizard, select **Enable support for the WS-Federation Passive protocol**. In the **WS-Federation Passive protocol URL:** field, enter the URL for the SharePoint Web application's root site including the "_trust" subdirectory. In this scenario, the URL to the SharePoint Web application is https://seo14. As a result, the WS-Federation Passive protocol URL is https://seo14/_trust/. Click **Next**.



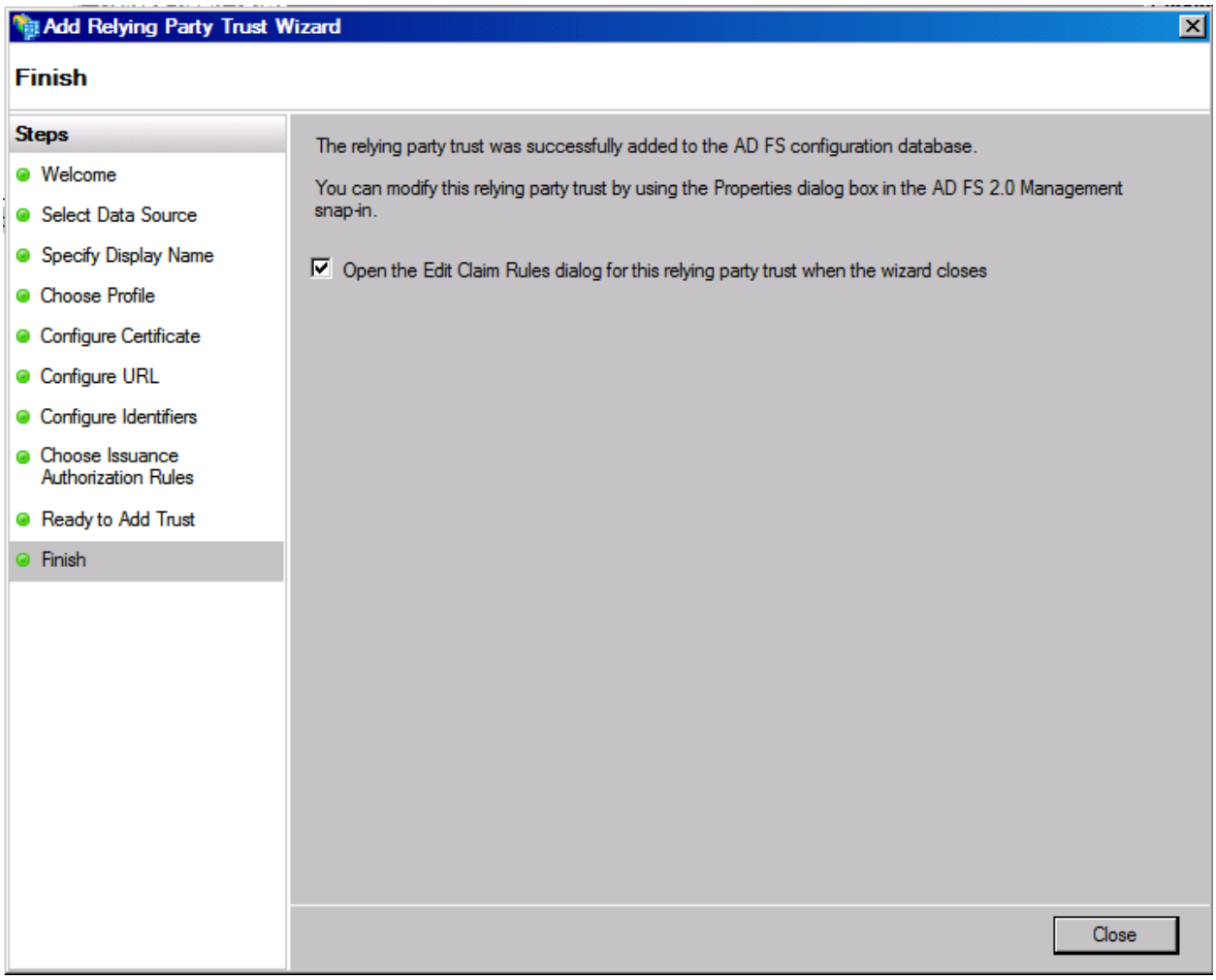
In the Add Relying Party Trust Wizard, enter a realm that the Web application will pass to AD FS when users log into the Web application in the **Relying party trust identified:** field, and then click **Add**. The realm is generally created in the format of urn:sharepoint:collab. The realm is associated with a Web application and is how AD FS maps the login request to the relying party trusts that it has. When used with SharePoint Server, AD FS sees the realm that is associated with the login request and looks that up to find the Relying Party trust. After AD FS authenticates the user, it looks to that WS-Federation Passive protocol URL to determine where to redirect the user afterwards. In this scenario the realm is listed as urn:seo:sharepoint. Therefore, when navigating to https://seo14, users are redirected to AD FS and SharePoint Server will be configured to use the realm, urn:seo:sharepoint, for that request. After AD FS has authenticated a user, it will redirect to https://seo14/_trust/ as the passive protocol URL for that relying party. Click **Next** to continue.



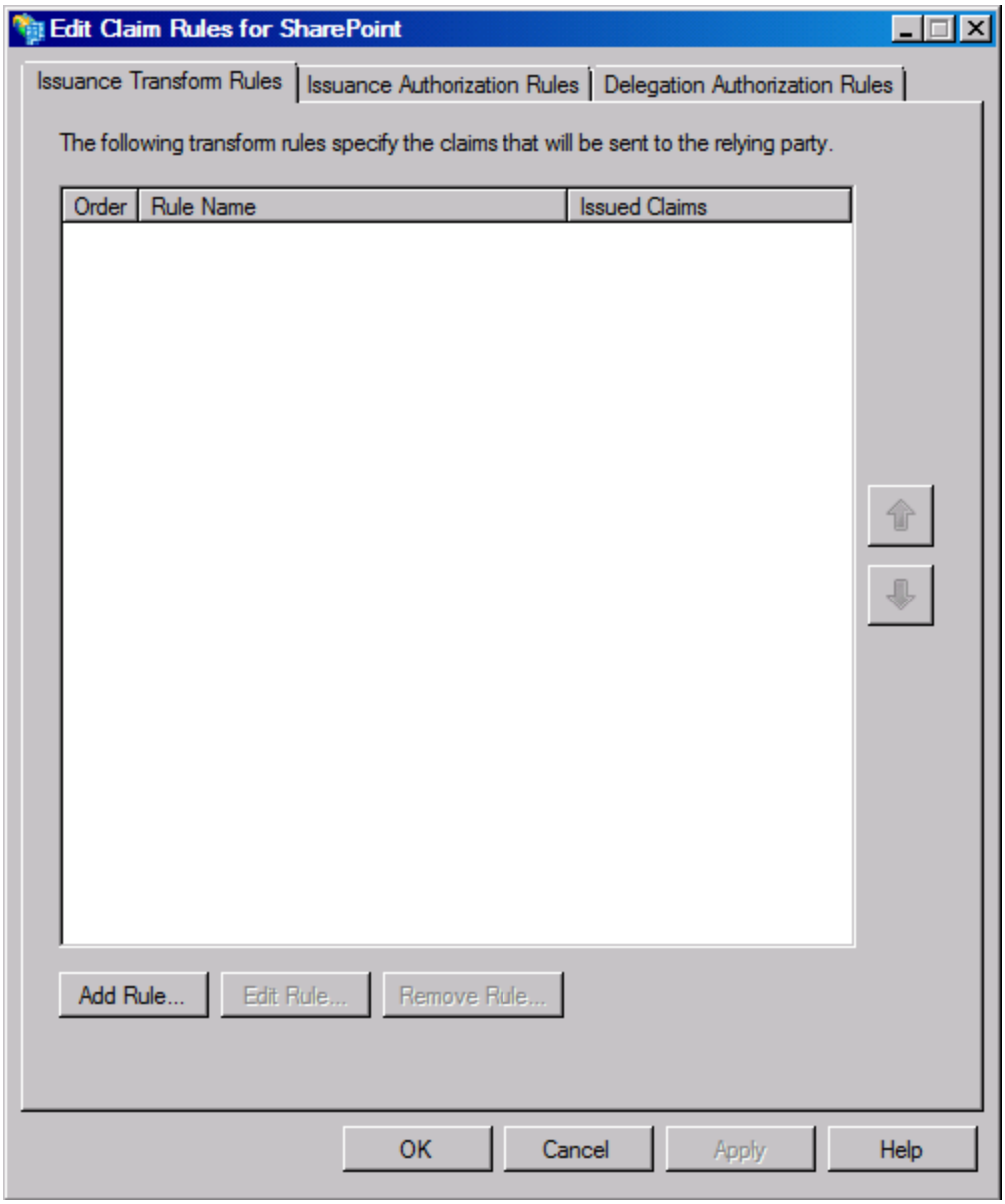
In the Add Relying Party Trust Wizard, select **Permit all users to access this relying party**, and then click **Next**.



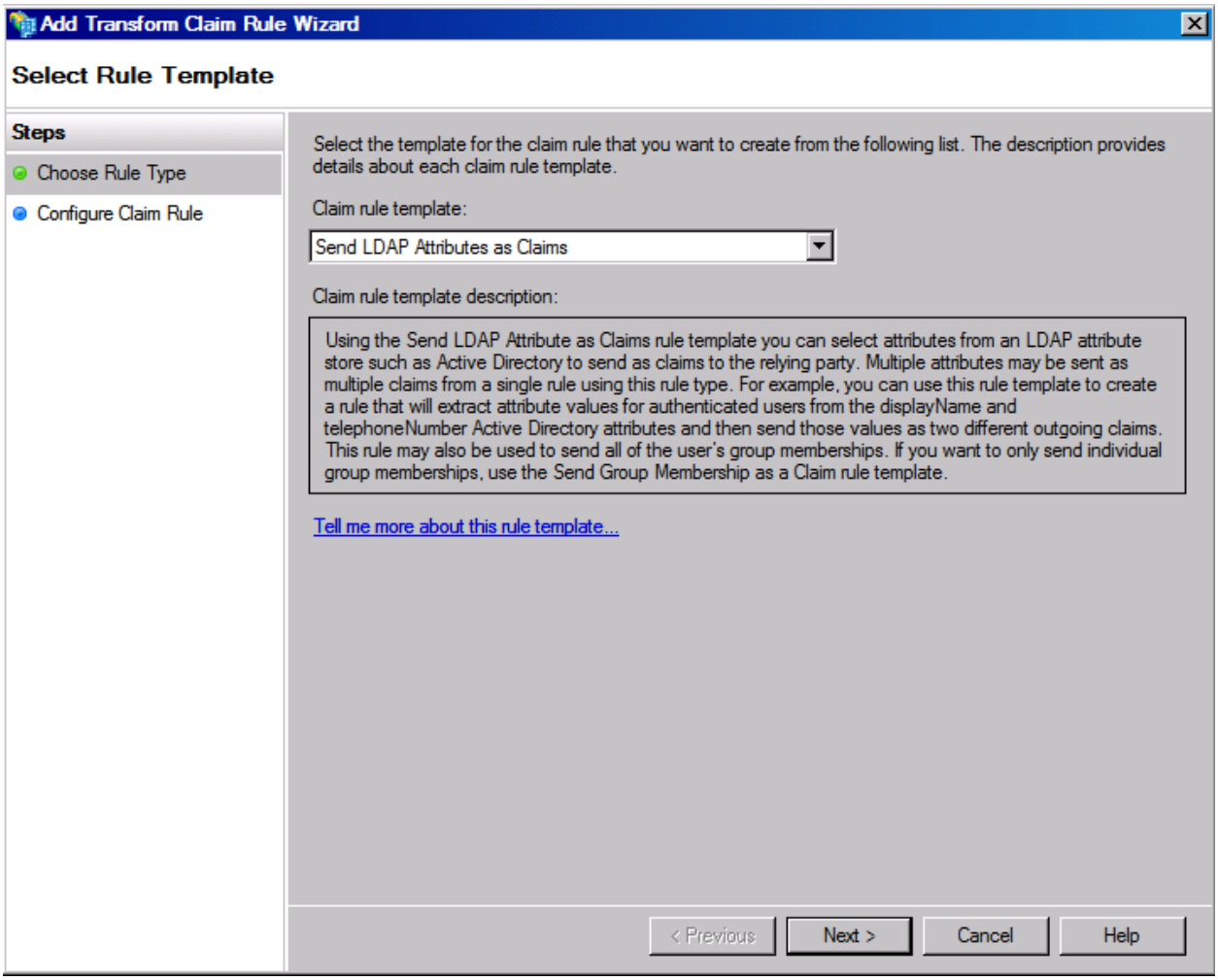
In the Add Relying Party Trust Wizard, click **Next**.



In the Add Relying Party Trust Wizard, select **Open the Edit Claim Rules dialog for this relying party trust when the wizard closes**. Claims rules are required to inform AD FS of the claims that should be submitted to SharePoint Server. Click **Close** to open the **Edit Claim Rules for SharePoint** dialog box.

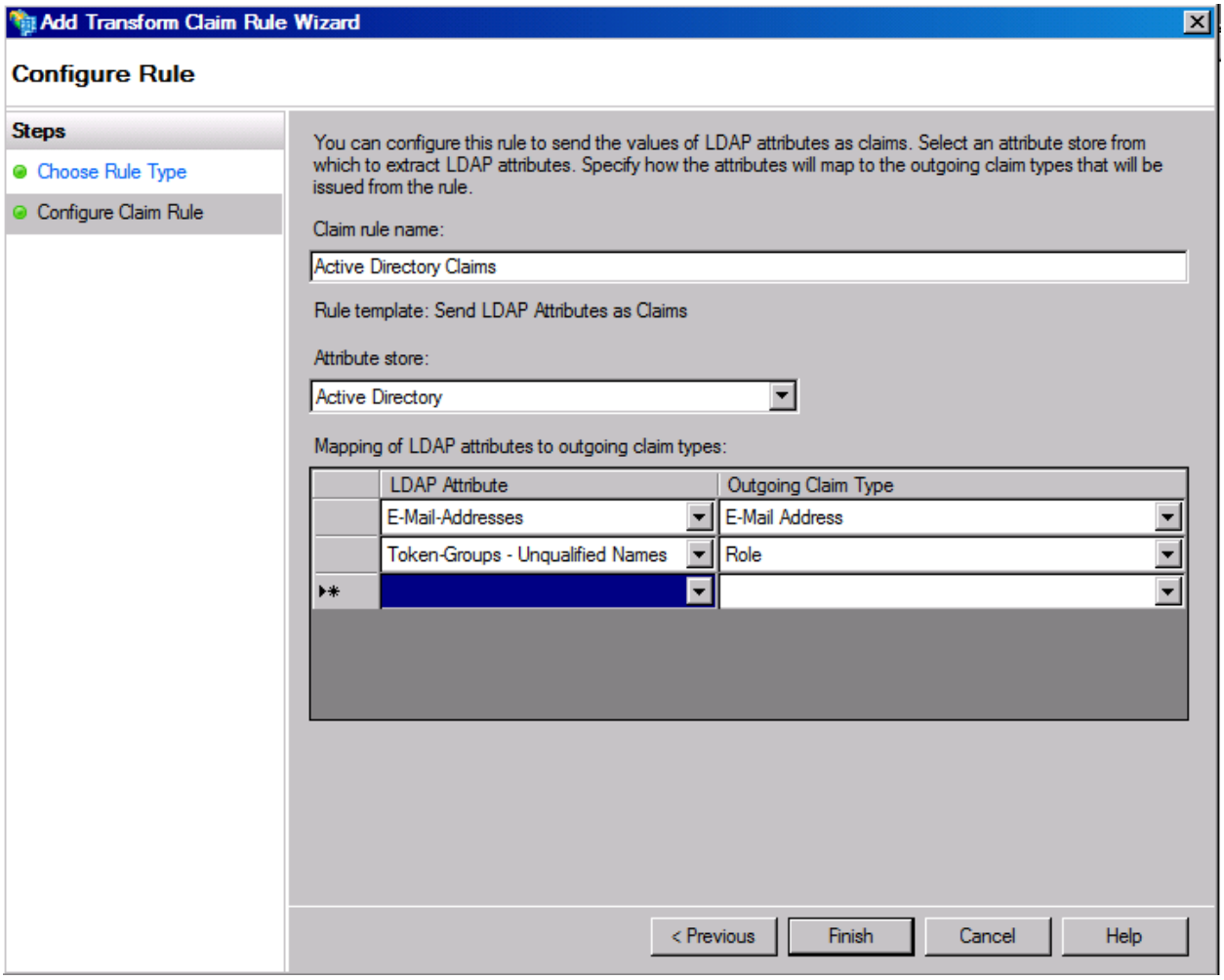


In the Edit Claim Rules for SharePoint dialog, click **Add Rule**.



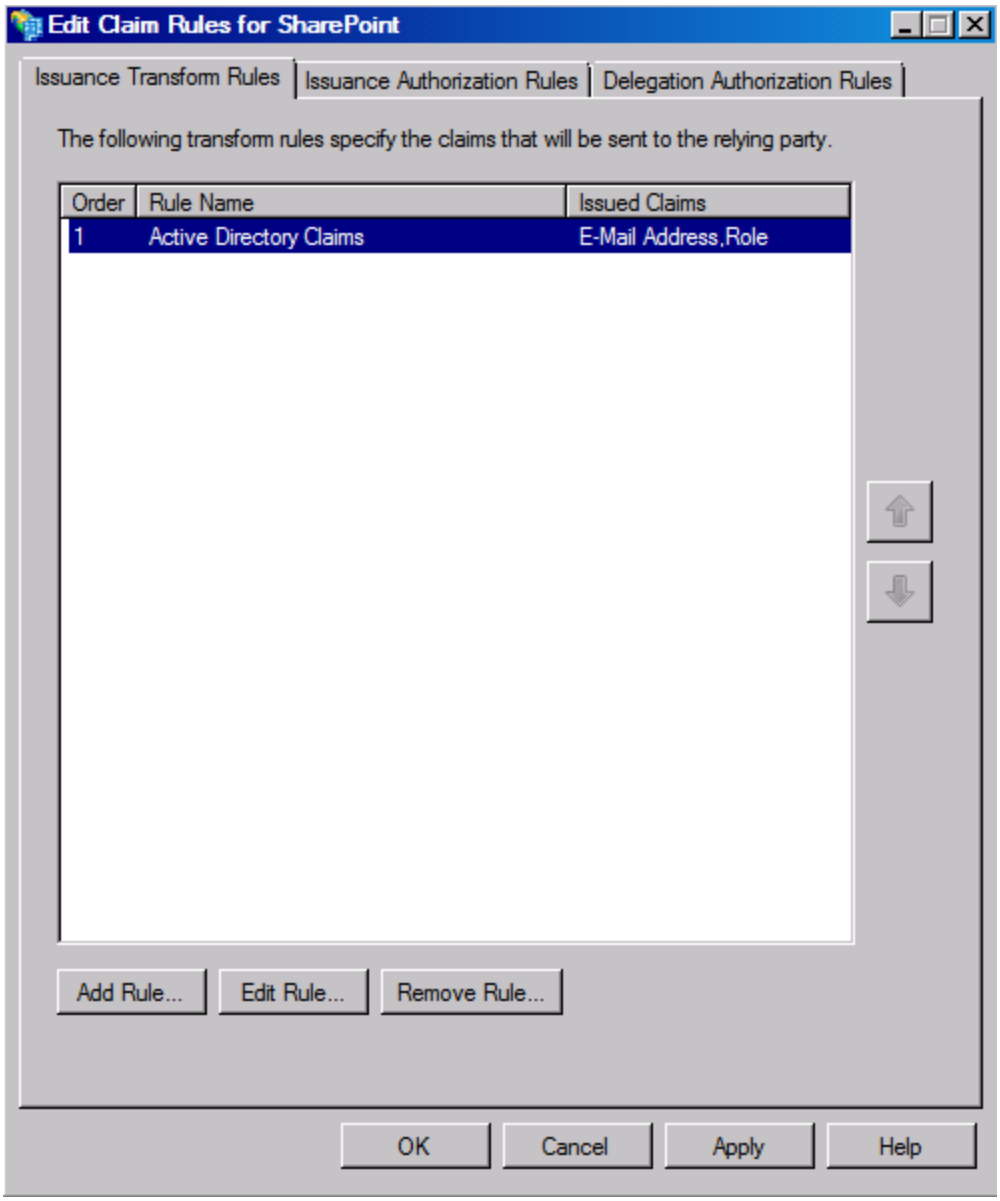
In the Add Transform Claim Rule Wizard, select **Send LDAP Attributes as Claims** from the list of available options, and then click **Next**.

This configuration implies that users will authenticate at AD FS and will to use the corporate Active Directory Domain Services to authenticate the users and determine their attributes.

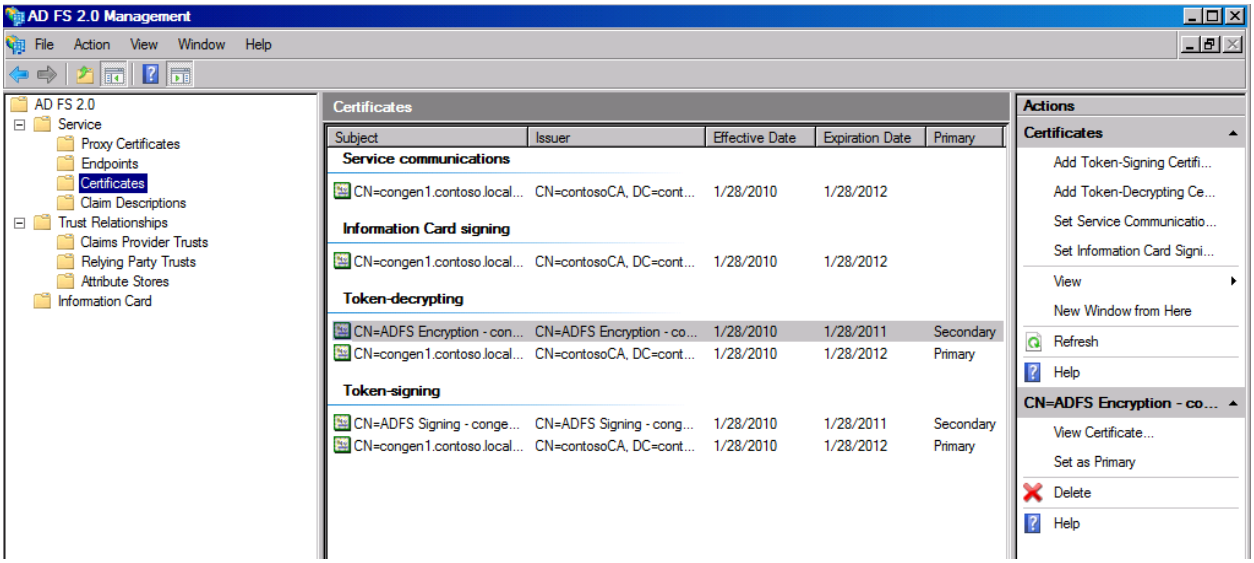


In the Add Transform Claim Rule Wizard, specify a claims rule name in the **Claims rule name:** field, and then select Active Directory Domain Services as the attribute store from the **Attribute store:** menu. In this scenario, a user's e-mail address will be used as the identifier and group memberships submitted in the role claim.

To configure the mapping, select the desired attribute from the menu, and then select the claim that it will be submitted as. In this scenario the E-Mail-Addresses attribute is used from Active Directory Domain Services to be submitted in the standard E-Mail Address claim and the groups to which a user belongs to be submitted in the standard Role claim. This scenario uses Token-Groups – Unqualified Names. and as a result sends the group name as a simple string, for example, the name of the group. Optionally the SID of the groups could be submitted. However, this adds additional complexity when assigning a Role claim to a SharePoint group. Click **Finish** to complete rule creation.

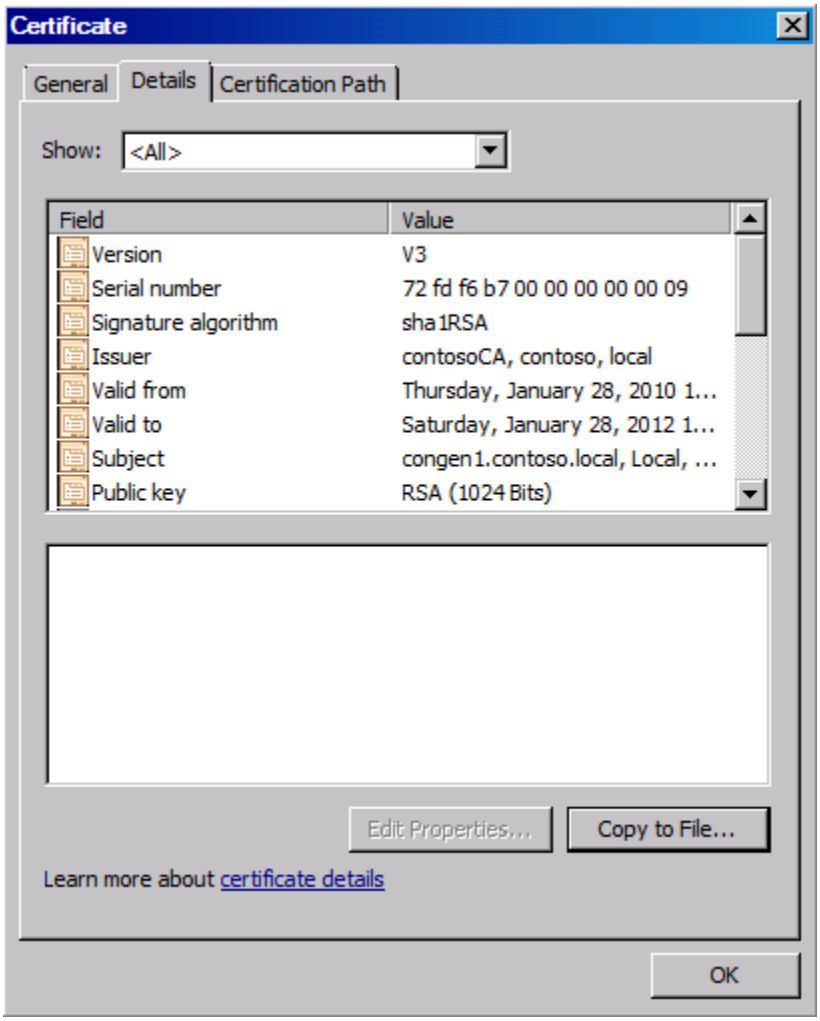


In the Edit Claim Rules for SharePoint dialog box, click **OK** to complete the process of creating the relying party trust in AD FS. AD FS uses a certificate to sign the tokens that it sends to ensure the consumer of the token that it has not been tampered with since it was created. SharePoint Server requires a copy of this certificate when configuring it to use AD FS as the IP-STS. To retrieve the token-signing certificate from AD FS, expand the Service node and click the **Certificates** node in the AD FS 2.0 Management console.



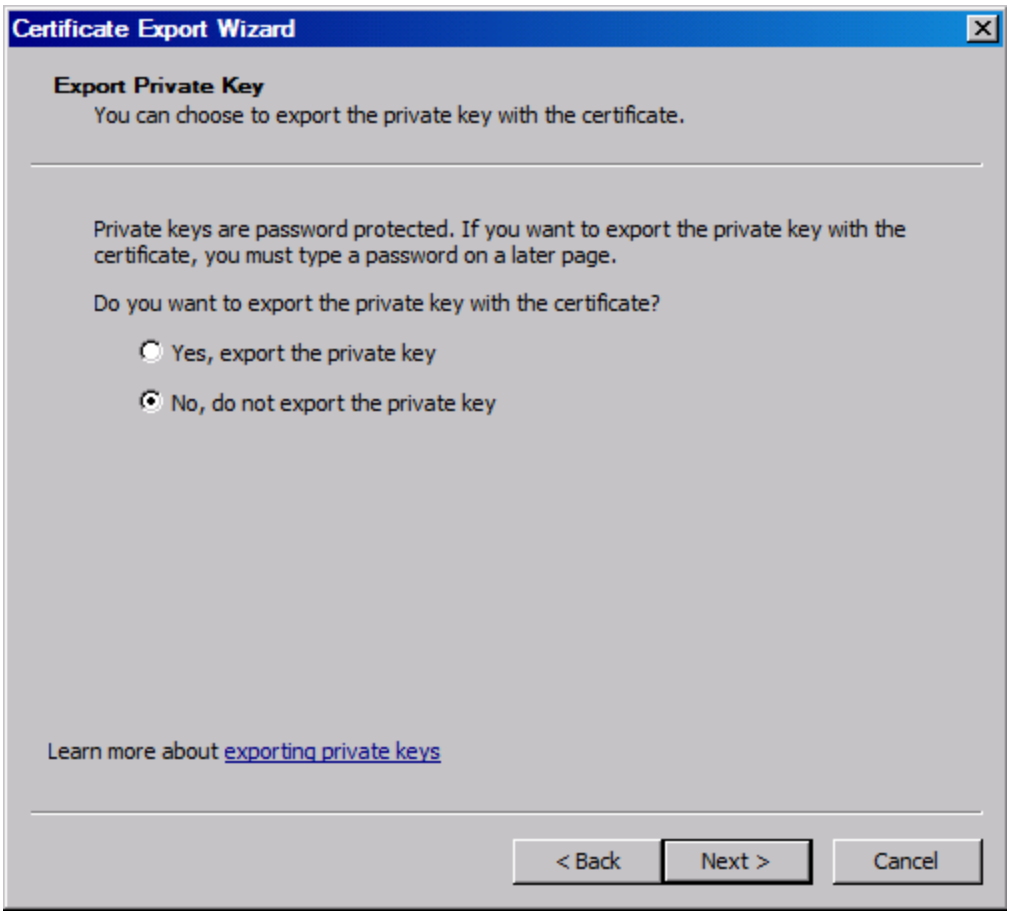
In AD FS 2.0 Management, locate the Token-signing certificates section, select the Primary token-signing certificate, and then select **View Certificate**.

In this scenario, the certificate created for SSL on the AD FS Website is selected. Click the **Details** tab on the Certificate dialog box for the certificate.

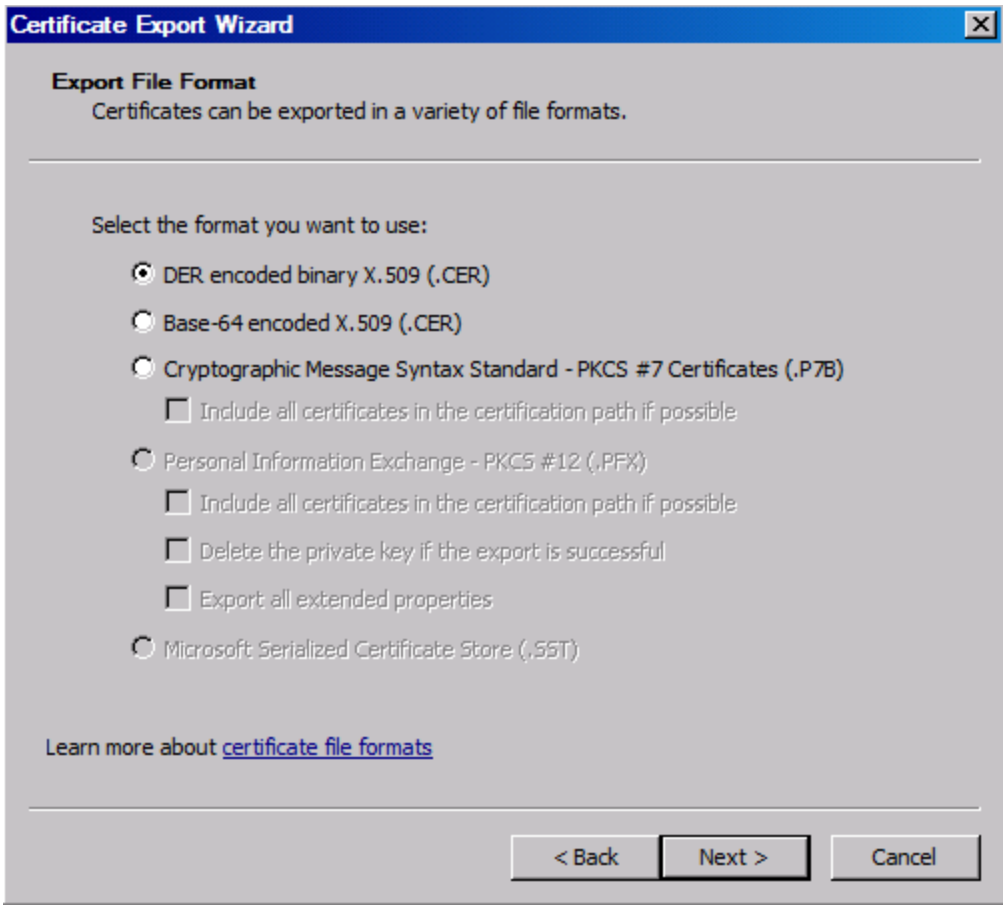


In the Certificate dialog box, click **Copy to File** to start the Certificate Export Wizard.

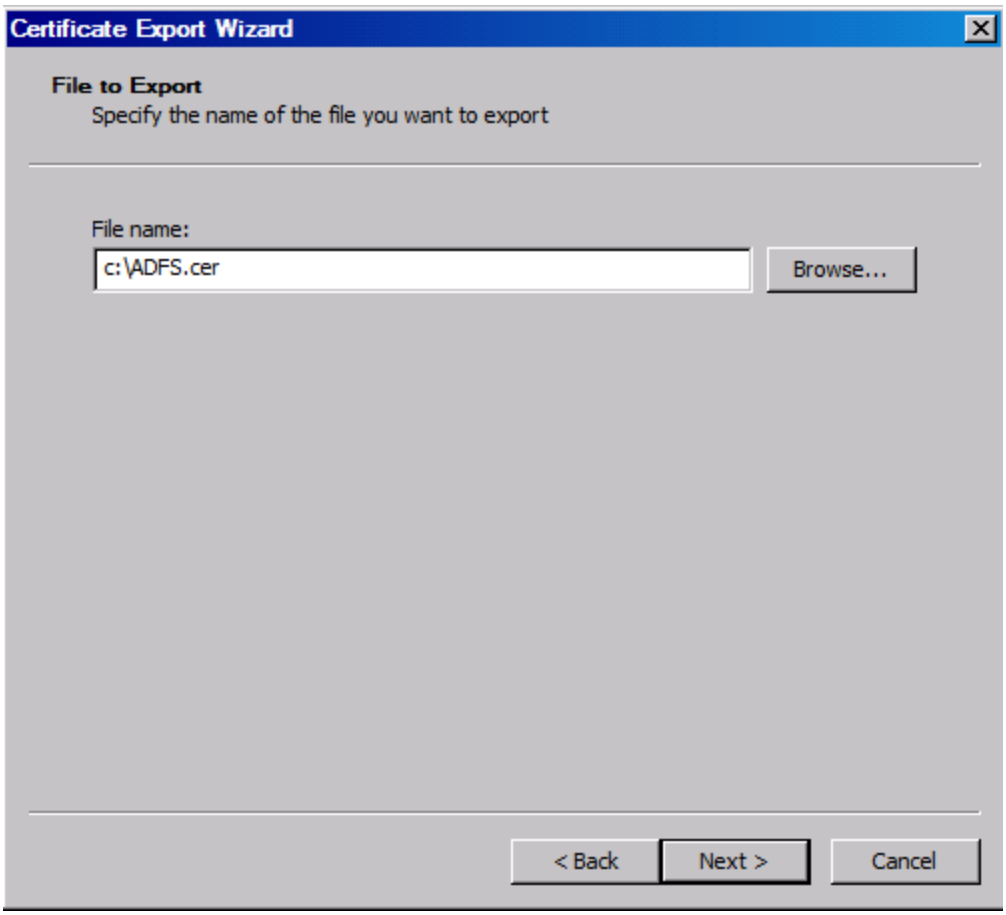
in the Certificate Export Wizard, click **Next**.



In the Certificate Export Wizard, select **No, do not export the private key**, and click **Next**.



In the Certificate Export Wizard, select **DER encoded binary X.509 (.CER)**, and then click **Next**.



In the Certificate Export Wizard, select a location to save the certificate, and then click **Next**.

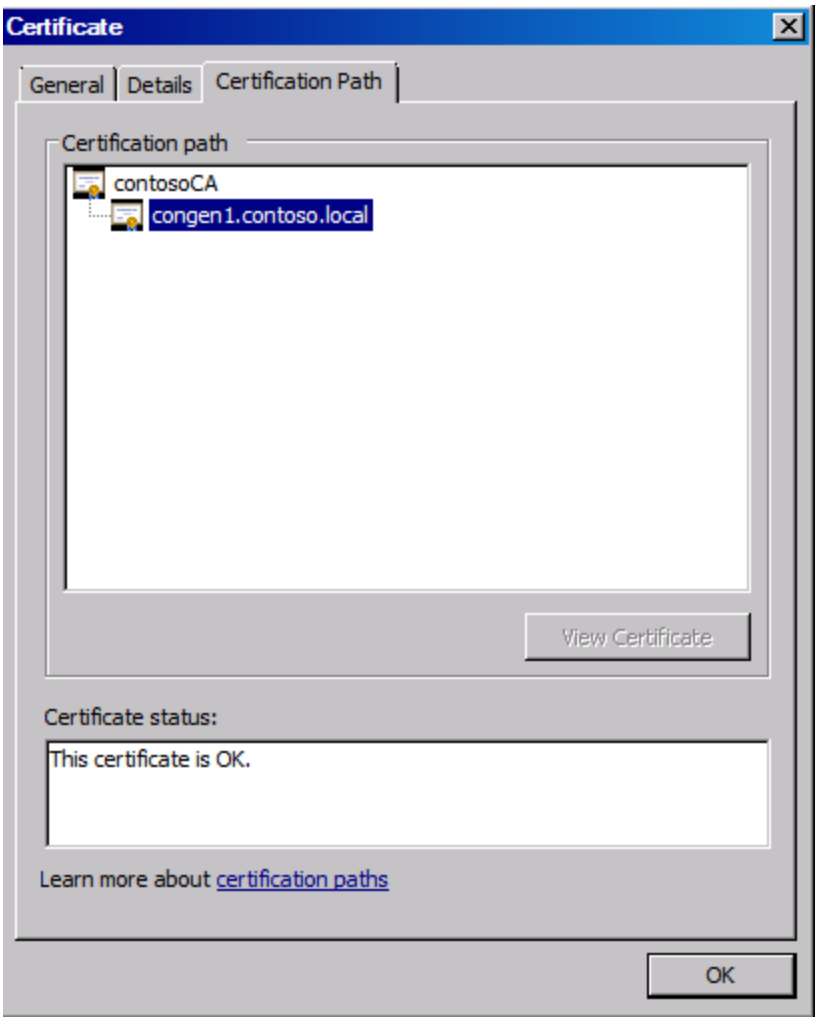
In the Certificate Export Wizard, click **Finish** to save the certificate to a local file. Copy the exported certificate to each server running SharePoint Server.

Configure SharePoint Server 2010

To complete this scenario you should create a new Web application that uses Claims Mode Authentication. Be sure to select Integrated Windows authentication – NTLM in the Authentication Settings. Ensure the Web application uses port 443 and that Use Secure Sockets Layer (SSL) is selected when creating the Web application. Ensure the correct bindings for the new virtual server (Web application) in IIS Manager to enable the assignment appropriate SSL certificate.

If the certificate has one or more parent certificates in its chain, each must be added to the list of SharePoint trusted root authorities.

To check the certificate chain, locate the token signing certificate copied over from AD FS in the previous section and double-click on it to open certificate properties window. On the Certification Path tab, determine if there are any other certificates in the chain. In this scenario, the token signing certificate has a parent – the root certificate authority certificate.



For each certificate in the chain a local copy is required. In the Certificates dialog box, select the appropriate certificate, and then click **View Certificate**. Export each certificate in the chain using the steps described previously.

Add each certificate to the list of trusted root authorities by using the following Windows PowerShell commands:

Windows PowerShell Command

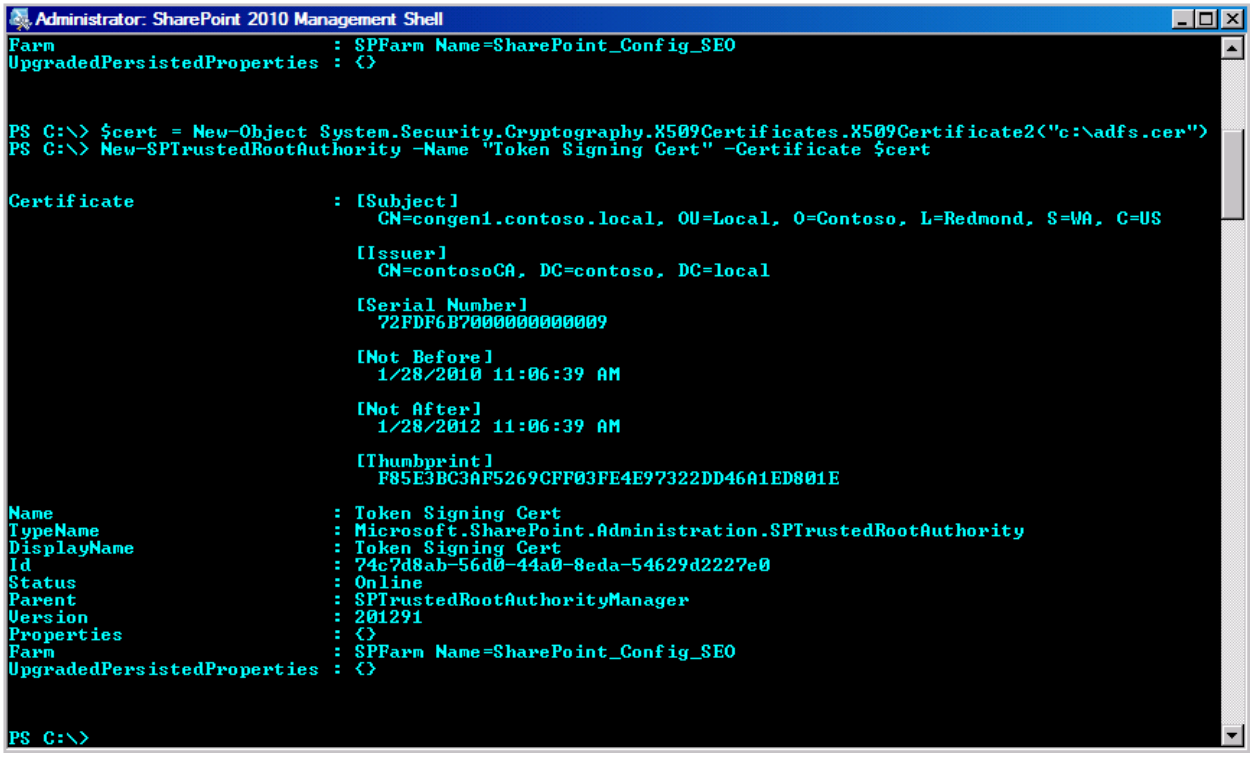
```
$root = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2("C:\AD
FSParent.cer")

New-SPTrustedRootAuthority -Name "Token Signing Cert Parent" -Certificate $root

$cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2("C:\AD
FS.cer ")

New-SPTrustedRootAuthority -Name "Token Signing Cert" -Certificate $cert
```

The following image illustrates the output generated by the previous Windows PowerShell command and is included for illustration purposes only.



Create the claims mappings to be used by SharePoint Server by using the following Windows PowerShell command:

Windows PowerShell Command

```
$map = New-SPClaimTypeMapping -IncomingClaimType
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress" -
IncomingClaimTypeDisplayName "EmailAddress" -SameAsIncoming

$map2 = New-SPClaimTypeMapping -IncomingClaimType
"http://schemas.microsoft.com/ws/2008/06/identity/claims/role" -
IncomingClaimTypeDisplayName "Role" -SameAsIncoming
```

Use the following Windows PowerShell command to create a variable for the realm that SharePoint Server will use:

Windows PowerShell Command

```
$realm = "urn:seo:sharepoint"
```

Create the SPTrustedIdentityTokenIssuer which correlates the information that SharePoint Server uses to connect and work with the IP-STS by using the following Windows PowerShell command:

Windows PowerShell Command

```
$ap = New-SPTrustedIdentityTokenIssuer -Name "SAML Provider" -Description "SharePoint
secured by SAML" -realm $realm -ImportTrustCertificate $cert -ClaimsMappings $map,$map2 -
SignInUrl "https://congen1.contoso.local/ADFS/ls" -IdentifierClaim
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"
```

The "Name" attribute describes which authentication provider a Web application is configured for.

The "Realm" attribute defines the realm to be used by the trusted identity token issuer.

The "ImportTrustCertificate" attribute is what is passed to the token signing certificate copied from the AD FS server in this scenario.

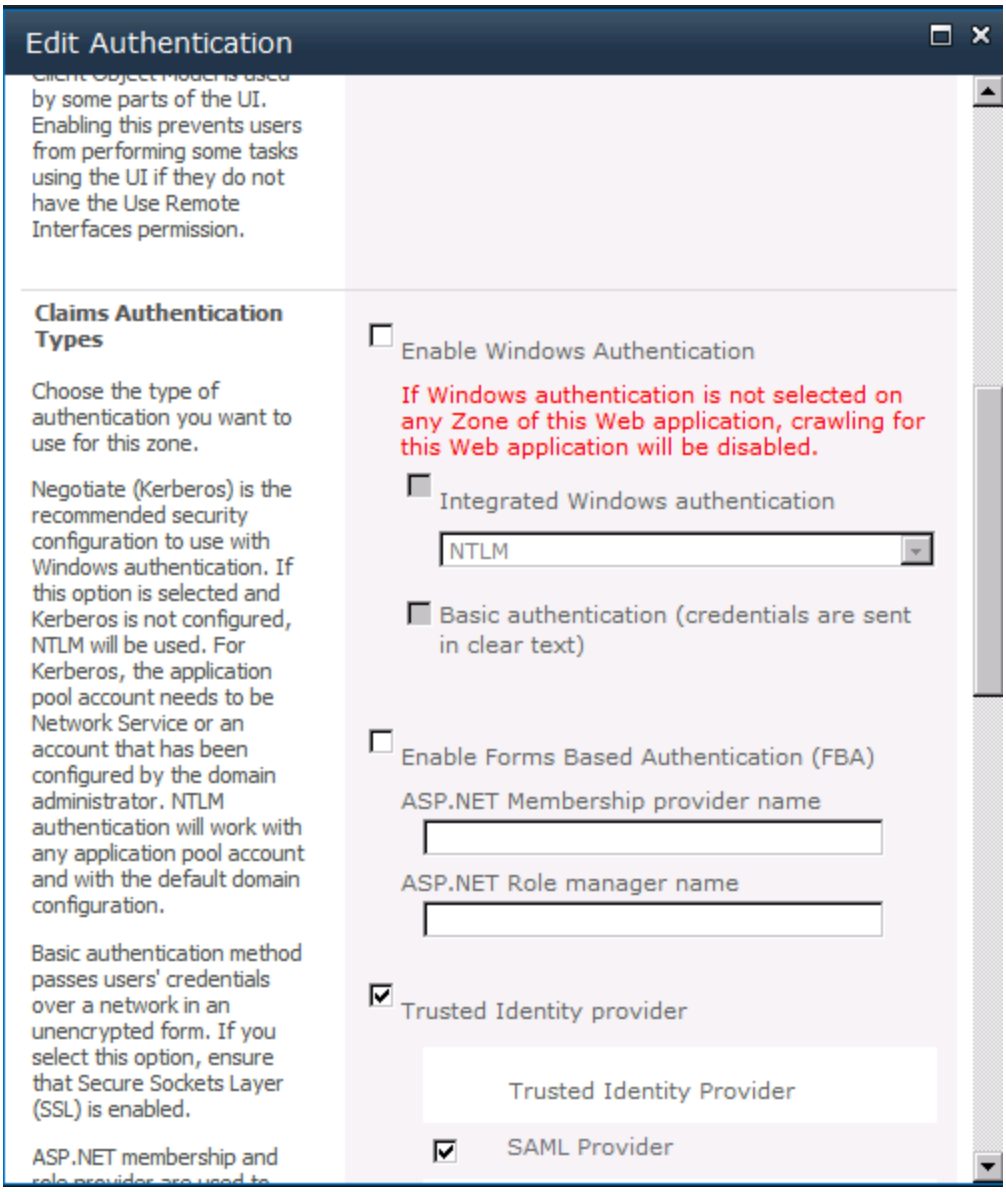
The "ClaimsMappings" attribute are the claims the trusted identity token issuer will use.

The "SignInUrl" is the URL that users should be redirected to authenticate with the IP-STS. In this scenario, users authenticate with the AD FS server by using Windows integrated security, so they are redirected to the /ADFS/ls subdirectory.

The "IdentifierClaim" attribute instructs SharePoint Server which of the claims will be the claim used to identify users. In this scenario the e-mail address is used to identify a user.

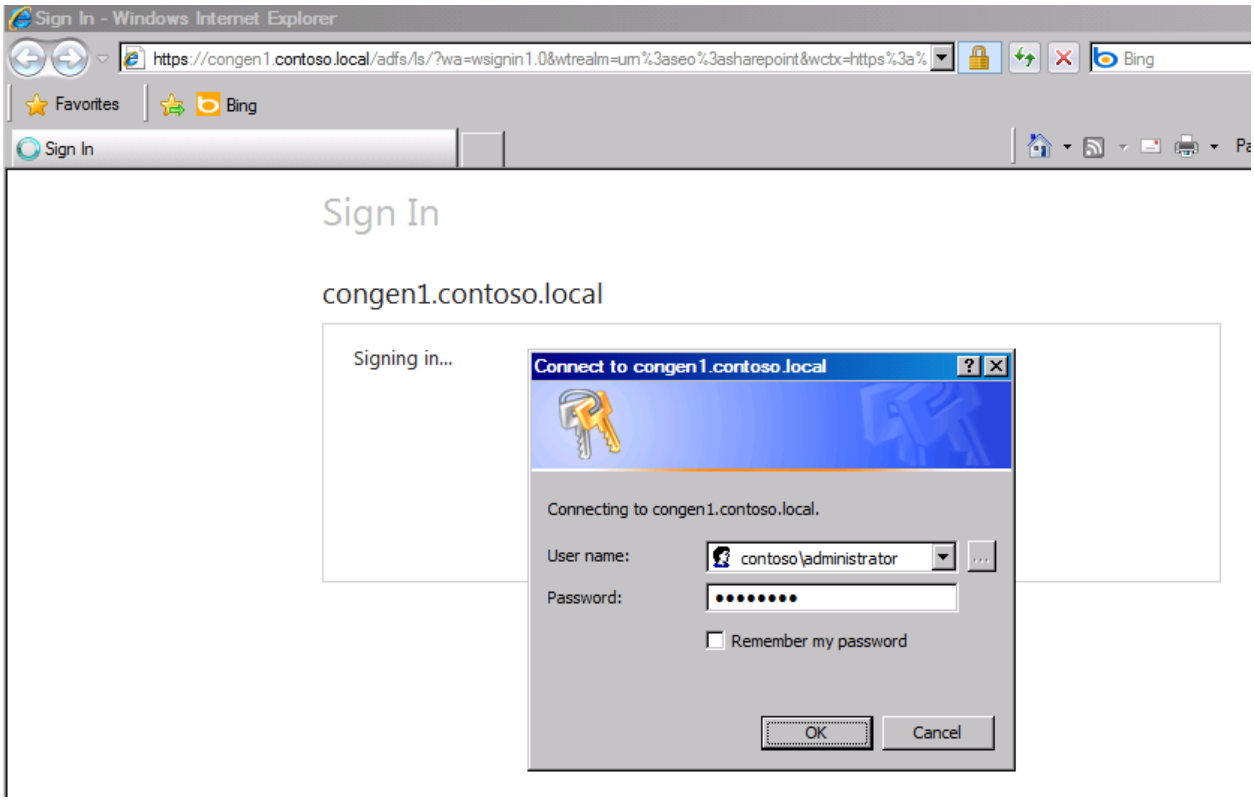
After creating the SPTrustedIdentityTokenIssuer used with the SharePoint Web application by using the Windows PowerShell command above, navigate to Central Administration click **Manage Web Applications**, and then select the Web application that will use AD FS to authenticate. **Click Authentication Providers** on the ribbon.

Click the link in the dialog box that corresponds to the zone in which to use AD FS to authenticate. Scroll down to the Authentication Types section and uncheck the NTLM option. Select **SAML Provider** from this list of trusted providers and then click **Save**.



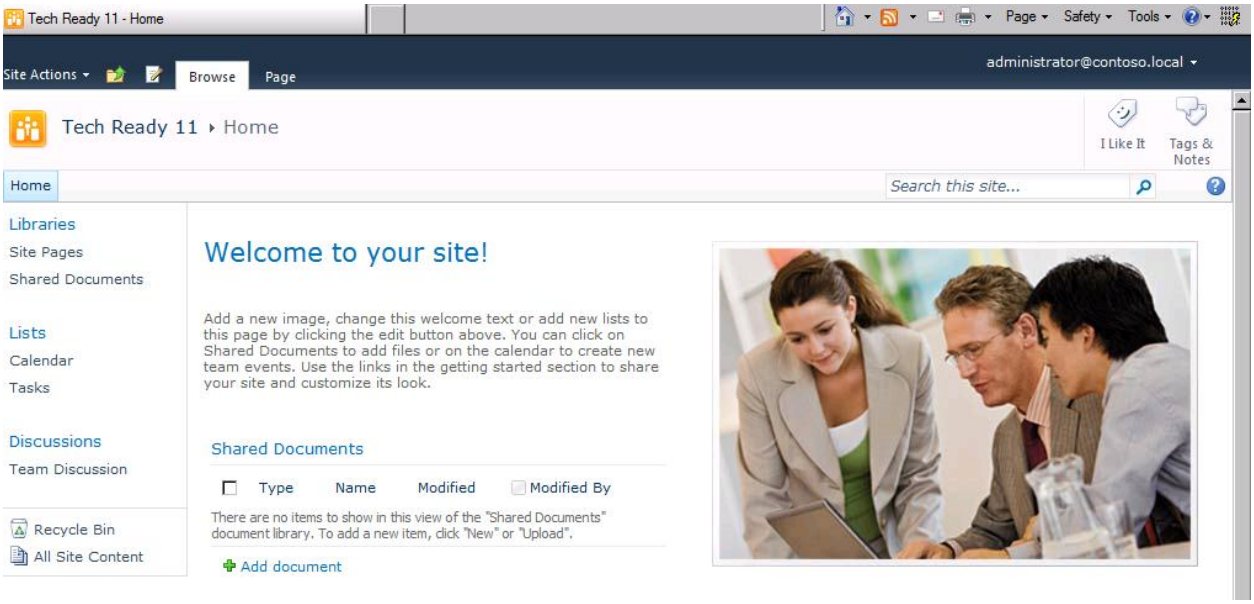
Create a new top-level site collection for the Web application and add a new user by using the identity claim format. In this scenario, the e-mail address is the identity claim, for example, administrator@contoso.local.

Open a new browser window and type https://seo14 (specific to this scenario) which will be redirected to the SignInUrl for the SPTrustedIdentityTokenIssuer associated with the Web application.



The URL in the browser window will represent the AD FS server. Authentication will occur by using Windows Credentials against the AD FS server as opposed to SharePoint Server.

SharePoint Server in this scenario is configured to use an e-mail address as the identity. However, users will authenticate over AD FS which will use the claim rule created to extract the e-mail address and groups and insert them into claims that will be sent back to SharePoint Server. After authentication, users are redirected back to SharePoint Server at https://seo14/_trust/ (configured in the Relying Party configured in AD FS in this scenario). At that point, SharePoint Server will complete the authentication process as it takes the claim that it receives in the SAML token and converts it to an SPUser, arriving at the home page for the top-level site collection.



NOTE

The login control is displaying the identity as an e-mail address as a result of it being the identity claim.

Creating Multiple Claims Authentication Web Applications in a Single SharePoint Server 2010 Farm

Creating multiple Web applications that use claims-based authentication in a single SharePoint Server 2010 server farm environment enables an organization to take advantage of multiple relying parties while using a single certificate. One primary point of confusion about how to configure multiple Web applications that use claims-based authentication in SharePoint Server 2010 is around the **SPTrustedIdentityTokenIssuer** object. You can associate a token-signing certificate only from a security token service (STS) with one **SPTrustedIdentityTokenIssuer** object. When you create your **SPTrustedIdentityTokenIssuer** object, you inform the **SPTrustedIdentityTokenIssuer** object about the following:

- Token signing certificate that you are using
- Realm that you are using

The realm is what is included in the query string that is sent back to your STS. The STS uses the realm to determine the appropriate relying part so that STS knows the claim rules to process, the URL to use to look up the trust policy for the Web application, and so on. While it is possible to add multiple token-signing certificates to Active Directory Federation Services (AD FS) 2.0 as an example, there is no way to instruct that a particular token-signing certificate should be used with a particular relying party. Therefore, you have to find a way to make it work with the single certificate.

The **SPTrustedIdentityProvider** object has a **ProviderRealms** property that takes multiple realms. For example, assume two Web applications: https://collab and https://mysites. To support this configuration, a Windows PowerShell command would be implemented to create a **SPTrustedIdentityTokenIssuer** resembling the following:

Windows PowerShell Command

```
$realm = "urn:sharepoint:collab"

$ap = New-SPTrustedIdentityTokenIssuer -Name "AD FS v2" -Description "AD FS v2" -Realm $re
alm -ImportTrustCertificate $cert -ClaimsMappings $map -SignInUrl "https://URLToYourAD FSS
erver/ADFS/ls" -IdentifierClaim http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emai
laddress
```

The **SPTrustedIdentityTokenIssuer** object is now created and has a default realm of urn:sharepoint:collab. In the previous example, a relying party is created in AD FS 2.0 by using the identifiers, urn:sharepoint:collab and https://collab/_trust/. To support an additional Web application, another realm needs to be added to the **SPTrustedIdentityTokenIssuer** object by using Windows PowerShell as illustrated below:

Windows PowerShell Command

```
$uri = new-object System.Uri("https://mysites")

$ap.ProviderRealms.Add($uri, "urn:sharepoint:mysites")

$ap.Update()
```

The URI should be the URL to the Web application that will use that realm. During authentication, SharePoint Server performs a lookup to locate the realm that is associated with the URI of that Web application which will be the URI that SharePoint Server uses. In this case, the realm urn:sharepoint:mysites is to be used with the Web application at https://mysites. Therefore, that URI was specified when adding the realm. A second relying party using the identifier urn:sharepoint:mysites and https://mysites/_trust/ is required to be configured in AD FS 2.0 to complete the configuration.

Setting the Login Token Expiration Correctly for SharePoint Server 2010 SAML Claims Users

This section is intended to describe the logon process for expiring login cookies. After SAML claims users retrieve their login cookie from their AD FS STS, they would never seem to time out. As a result, they could close the browser, and several minutes or even hours later open the browser again and just navigate directly to the site without having to authenticate to AD FS again. In addition the Microsoft Office 2010 client applications work in a similar way.

When navigating to a SharePoint site that is secured with SAML claims for the first time, a user is redirected for authentication and retrieval of claims. The SAML identity provider (IP-STS) manages those requests and redirects the user back to SharePoint Server. When the user returns to SharePoint Server, a FedAuth cookie is created which determines whether or not the user has been authenticated. In Office SharePoint Server 2007 and AD FS 1.x, Web SSO cookies were session-based and not saved to disk which is a key scenario differentiator. For example, in that scenario when users closed their browsers, the cookies were destroyed which required the user to authenticate again each time the browser was closed and opened.

To configure a manageable lifetime with SAML tokens:

1. The TokenLifetime property can be set per Relying Party in AD FS.

2. Windows PowerShell can be used to set the TokenLifetime of AD FS when creating the relying party. See the following Windows PowerShell command for an example of setting the TokenLifetime of AD FS:

   Windows PowerShell Command

```
Add-AD FSRelyingPartyTrust -Name "FC1" -Identifier "https://fc1/_trust/" -
WsFedEndpoint "https://fc1/_trust/" -TokenLifetime 2 -SignatureAlgorithm
http://www.w3.org/2000/09/xmldsig#rsa-sha1
```

   After creating the Relying party by using Windows PowerShell to set the TokenLifetime, the following manual steps are required:

   a. Add the realm to the list of identifiers (for example. urn:sharepoint:xxx)
   b. Add an Issuance Authorization Rule to permit access to all users
   c. Add an Issue Transform Rule to send over email address and roles

3. To prevent logon looping, the LogonTokenCacheExpirationWindow should be configured to a value to be less than the SAML TokenLifetime. The following Windows PowerShell example illustrates how to set the TokenLifetime in SharePoint:

   Windows PowerShell Command

```
$sts = Get-SPSecurityTokenServiceConfig

$sts.LogonTokenCacheExpirationWindow = (New-TimeSpan –minutes 1)

$sts.Update()

Iisreset
```

This section represents the most common cause of authentication looping and the most common way to resolve it.

Setting up a Federation Trust between AD FS 2.0 and SharePoint Server 2010

For information about how to set up a federation trust between AD FS 2.0 and SharePoint Server 2010, see [How to configure AD FS v 2.0 in SharePoint Server 2010](http://go.microsoft.com/fwlink/?LinkId=228908) (http://go.microsoft.com/fwlink/?LinkId=228908).

Creating a Custom Claims Provider

Overview of Claims Providers

A claims provider accesses a Security Token Service (STS) to encapsulate sets of claims assertions into security tokens. The security tokens are then processed to enable users to access SharePoint Web application resources. When a user signs in to SharePoint Foundation 2010 or SharePoint Server 2010, the user's token is validated and then used to log on to SharePoint Foundation 2010 or SharePoint Server 2010.

A claims provider in SharePoint Server 2010 is used primarily for two reasons:

- To augment claims
- To provide name resolution

Claims augmentation is part of a process that occurs after you log on to a claims authentication site. It is important to remember that the following can run with claims authentication in SharePoint Server 2010:

- Windows claims (when you log on with NTLM or the Kerberos protocol)
- Forms-based authentication claims (when you use an ASP.NET membership and role provider)
- Security Assertions Markup Language (SAML) claims (when you log on by using a security token service (STS), such as Active Directory Federation Services (AD FS) 2.0)

After a user logs on, all the registered claims providers for that Web application are triggered. When the claims providers are triggered, they can add claims that were not provided at logon time. For example, assume that a user logged on by using Windows authentication. However, you would like to grant that user an additional claim that can be used for accessing an SAP system. This is where claims augmentation comes in.

Name resolution comes in two parts: the People Picker control and the type-in control. When you want to enable users to search for claims and find your claims, a claims provider enables you to do this by using the People Picker. The type-in control enables you to type in a user name, or a claim, and then click the resolve button. Your claims provider can examine that input and determine whether it is valid. If it is valid, the claims provider will "do the best thing" to resolve it.

In this scenario rights are assigned in a site collection based on an individual's favorite basketball team. The identity of users or how they authenticated are not important. Instead, only the attribute, which is that individual's favorite basketball team, for the specified user is compelling in this scenario.

The Web application in this scenario assumes both Windows claims and forms-based authentication. Using the standard SQL membership and role provider for the forms-based authentication users, the list of forms-based authentication accounts with **user1** through **user50** have been prepopulated. To simplify the process, an individual's favorite team is determined in the following way:

- The favorite team of all Windows users is **Consolidated Messenger.**
- For forms-based authentication users:
  - The favorite team of **user1** through **user15** is **Consolidated Messenger**.
  - The favorite team of **user16** through **user30** is **Wingtip Toys**.
  - The favorite team of **user31** through **user50** is **Tailspin Toys**.

Augmenting Claims and Registering Your Provider

In this scenario, the previously defined rules (a user's favorite basketball team) is added to the user token that is used in claims augmentation.

The following steps provide sample code associated with authoring a claims provider by using Visual Studio 2010 to create a class library application.

Add References

Create a new class library and add references to Microsoft.SharePoint and **Microsoft.IdentityModel**. The Microsoft.IdentityModel.dll is installed as part of the Windows Identity Foundation. The following is the full path of Microsoft.IdentityModel.dll.

```
<drive>:\Program Files\Reference Assemblies\Microsoft\Windows Identity
Foundation\v3.5\Microsoft.IdentityModel.dll
```

Add using Statements and Class Inheritance

To add the necessary **using** statements and define the base class that the class inherits from, see the following code sample.

C# Code

```
using Microsoft.SharePoint.Diagnostics;
using Microsoft.SharePoint;
using Microsoft.SharePoint.Administration;
using Microsoft.SharePoint.Administration.Claims;
using Microsoft.SharePoint.WebControls;
```

The class must inherit from the SPClaimProvider base class. The following code illustrates how the class appears at the start:

NOTE

The name, SQLClaimsProvider, in the following code sample is used for illustration purposes. The code sample also implies Windows claims.

C# Code

```
namespace SqlClaimsProvider
{
    public class SqlClaims : SPClaimProvider
    {
        public SqlClaims(string displayName)
            : base(displayName)
        {
        }
    }
}
```

Adding the Required Implementation

As opposed to stepping through all the interfaces to be implemented, pause over the SPClaimProvider name in the previous code, click the drop-down arrow that appears under **S**, and then select the **Implement abstract class 'SPClaimProvider'** command.

Implementing the Functionality Included with the Provider

Following the implementation of the abstract class, five properties in the **SPClaimProvider** class must be implemented: **Name**, **SupportsEntityInformation**, **SupportsHierarchy**, **SupportsResolve**, and **SupportsSearch**.

Start with the **Supports*** properties. If claims augmentation is the extent of the effort, only the entity information must be supported. The property list for this class resembles the following.

C# Code

```
public override bool SupportsEntityInformation
{
      get
      {
            return true;
      }
}
public override bool SupportsHierarchy
{
      get
      {
            return false;
      }
```

```
}
public override bool SupportsResolve
{
      get
      {
            return false;
      }
}
public override bool SupportsSearch
{
      get
      {
            return false;
      }
}
```

Next, implement support for the Name property. It is recommended to support both a display name and an "internal" name by which your provider can be referenced.

In this case, internal static properties are used so they can be shared with another class.

C# Code

```
internal static string ProviderDisplayName
{
      get
      {
            return "Basketball Teams";
      }
}
internal static string ProviderInternalName
{
      get
      {
            return "BasketballTeamProvider";
      }
}
public override string Name
{
      get
      {
            return ProviderInternalName;
      }
}
```

The previous steps provide the basic implementation.

An array is set up to be used in the claims provider for the teams. It is added at the top of the class, as follows.

C# Code

```
// Teams that we are using.
```

```
private string[] ourTeams = new string[] { "Consolidated Messenger", "Wingtip Toys",
"Tailspin Toys " };
```

Next implement the core of the claims provider. To perform claims augmentation, add implementation code for the **FillClaimsForEntity** method, the **FillClaimTypes** method, and the **FillClaimValueTypes** method in the **SPClaimProvider** class.

Create simple helper functions because they are called throughout the creation of the claims provider. These helper functions define the **ClaimType**, which is the claim namespace or identifier, and the **ClaimValueType**, such as **string** or **int**.

The following code samples define the two helper functions.

C# Code

```
private static string SqlClaimType
{
      get
      {
            return "http://schema.contoso.local/teams";
      }
}
private static string SqlClaimValueType
{
      get
      {
            return Microsoft.IdentityModel.Claims.ClaimValueTypes.String;
      }
}
```

After completing the previous steps, a claim is added with the name, http://schema.contoso.local/teams, and the value in that claim is a string.

The previous sample code implies the value will be **Consolidated Messenger**, **Wingtip Toys**, or **Tailspin Toys**. For the claim name itself, there are no required rules for what the name must appear as. The general recommendation is to follow the format, schemas.*SomeCompanyName.com*/*ClaimName*. In this scenario, the domain is **contoso.local**. Therefore, it is represented as **schemas.contoso.local**, and **teams** is the property tracked by this claim.

The following sample code describes how to add the claim in the **FillClaimsForEntity** method of the **SPClaimProvider** class.

C# Code

```
protected override void FillClaimsForEntity(Uri context, SPClaim entity,
      List<SPClaim> claims)
{
if (entity == null)
      throw new ArgumentNullException("entity");
```

```
if (claims == null)
      throw new ArgumentNullException("claims");
// Determine who the user is, so that we know what team to add to their claim
// entity. The value from the input parameter contains the name of the
// authenticated user. For a SQL forms-based authentication user, it looks similar to
// 0#.f|sqlmembership|user1; for a Windows claims user it looks similar
// to 0#.w|contoso\\wilmaf.
// I will skip some uninteresting code here, to look at that name and determine
// whether it is a forms-based authentication user or a Windows user, and if it is a forms-
based authentication user,
// determine what the number part of the name is that follows "user".
string team = string.Empty;
int userID = 0;
// After the uninteresting code, "userID" will equal -1 if it is a Windows user.
// If it is a forms-based authentication user, it will contain the number that follows
"user".
// Determine what the user's favorite team is.
if (userID > 0)
{
      // Plug in the appropriate team.
      if (userID > 30)
            team = ourTeams[2];
      else if (userID > 15)
            team = ourTeams[1];
      else
            team = ourTeams[0];
}
else
      team = ourTeams[1];
      // If they are not one of our forms-based authentication users,
      // make their favorite team Wingtip Toys.

// Add the claim.
claims.Add(CreateClaim(SqlClaimType, team, SqlClaimValueType));
}
```

In the final line of code in the previous sample code, the input claims parameter is taken as a list of SPClaim objects and a new claim is created using the **CreateClaim** helper method.

Tip:

Use the helper methods when possible. They generally do more than merely using the default constructors, and you will generally find that things work more completely when you use them.

Two helper methods, **SqlClaimType** method and the **SqlClaimValueType** method, are used when creating the claims as described previously.

In order to complete the implementation, two additional methods are required for: **FillClaimTypes** and **FillClaimValueTypes**. These methods are simplified because the helper methods will be used for them. The following sample code resembles their implementation.

C# Code

```
protected override void FillClaimTypes(List<string> claimTypes)
{
      if (claimTypes == null)
            throw new ArgumentNullException("claimTypes");

      // Add our claim type.
      claimTypes.Add(SqlClaimType);
}
protected override void FillClaimValueTypes(List<string> claimValueTypes)
{
      if (claimValueTypes == null)
            throw new ArgumentNullException("claimValueTypes");

      // Add our claim value type.
      claimValueTypes.Add(SqlClaimValueType);
}
```

The basic implementation of a claims provider that provides augmentation is most commonly implemented through a claims Feature Receiver.

Creating the Claims Provider Feature Receiver

To create a claims provider Feature Receiver, start by adding a class to the project. This class will inherit from the SPClaimProviderFeatureReceiver base class.

C# Code

```
using Microsoft.SharePoint;
using Microsoft.SharePoint.Administration;
using Microsoft.SharePoint.Administration.Claims;
using Microsoft.SharePoint.Diagnostics;
namespace SqlClaimsProvider
{
    public class SqlClaimsReceiver : SPClaimProviderFeatureReceiver
    {
        private void
              ExecBaseFeatureActivated(
              Microsoft.SharePoint.SPFeatureReceiverProperties properties)
        {
            // Wrapper function for base FeatureActivated. Used because base
            // keyword can lead to unverifiable code inside lambda expression.
            base.FeatureActivated(properties);
        }
        public override string ClaimProviderAssembly
        {
            get
            {
                return typeof(SqlClaims).Assembly.FullName;
            }
        }
        public override string ClaimProviderDescription
        {
            get
            {
                return "A sample provider written by Dan Jump";
```

```
            }
        }
        public override string ClaimProviderDisplayName
        {
            get
            {
                // This is where we reuse that internal static property.
                return SqlClaims.ProviderDisplayName;
            }
        }
        public override string ClaimProviderType
        {
            get
            {
                return typeof(SqlClaims).FullName;
            }
        }
        public override void FeatureActivated(
              SPFeatureReceiverProperties properties)
        {
            ExecBaseFeatureActivated(properties);
        }
    }
}
```

NOTE

- The **ClaimProviderAssembly** property and the **ClaimProviderType** property of the SPClaimProviderFeatureReceiver class are merely using type properties of the custom claims provider class that we wrote.
- **ClaimProviderDisplayName** uses the internal static property that we created on our claims provider class. We created the property so that we could reuse it in our claims feature receiver.
- The **FeatureActivated** event calls our special internal method for activating properties.

Compiling the Assembly and Adding It to the Global Assembly Cache

The following example implies a manual process; however, solution packaging and deployment can be used for greater efficiency in deployment and management.

The assembly must be strongly named and compiled to complete these steps.

Creating and Deploying the Claims Provider Feature

In this section a standard Feature is created for deployment to the SharePoint farm. The following XML snippet is included for reference.

Note:

For more information about how to install and uninstall features, see Installing or Uninstalling Features (http://go.microsoft.com/fwlink/?LinkId=228924).

XML Snippet

```
<Feature   Id="3E864B5C-D855-43e4-B41A-6E054C6C0352"
           Title="Sql Claims Provider Feature"
           Scope="Farm"
           ReceiverAssembly="SqlClaimsProvider, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=eaf1ba21464322ae"
           ReceiverClass="SqlClaimsProvider.SqlClaimsReceiver"
           xmlns="http://schemas.microsoft.com/sharepoint/" />
```

Using the referenced XML snippet, a directory is created as SqlClaimsProvider in the Features folder where the Feature is installed. As a result of being defined as a farm-scoped Feature, it is automatically activated upon deployment.

To deploy the Feature by using Windows® PowerShell®.

Windows PowerShell Command

```
install-spfeature -path SqlClaimsProvider
```

This completes the implementation.

Logging In to the Site and Trying It Out

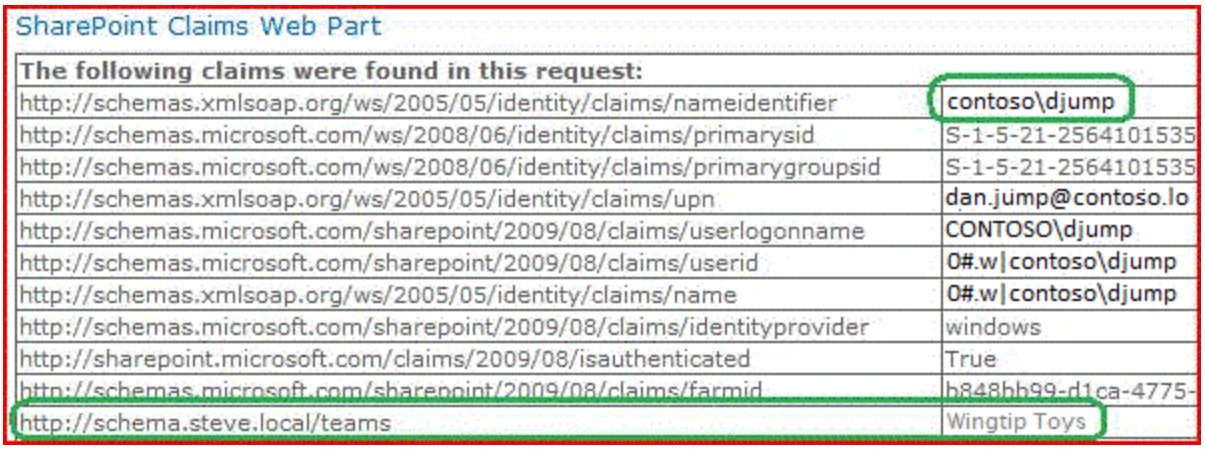
To validate whether claims are being appropriately augmented, this scenario includes a Web Part that displays all claims.

Note:

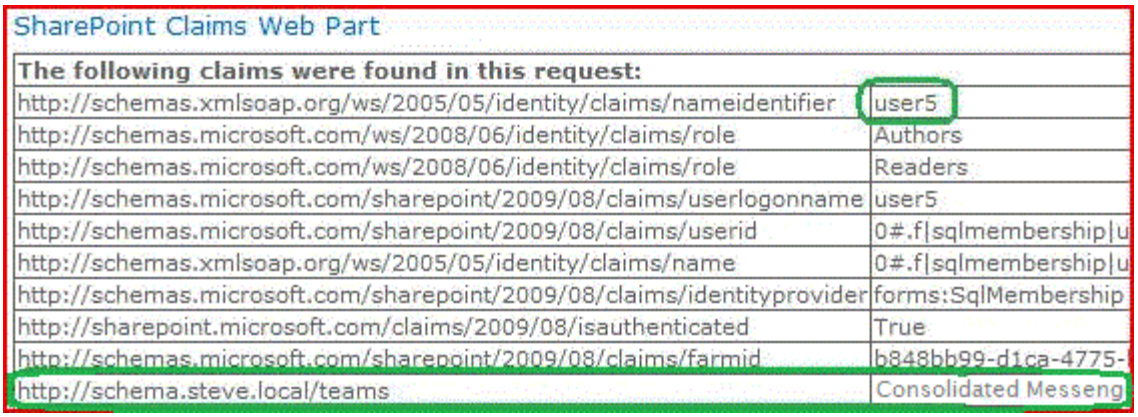
For more information about how to sign on to SharePoint Server, see Incoming Claims: Signing into SharePoint (http://go.microsoft.com/fwlink/?LinkId=228927).

In this scenario a user logs on to the site by using Windows credentials keeping in mind from the code earlier that a Windows user should always be assigned **Wingtip Toys** as his or her favorite team.

The following illustration depicts how this appears for a Windows user.



The following illustration depicts what it resembles for a forms-based authenticated user.



The following section describes how to work with the People Picker and also describes supporting and adding a hierarchy to the People Picker.

Adding Support for Hierarchy Nodes

This section describes a simple method to add a hierarchy to the People Picker control illustrated further in this document.

The sample code in the previous section modifies only one property and one method of the **SPClaimProvider** class: the **SupportsHierarchy** property and the **FillHierarchy** method.

To add support for hierarchy nodes, the **SupportsHierarchy** property should be set to return **true** to instruct SharePoint Server to fill a hierarchy, as shown in the code sample that follows.

C# Code

```
public override bool SupportsHierarchy
{
      get
      {
```

```
            return true;
      }
}
```

After you set the **SupportsHierarchy** property to **true**, the **FillHierarchy** method will be called by SharePoint Server when the People Picker is displayed.

After the node is added to the hierarchy, a display name and node identifier (ID) is required. Simple helper arrays to track label names and IDs are created as shown in the code sample that follows.

C# Code

```
// Teams that we are using.
private string[] ourTeams = new string[] { "Consolidated Messenger", "Wingtip Toys",
"Tailspin Toys " };
// Keys for our People Picker hierarchy.
private string[] teamKeys = new string[] { "empUS", "empEMEA", "empASIA" };
// Labels for our nodes in the People Picker.
private string[] teamLabels = new string[] { "US Teams", "European Teams", "Asian Teams" };
```

The **FillHierarchy** method is used to add the nodes when the People Picker is displayed.

A claim can be used to provision a number of security items in a SharePoint site with the exception of a site collection administrator. When a search is executed for a site collection administrator, the *entityTypes* array has one item, and it is Users. In other cases, at least six items are presented in the *entityTypes* array which provides indication of the search performed. The following sample code adds a special check when the hierarchy is filled to prevent the addition of an unused hierarchy.

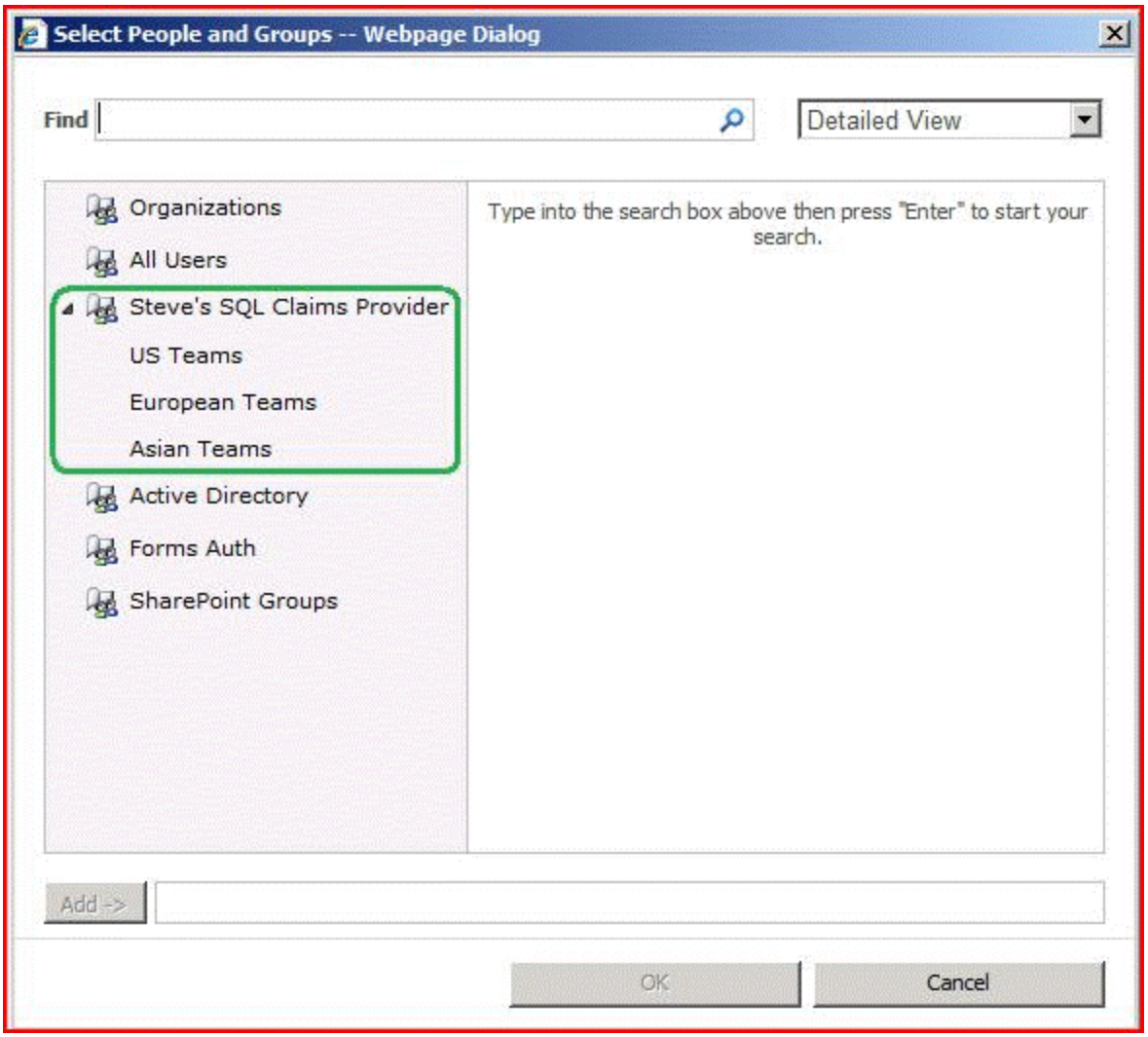
C# Code

```
protected override void FillHierarchy(Uri context, string[] entityTypes,
      string hierarchyNodeID, int numberOfLevels,
      Microsoft.SharePoint.WebControls.SPProviderHierarchyTree hierarchy)
{
// Ensure that People Picker is asking for the type of entity that we
// return; site collection administrator will not return, for example.
if (!EntityTypesContain(entityTypes, SPClaimEntityTypes.FormsRole))
      return;
// Check to see whether the hierarchyNodeID is null; it is when the control
// is first loaded, but if a user clicks on one of the nodes, it will return
// the key of the node that was clicked. This lets you build out a
// hierarchy as a user clicks something, instead of all at once. So I
// wrote the code for that scenario, but I am not using it in that way.
// Which means that I could have just as easily used an
// if (hierarchyNodeID == null) in this particular implementation, instead
// of a switch statement.
switch (hierarchyNodeID)
{
      case null:
            // When it first loads, add all our nodes.
```

```
            hierarchy.AddChild(new
                  Microsoft.SharePoint.WebControls.SPProviderHierarchyNode(
                  SqlClaims.ProviderInternalName,
                  teamLabels[0],
                  teamKeys[0],
                  true));
            hierarchy.AddChild(new
                  Microsoft.SharePoint.WebControls.SPProviderHierarchyNode(
                  SqlClaims.ProviderInternalName,
                  teamLabels[1],
                  teamKeys[1],
                  true));
            hierarchy.AddChild(new
                  Microsoft.SharePoint.WebControls.SPProviderHierarchyNode(
                  SqlClaims.ProviderInternalName,
                  teamLabels[2],
                  teamKeys[2],
                  true));
            break;
      default:
            break;
}
}
```

In the code samples, the only check that is performed is whether the *hierarchyNodeID* is null, which it always is the first time that the control loads. If the *hierarchyNodeID* is null, node trees are added in the left pane.

The following illustration depicts the People Picker following implementation of this code.



Searching Claims

In this section the following concepts are described in detail:

- Creating a custom claims provider
- Performing claims augmentation
- Adding a hierarchy to the People Picker

This section describes how to implement searching for claims in the People Picker with the custom claims provider. Provisioning permissions in a site that is based on an individual's favorite basketball team is not possible unless a custom claim can be selected.

Provisioning permissions based on an attribute of a user, instead of on whom a user is, opens a number of possibilities. For example, a user's method of authentication or whether that user is a

Windows user or a forms-based authentication user is not important. This scenario is concerned with the favorite basketball team associated with that user.

As an example, the following scenario is applicable to Office SharePoint Server 2007 and SharePoint Server 2010—assuming users are on their corporate network and authentication is supplied using Windows credentials. However, outside the corporate network, users access the same content over the extranet by using forms-based authentication. From a SharePoint Server perspective, a user is considered as two separate entities because no user mapping is provided. The user is this scenario is treated as separate entities which represent each authentication method.

Claims-based authentication removes this inconsistency because permissions are assigned based on a user's favorite basketball team. Imagine a piece of corporate metadata that is associated with a person, and the claims provider doing a lookup to some other system to determine all the different identities for a person—Windows authentication, forms-based authentication, SAP, CRM, and so on—and being able to map some other identifier or set of claims to that identity. Those claims are then used to grant access to resources.

Assume a scenario in which single sign-on is used between separate Web applications: **user1** in Web application 1 differs from **user1** in Web application 2. If the user's favorite basketball team is used as the claim, the user can move seamlessly between the Web applications because the claims are augmented each time the user authenticates.

For the custom claims provider to support search, support must be added for the following properties and methods of the **SPClaimProvider** class: **SupportsSearch**, **SupportsResolve**, **FillSearch**, **FillResolve** (the overload with **SPClaim** as an input parameter), **FillSchema**, and **FillEntityTypes** as illustrated in the following code sample.

C# Code

```
public override bool SupportsSearch
{
      get
      {
            return true;
      }
}
public override bool SupportsResolve
{
      get
      {
            return true;
      }
}
```

The **FillSearch** method:

C# Code

```
protected override void FillSearch(Uri context, string[] entityTypes,
      string searchPattern, string hierarchyNodeID, int maxCount,
      Microsoft.SharePoint.WebControls.SPProviderHierarchyTree searchTree)
{
// Ensure that People Picker is asking for the type of entity that we
// return; site collection administrator will not return, for example.
if (!EntityTypesContain(entityTypes, SPClaimEntityTypes.FormsRole))
      return;
// The counter to track what node we are in; it will be used to call into
// our helper arrays that were covered in part 1 and part 2 of this article.
int teamNode = -1;
// NOTE: If we were not using hierarchies in the People Picker, we would
// probably use a list of PickerEntity instances so that we could
// add them all to the People Picker in one call. I have stubbed it out
// here for demonstration purposes so you can see how you
// would do it.
// Picker entities that we will add if we find something.
// List<PickerEntity> matches = new List<PickerEntity>();
// The node where we will place our matches.
Microsoft.SharePoint.WebControls.SPProviderHierarchyNode matchNode = null;
// Look to see whether the value that is typed in matches any of our teams.
foreach (string team in ourTeams)
{
      // Increment team node tracker.
      teamNode += 1;

      // Simple way to do a string comparison to the search criteria.
      // This way, all a person has to type in to find Consolidated Messenger is "b".
      if (team.ToLower().StartsWith(searchPattern.ToLower()))
      {
            // We have a match, create a matching entity.
            // This is a helper method that I will explain later.
            PickerEntity pe = GetPickerEntity(team);
            // If we did not have a hierarchy, we would add it here
            // by using the list described previously.
            // matches.Add(pe);
            // Add the team node where it should be displayed;
            // ensure that we have not already added a node to the tree
            // for this team's location.
            if (!searchTree.HasChild(teamKeys[teamNode]))
            {
                  // Create the node so that we can show our match in there too.
                  matchNode = new
                  SPProviderHierarchyNode(SqlClaims.ProviderInternalName,
                  teamLabels[teamNode],
                  teamKeys[teamNode],
                  true);
                  // Add it to the tree.
                  searchTree.AddChild(matchNode);
             }
             else
                  // Get the node for this team.
                  matchNode = searchTree.Children.Where(theNode =>
                  theNode.HierarchyNodeID == teamKeys[teamNode]).First();

            // Add the picker entity to our tree node.
            matchNode.AddEntity(pe);
      }
```

```
}
// If we were adding all the matches at once, that is, if we were not using
// a hierarchy, then we would add the list of matches here.
// if (matches.Count > 0)
//    searchTree.AddEntities(matches);
}
```

In this example, the user entered text in the search box in People Picker and clicked the search button. The claims provider is called because the **SupportsSearch** property returns **true**. The method that is called is **FillSearch**. In that method, the submitted text, which is provided by the *searchPattern* input parameter, is examined. Each team name is examined to determine whether any of them start with the value that was entered in the search box. If a match is located, a new **PickerEntity** is created to either find or create a search tree hierarchy node with the location of that team, and the PickerEntity entity is added to that node.

The code illustrates how to use a special helper function named **GetPickerEntity** where the name of the team that is passed is found as described further in the following code.

C# Code

```
private PickerEntity GetPickerEntity(string ClaimValue)
{
// Use the helper function!
PickerEntity pe = CreatePickerEntity();
// Set the claim that is associated with this match.
pe.Claim = CreateClaim(SqlClaimType, ClaimValue, SqlClaimValueType);
// Set the tooltip that is displayed when you pause over the resolved claim.
pe.Description = SqlClaims.ProviderDisplayName + ":" + ClaimValue;
// Set the text that we will display.
pe.DisplayText = ClaimValue;
// Store it here, in the hashtable **
pe.EntityData[PeopleEditorEntityDataKeys.DisplayName] = ClaimValue;
// We plug this in as a role type entity.
pe.EntityType = SPClaimEntityTypes.FormsRole;
// Flag the entry as being resolved.
pe.IsResolved = true;
// This is the first part of the description that shows
// above the matches, like Role: Forms Auth when
// you do an forms-based authentication search and find a matching role.
pe.EntityGroupName = "Favorite Team";
return pe;
}
```

As a result, a new **PickerEntity** object is created. As the code comments indicate, the **CreatePickerEntity** method of the **SPClaimProvider** class is another helper function that SharePoint Server provides.

After creating the **PickerEntity** object, it must be described. In this case, "it" is a claim for a favorite basketball team. The way it is described to the system is to create a claim by using the same approach that as illustrated in the claims augmentation code in this document.

The team name is passed to the helper function to support this scenario and place the user's favorite basketball team into the favorite basketball team claim. Several properties are set on the **PickerEntity** object so that it displays and behaves as other claims that are presented for Windows claims or forms-based authentication claims, and the **PickerEntity** that was created is returned.

To support selection of the displayed claim in the People Picker, an additional overload is required, the **FillResolve** method. This is an overloaded method. The method that is called into when closing the **People Picker** dialog box is the method that contains an **SPClaim** input parameter. This implementation is provided in the code sample that follows.

C# Code

```
protected override void FillResolve(Uri context, string[] entityTypes,
      SPClaim resolveInput,
      List<Microsoft.SharePoint.WebControls.PickerEntity> resolved)
{
// Ensure that People Picker is asking for the type of entity that we
// return; site collection administrator will not return, for example.
if (!EntityTypesContain(entityTypes, SPClaimEntityTypes.FormsRole))
      return;
// Same sort of code as in search, to validate that we have a match.
foreach (string team in ourTeams)
{
if (team.ToLower() == resolveInput.Value.ToLower())
{
      // We have a match; create a matching entity with helper method.
      PickerEntity pe = GetPickerEntity(team);

      // Add it to the return list of picker entries.
      resolved.Add(pe);
}
}
```

The **FillResolve** method is basic. The **Value** property of the **SPClaim** class contains the value for the claim. Because multiple items may have been selected in the People Picker as passed in a claim, the code looks through the team list again to see whether the claim value matches one of the teams. If a match is located, the helper method is called to create a **PickerEntity** object, and then added to the list of **PickerEntity** instances for SharePoint Server. As a result, a resolved entry in the type-in control is presented following selection in the People Picker.

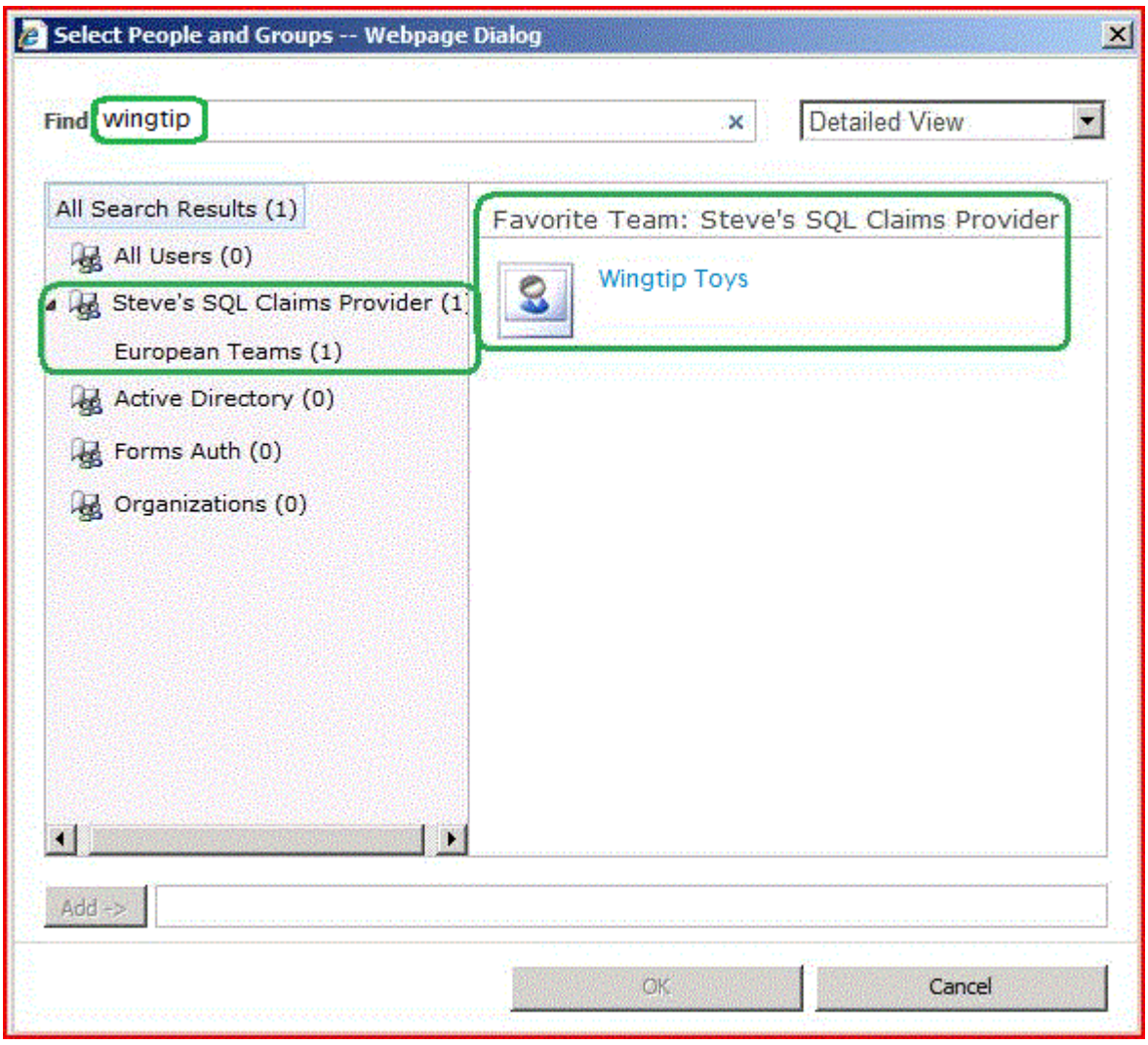
The following methods are required to be implemented: the **FillSchema** method and the **FillEntityTypes** method of the **SPClaimProvider** base class. In the **FillSchema** method, the minimal property is added to be used in the People Picker, which is the display name.

To return the claim type or claims that the provider uses (**SPClaimEntityTypes.FormsRole** in the **GetPickerEntity** method), the role must be added to the list of entity types as shown in the code sample that follows.

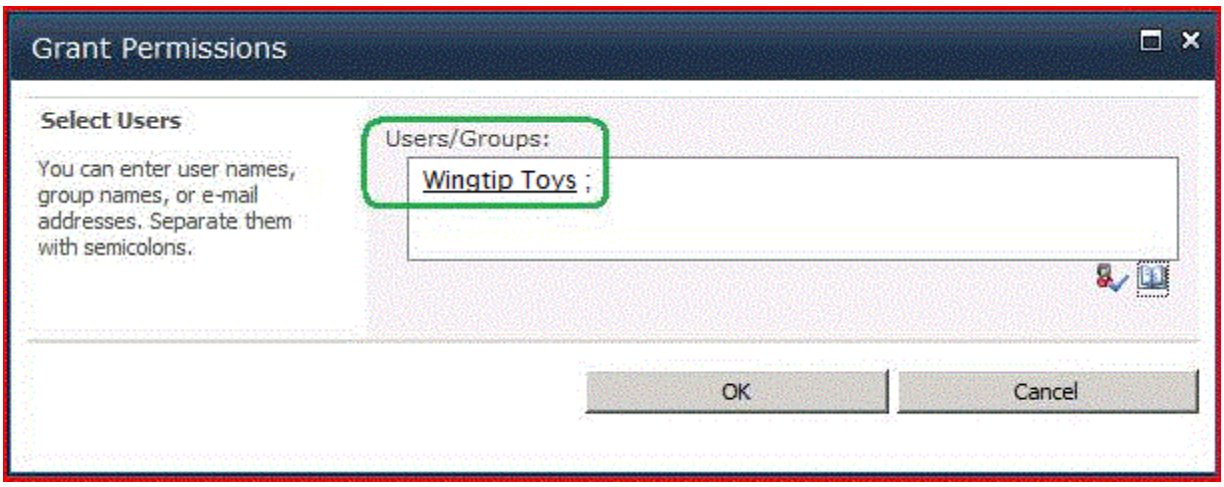
C# Code

```
protected override void
FillSchema(Microsoft.SharePoint.WebControls.SPProviderSchema schema)
{
// Add the schema element that we need at a minimum in our picker node.
schema.AddSchemaElement(new
      SPSchemaElement(PeopleEditorEntityDataKeys.DisplayName,
      "Display Name", SPSchemaElementType.Both));
}
protected override void FillEntityTypes(List<string> entityTypes)
{
// Return the type of entity claim that we are using (only one in this case).
entityTypes.Add(SPClaimEntityTypes.FormsRole);
}
```

The following illustration depicts how the People Picker appears after searching for **wingtip**.



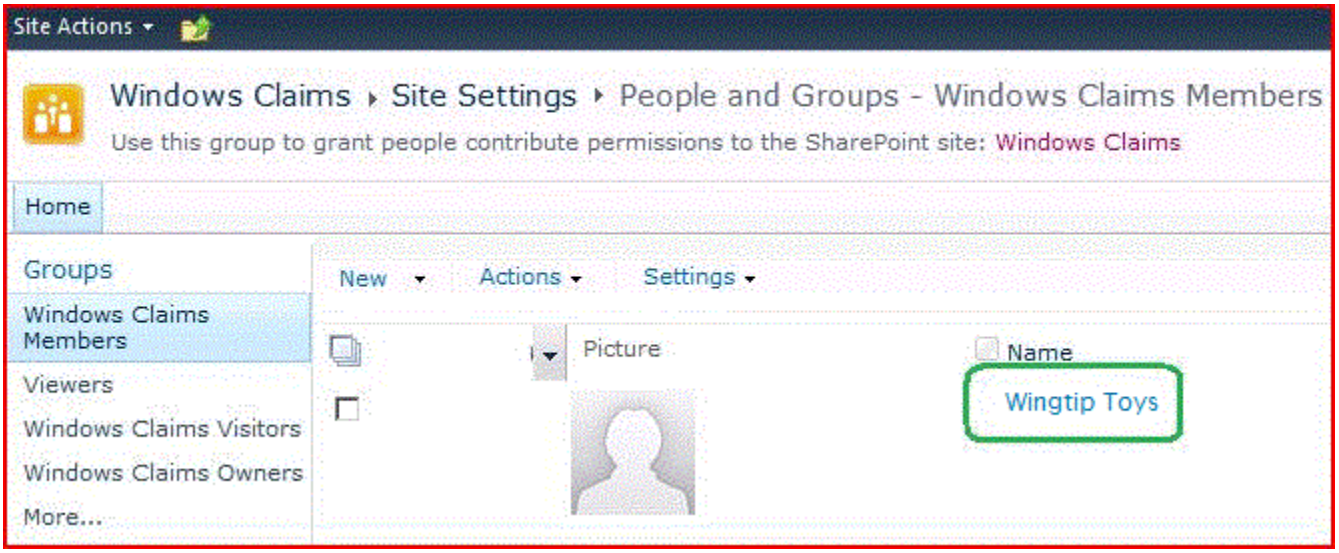
The following illustration depicts how the type-in control appears after adding the claim and then clicking **OK**.



The sample code provided here makes it possible to provision a user based on a favorite basketball team. By selecting Wingtip Toys, it is possible to allow all users whose favorite basketball team is Wingtip Toys to contribute content to the site.

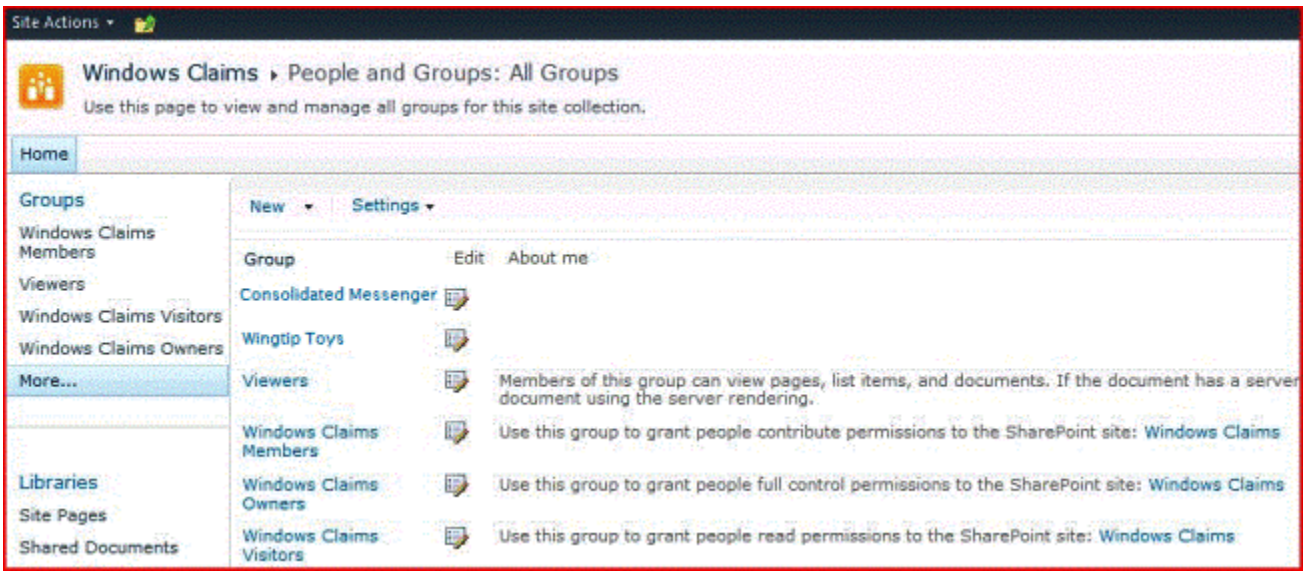
In this example the claims are added to the **Members** group.

The following illustration depicts how that appears (**Windows Claims** is the name of the site collection).



In this example, a claim for Wingtip Toys is added to the Members group and a claim for Consolidated Messenger is added to the Visitors group. No Windows user, group, forms-based authentication, or forms-based authentication role rights where added to the site.

The following illustration lists groups for the site collection.



Supporting Name Resolution

This section defines how name resolution support can be added in the type-in control.

To add name resolution support, implement the following property and method of the **SPClaimProvider** class: the **SupportsResolve** property and the **FillResolve** method.

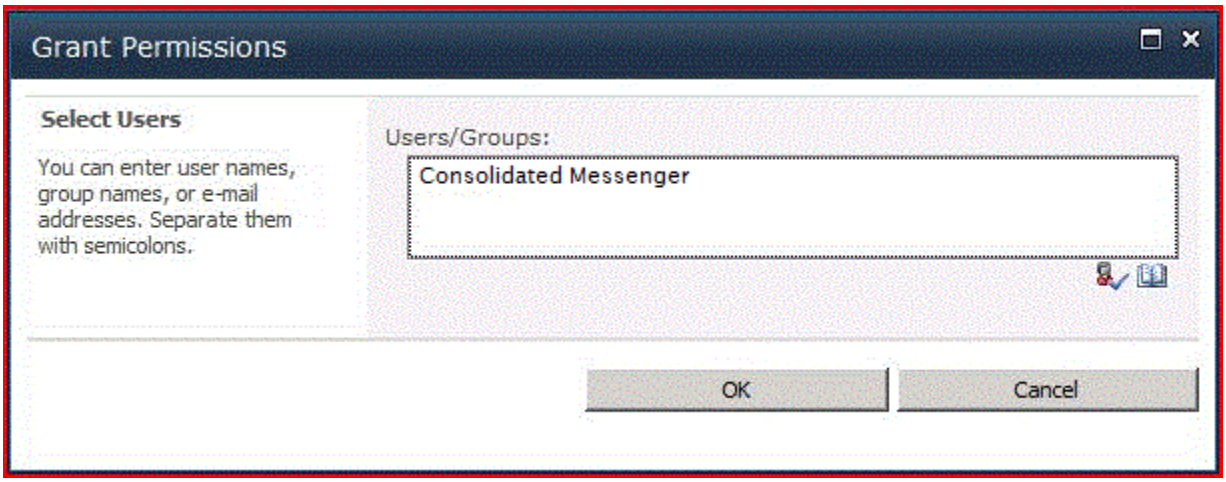
The following code implements the other overload of the **FillResolve** method that was not described in Searching Claims in this document.

C# Code

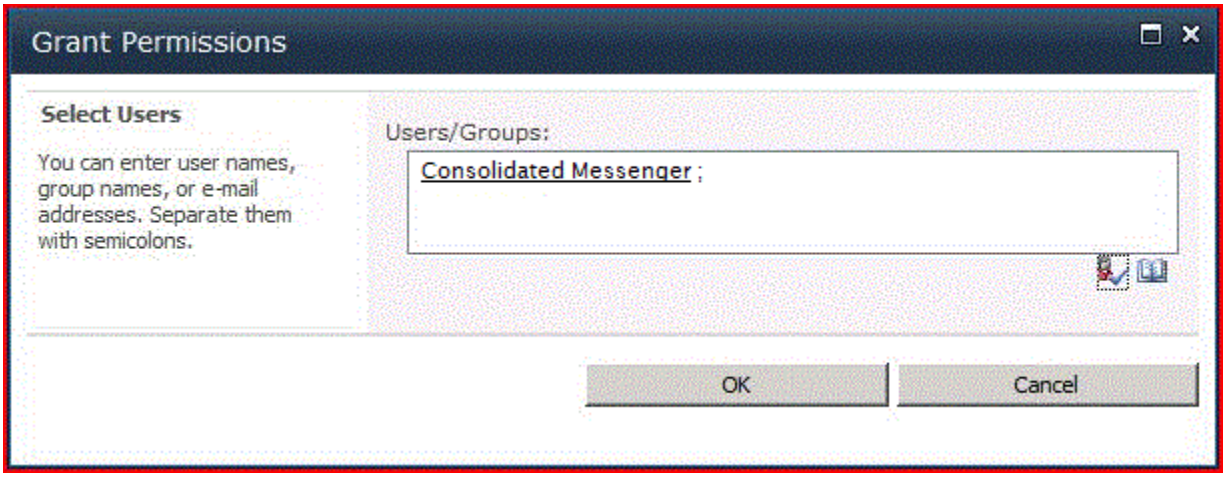
```
protected override void FillResolve(Uri context, string[] entityTypes,
      string resolveInput,
      List<Microsoft.SharePoint.WebControls.PickerEntity> resolved)
{
// Ensure that People Picker is asking for the type of entity that we
// return; site collection administrator will not return, for example.
if (!EntityTypesContain(entityTypes, SPClaimEntityTypes.FormsRole))
      return;
// Same sort of code as in search, to validate that we have a match.
foreach (string team in ourTeams)
{
      if (team.ToLower() == resolveInput.ToLower())
      {
            // We have a match;, create a matching entity.
            PickerEntity pe = GetPickerEntity(team);
            // Add it to the return list of picker entries.
            resolved.Add(pe);
       }
}
}
```

The following illustrations depict the type-in control before and after name resolution.

Type-in control before clicking resolve



Type-in control after resolve



Conclusion

You can use a claims provider in SharePoint Server 2010 to augment claims and provide name resolution. By using claims authentication, you can assign rights based on claims without having to know who users are, or how they are authenticated. You only have to know the attributes of the users. You can, for example, use a piece of corporate metadata that is associated with a person and have the claims provider do a lookup to another system to determine the different identities of a particular person—Windows, forms-based authentication, SAP, CRM, and so on—and map another identifier or set of claims to that identity. Those claims are then used to grant access to resources.

Migration of Users in Classic Mode to Trusted Provider Claims

Introduction

This section describes the method to migrate Windows (domain) users to Trusted Provider (user identity) claims.

SharePoint Web applications (which are configured to use Windows Authentication) might be migrated from Office SharePoint Server 2007 to a SharePoint Server 2010 environment where these Web Applications will be configured to use claims-based authentication (for SAML claims – using Trusted Providers). The migration process has to make sure that the permissions set on SharePoint securable objects (sites, lists, and so on,) and the domain users are migrated to trusted provider user identity claims.

The MigrateUsers method of the SPWebApplication class migrates users from classic Windows (domain) to either classic Windows (domain) or Windows claims. It does not migrate users from classic Windows to Trusted Provider user identity claims. To address this requirement, the Office 2010 Cumulative Update for August 2011 modifies the MigrateUsers method to consume a custom MigrateUser Callback method, which can be custom developed by customers based on their requirements.

If users are not migrated to Trusted Provider user identity claims, the content which was created in SharePoint Server 2007 by using Windows domain users will be orphaned.

Assumptions

- The necessary Web Applications have been created and configured to use claims-based authentication with a Trusted Provider.
- Content has been migrated from SharePoint Server 2007 to SharePoint Server 2010.
- The SharePoint Server 2010 environment has been updated with Office 2010 Cumulative Update for August 2011.

Claims Encoding

The following describes claims encoding. Keep in mind that the way a claim is created for each token issuer can be different based on the primary identity claim that was used. It is generally recommended to review the user profiles after adding them before activating the claim provider feature.

Example: i:0#.w|contoso\wbaer

Definitions:

i = Identity Claim all other claims will use “c” as opposed to “i”

: = Colon

0 = Reserved to support future Claims

#/? = Claim Type Encoded Value. The default claim types will have a hardcoded encoded value that will enable parity across farms.

E.g. Key: ? Value: http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier

Key: # Value: http://schemas.microsoft.com/sharepoint/2009/08/claims/userlogonname

./0 = Claim Value Type. The default claim value types will have a hardcoded encoded value that will enable parity across farms.

E.g. Key: . Value: urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name

Key: 0 Value: http://www.w3.org/2001/XMLSchema#string

w/m/r/t/p/s = Original Issuer Type -> w = windows, m = membership, r = role, t = trusted STS, p = personal card, s= local sts claim

How to implement the IMigrateUserCallback interface

This section describes how to implement the ConvertFromOldUser method which is part of IMigrateUserCallback interface.

The IMigrateUserCallback is part of the Microsoft.SharePoint.Administration namespace in Microsoft.SharePoint.dll. This interface has only one method definition which is ConvertFromOldUser.

ConvertFromOldUser method

This method takes the following parameters:

- Old User value (In the case of Windows Domain User, this value would be the SID of the domain account.)
- Authentication Type (If the content was migrated from Classic Windows authentication-enabled SharePoint Web Applications, this would be “Windows”.)
- Flag to indicate whether this Old User is a Group or not

Sample code for implementing ConvertFromOldUser method

The sample code below is a class library which shows the implementation of the ConvertFromOldUser method which is part of IMigrateUserCallback interface.

C# Code

```
using System;
using System.Security;
using System.Security.Principal;
using Microsoft.IdentityModel.Claims;
```

```
using Microsoft.SharePoint;
using Microsoft.SharePoint.Administration;
using Microsoft.SharePoint.Administration.Claims;

namespace RajuSakthivel.SharePoint.Claims.ConvertToTrustedClaims
{
    public class ConvertToTrustedClaims : IMigrateUserCallback
    {
        private string claimProviderName;
        private string userIdentityClaimType;

        public ConvertToTrustedClaims(string ClaimProviderName, string UserIdentityClaimType)
        {
            TrustedClaimProviderName = ClaimProviderName;
            TrustedUserIdentityClaimType = UserIdentityClaimType;
        }

        public string TrustedClaimProviderName
        {
            get
            {
                return claimProviderName;
            }
            set
            {
                claimProviderName = value;
            }
        }

        public string TrustedUserIdentityClaimType
        {
            get
            {
                return userIdentityClaimType;
            }
            set
            {
                userIdentityClaimType = value;
            }
        }

        public string ConvertFromOldUser(string oldUser,
SPWebApplication.AuthenticationMethod authType, bool isGroup)
        {
            string encodedClaim = oldUser;

            // Convert only windows user to claims
            if (authType == SPWebApplication.AuthenticationMethod.Windows)
            {
                // Deals with user, not group
                if (!isGroup)
                {
                    // Get the old user value - the parameter oldUser is in sid format
                    SecurityIdentifier oldUserSid = new SecurityIdentifier(oldUser);
                    NTAccount oldUserAccount =
(NTAccount)oldUserSid.Translate(typeof(NTAccount));

                    // Skip the Local Service account
```

```
                    if (oldUserAccount.ToString().ToUpper() != "NT AUTHORITY\\LOCAL SERVICE")
                    {
                        // Get the user from oldUserAccount which is in domain\user format
                        string claimValue = oldUserAccount.ToString().Split('\\')[1];

                        // Create the user claim
                        SPClaim userClaim =
SPClaimProviderManager.CreateUserClaim(claimValue, SPOriginalIssuerType.TrustedProvider,
TrustedClaimProviderName);

                        encodedClaim = userClaim.ToEncodedString();
                    }
                }
            }

            return encodedClaim;
        }
    }
}
```

The sample code in the previous section does not manage groups or other account filtering. The code illustrates how to migrate a Windows domain user to a Trusted Provider user claim and has been tested to migrate Windows domain users to have Trusted Provider user identity claims.

The user identity claim value in the sample code is for demonstration purposes and a developer has to supply the correct claim value based on the related business requirements.

How to use the custom IMigrateUserCallback

This section describes how to use the custom IMigrateUserCallback which was shown in the previous section.

Sample code which uses the custom IMigrateUserCallback

The following sample code is a console application which consumes custom IMigrateUserCallback while calling the MigrateUsers method on a SharePoint Web application.

C# Code

```
using System;
using Microsoft.SharePoint.Administration;
namespace RajuSakthivel.SharePoint.Claims.ConvertToTrustedClaims
{
    class Program
    {
        static void Main(string[] args)
        {
            string WebApplicationUrl = @"http://mywfe:2010/";
            string ClaimProviderName = "AD FSV2STS";
            string UserIdentityClaimType =
"http://schemas.mydomain.com/2010/identity/claims/myid";
```

```
            try
            {
                // Get the Web Application object
                SPWebApplication webApp = SPWebApplication.Lookup(new
Uri(WebApplicationUrl));

                if (webApp != null)
                {
                    // Instantiate the custom class which implements the IMigrateUserCallback
interface
                    ConvertToTrustedClaims c2TrutedClaims = new
ConvertToTrustedClaims(ClaimProviderName, UserIdentityClaimType);
                    IMigrateUserCallback c2TrustedClaimsCallback = c2TrutedClaims as
IMigrateUserCallback;

                    // Call the MigrateUsers method on Web Application by passing the custom
IMigrateUserCallback object
                    webApp.MigrateUsers(c2TrustedClaimsCallback);

                    Console.WriteLine("Users were migrated successfully.");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine("An error occurred! [" + ex.Message + "]");
            }
        }
    }
}
```

Site Collection - UserInfo table

During the MigrateUser process, the tp_login field (login user name), which is part of the UserInfo table, is modified.

UserInfo table before MigrateUsers

The table below illustrates the user row before the MigrateUsers call was executed and tp_login has Windows domain user value.

	tp_SiteID	tp_ID	tp_DomainGroup	tp_Deleted	tp_IsActive	tp_Login	tp_Title
1	2A81876F-317F-4731-92F6-828C0795C9C7	1	0	0	1	i:0ñ.t|adfsv2sts|svc_sharepoint_farm	mydomain\svc_sharepoint_farm
2	2A81876F-317F-4731-92F6-828C0795C9C7	4	1	0	1	NT AUTHORITY\authenticated users	NT AUTHORITY\Authenticated Users
3	2A81876F-317F-4731-92F6-828C0795C9C7	8	0	0	0	i:0ñ.t|adfsv2sts|local service	SharePoint Farm Service Account
4	2A81876F-317F-4731-92F6-828C0795C9C7	9	0	0	1	i:0ñ.t|adfsv2sts|migrateuser100	User100 Migrate
5	2A81876F-317F-4731-92F6-828C0795C9C7	10	0	0	0	NT AUTHORITY\LOCAL SERVICE	NT AUTHORITY\LOCAL SERVICE
6	2A81876F-317F-4731-92F6-828C0795C9C7	11	0	0	1	i:0ñ.t|adfsv2sts|migrateuser101	User101 Migrate
7	2A81876F-317F-4731-92F6-828C0795C9C7	12	0	0	1	MYDOMAIN\migrateuser102	User102 Migrate
8	2A81876F-317F-4731-92F6-828C0795C9C7	1073741823	0	0	0	SHAREPOINT\system	System Account

UserInfo table after MigrateUsers

The table below illustrates the user row after the MigrateUsers call was executed and tp_login now has a user identity claim for the supplied trusted provider.

	tp_SiteID	tp_ID	tp_DomainGroup	tp_Deleted	tp_IsActive	tp_Login	tp_Title
1	2A81876F-317F-4731-92F6-828C0795C9C7	1	0	0	1	i:0ñ.t|adfsv2sts|svc_sharepoint_farm	mydomain\svc_sharepoint_farm
2	2A81876F-317F-4731-92F6-828C0795C9C7	4	1	0	1	NT AUTHORITY\authenticated users	NT AUTHORITY\Authenticated Users
3	2A81876F-317F-4731-92F6-828C0795C9C7	8	0	0	0	i:0ñ.t|adfsv2sts|local service	SharePoint Farm Service Account
4	2A81876F-317F-4731-92F6-828C0795C9C7	9	0	0	1	i:0ñ.t|adfsv2sts|migrateuser100	User100 Migrate
5	2A81876F-317F-4731-92F6-828C0795C9C7	10	0	0	0	NT AUTHORITY\LOCAL SERVICE	NT AUTHORITY\LOCAL SERVICE
6	2A81876F-317F-4731-92F6-828C0795C9C7	11	0	0	1	i:0ñ.t|adfsv2sts|migrateuser101	User101 Migrate
7	2A81876F-317F-4731-92F6-828C0795C9C7	12	0	0	1	i:0ñ.t|adfsv2sts|migrateuser102	User102 Migrate
8	2A81876F-317F-4731-92F6-828C0795C9C7	1073741823	0	0	0	SHAREPOINT\system	System Account

How to Enable Tracing for SharePoint Server 2010 Claims

The following identifies resources to assist debugging and viewing claims with SharePoint Server 2010.

This section accompanies the MSDN WIF Tracing (http://go.microsoft.com/fwlink/?LinkID=196691) article which describes how to use tracing in Windows Identity Foundation.

You must enable tracing in both the *Secure Token Web Service* Web.Config and the claims-enabled *SharePoint Server 2010 Web Application* Web.Config.

Prerequisites

- Download and install the Service Trace Viewer Tool (SvcTraceViewer.exe). This will be available with the .Net 3.5 or 4.0 SDK.

Setup

1. Enable Windows Identity Foundation (WIF) Tracing in the Secure Token Web Service
   a) Go to *C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\WebServices\Root*. This is the location of the *Web.Config* file for the IIS Web Site that hosts the SharePoint Server 2010 Secure Token Web Service.
   b) Open Web.Config in a text editor, such as Notepad.
   c) Copy the following to the Web.Config:

XML Snippet

```
<system.diagnostics>
  <sources>
    <source name="Microsoft.IdentityModel" switchValue="Verbose">
      <listeners>
        <add name="wif" />
      </listeners>
    </source>
    <source name="System.ServiceModel.MessageLogging" logKnownPii="false"
switchValue="Verbose">
      <listeners>
```

```
        <add name="wcf" />
      </listeners>
    </source>
  </sources>
  <sharedListeners>
    <add name="wcf" type="System.Diagnostics.XmlWriterTraceListener"
initializeData="C:\logs\WCF-WebService.svclog"/>
    <add name="wif" type="System.Diagnostics.XmlWriterTraceListener"
initializeData="C:\logs\WIF-WebService.svclog"/>
  </sharedListeners>
  <trace autoflush="true" />
</system.diagnostics>
```

Tip:

For each <add/> element in the <sharedListeners/> element, make sure that you give each trace file a *.svclog extension. This will allow you to double-click the trace files to open them in the Service Trace Viewer Tool.

d) Save the changes to the Web.Config.

2. Enable Windows Identity Foundation (WIF) Tracing in the Web Application.
   a) Go to *C:\inetpub\wwwroot\wss\VirtualDirectories\Your_Virtual_Directory*. This is the location of the *Web.Config* file for the IIS Web Site that hosts your SharePoint Server 2010 Web Application.
   b) Open Web.Config in a text editor, such as Notepad.
   c) Copy the following to the Web.Config:

XML Snippet

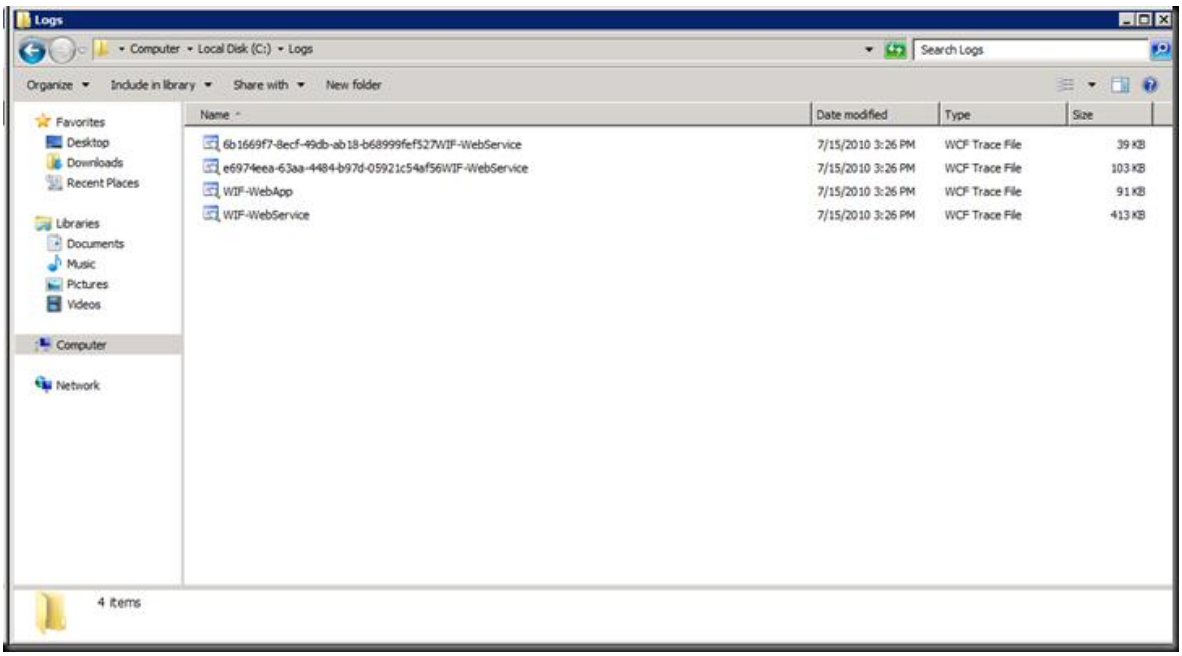
```
<system.diagnostics>
  <sources>
    <source name="Microsoft.IdentityModel" switchValue="Verbose">
      <listeners>
        <add name="wif" />
      </listeners>
    </source>
    <source name="System.ServiceModel.MessageLogging" logKnownPii="false"
switchValue="Verbose">
      <listeners>
        <add name="wcf" />
      </listeners>
    </source>
  </sources>
  <sharedListeners>
    <add name="wcf" type="System.Diagnostics.XmlWriterTraceListener"
initializeData="C:\logs\WCF-WebApp.svclog" />
    <add name="wif" type="System.Diagnostics.XmlWriterTraceListener"
initializeData="C:\logs\WIF-WebApp.svclog" />
  </sharedListeners>
  <trace autoflush="true" />
</system.diagnostics>
```

NOTE

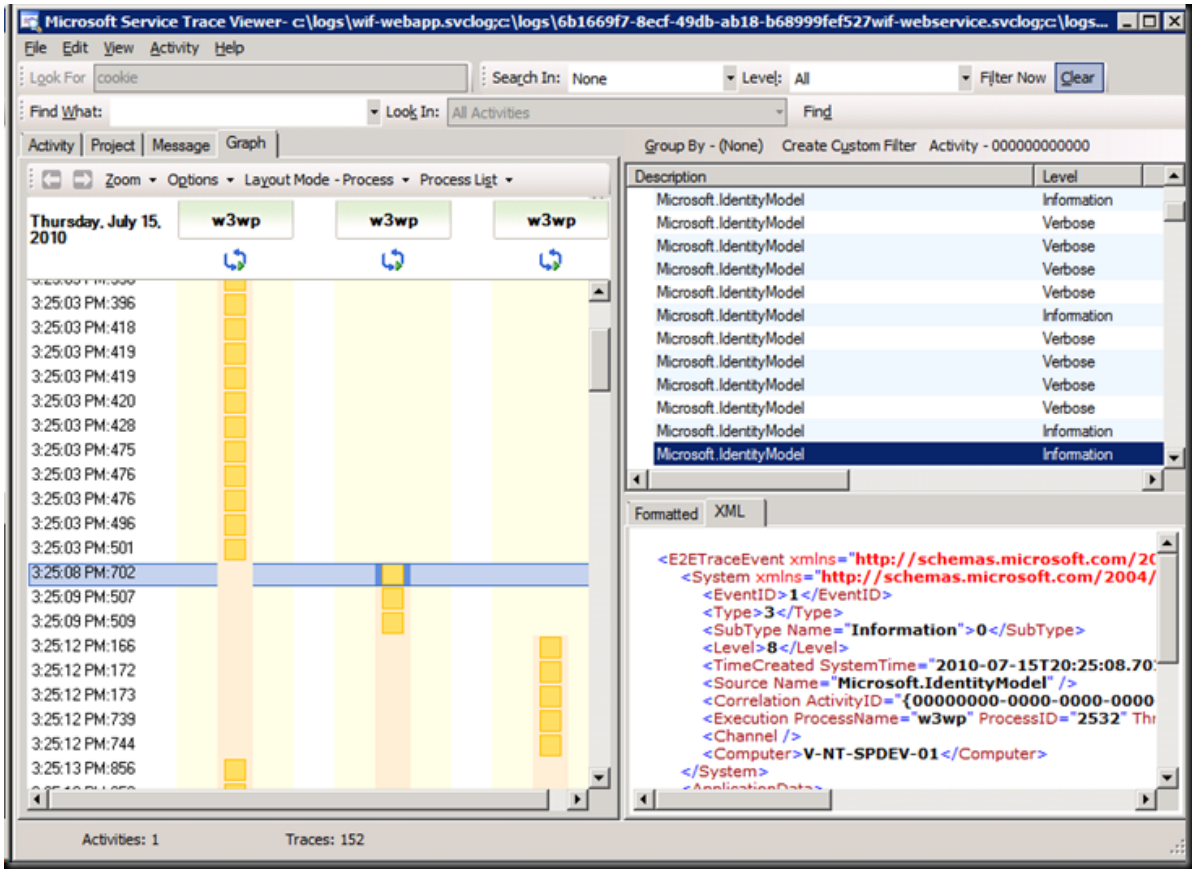
Ensure that different names are provided to the trace files for the SharePoint Server 2010 Web application, or tracing will not work due to file locking by the other App Pool process.

d) Save your changes to the Web.Config.

3. Run the Scenario to initiate the trace in the log directory.
Several new files will be present as a result:



4. Open each of the *.svclog files in the Service Trace Viewer Tool .
   a) On the Service Trace Viewer Tool menu, click **File**, click **Add**, then go to the directory that contains the trace files.
5. On the Service Trace Viewer Tool Activity tab, double click the first activity.
6. The result should resemble the following:



7. Ensure that all tracing is disabled when completed as the *Verbose* trace settings will create large files quickly.

The **Graph** tab will show activities that correlate to each W3WP process. Click the **XML** tab to review the raw trace record.

The following tables (excerpt from the WIF Tracing article (http://go.microsoft.com/fwlink/?LinkID=196691) show which traces are useful for which scenarios.

STS Developer Reference

Token Issuance	Scenario	Trace
WS-Federation	Fails to issue token	HashTrace
	Warning: WS-Federation Message Not Processed	HashTrace
	Success	Token

	RST Received	Token
	POST body (including wresult)	PassiveMessage, WsFedMessage
	HashTraceRecord	HashTrace
WS-Trust	ActAs Request in RST	HashTrace
	RSTR/RST (WCF Traces)	HashTrace

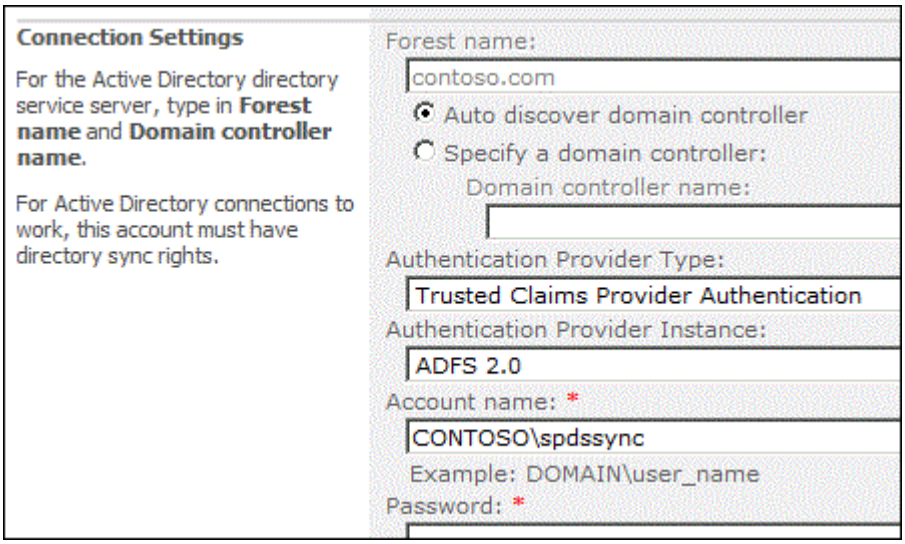
RP Developer Reference

Token Validation	Scenario	Trace
WS-Federation	Fail (Audience URI, Certificate Validation)	HashTrace, Reference
	Warning: WS-Federation Message Not Processed	HashTrace, Reference
	Success	Token
	ClaimsPrincipal	ClaimsPrincipal
	POST body (FedPassive Response)	PassiveMessage, WsFedMessage
	Cookie (Name)	ChunkedCookie
Others	Authorizations	ClaimsPrincipal
	Exceptions	Exception

Trusted Identity Providers and User Profile Synchronization

This section describes the steps that enable user profile synchronization to use claims-based authentication with a trusted identity provider.

When configuring a directory synchronization connection, you can specify the type of authentication provider that will be used to access the imported profiles. In the case of a trusted claims provider, you may also select the specific trusted provider configured in the farm.



To provide the user profile application sufficient information to map the imported profiles to authenticated users, you have to set the imported attributed to the corresponding authenticated users identity claim. The claim is immutable. It may not be changed after it is configured in the trusted identity provider. To map users correctly, the user profile system must be informed of the which of the attributes that it may be importing to be used as the identity claim. The key is to identify the identity claim for the user profile system so that there is enough information to correlate the identity claim with the corresponding profile entry.

The **Claim User Identifier** property is used to establish the mapping.

The above illustration identifies the **Claim User Identifier** property mapped to the ***mail*** incoming attribute. This is an arbitrary claim that is specified in this implementation. Any claim may be utilized. That claim must be the attribute that is being presented as the identity claim from the trusted identity provider.

The next profile import will result in users who are associated to the corresponding profile entries.

Using Audiences with Claims-Based Sites

The Audience functionality is used commonly in SharePoint implementations. The following identifies how SAML claims work with the Audiences feature in SharePoint Server 2010. By default, synchronization support is available for Active Directory Domain Services, a few LDAP sources and from a Lightweight Directory Interchange Format (LDIF) file. For more information, see Configure profile synchronization using a Lightweight Directory Interchange Format (LDIF) file (http://go.microsoft.com/fwlink/?LinkId=229026). A problem is that the account name for most SAML claims users is something like *i:05:t|AD FS with roles|fred@contoso.com*.

To take advantage of audiences, you will need to create profiles for users either manually or through custom code. However, users should be created with their proper claims attributes, for example *i:05:t|AD FS with roles|fred@contoso.com* as the Account Name and then populate the other fields with data that you want to use in your audiences.

After profile are created, Audiences can be created. You cannot use a user-based, such as membership in a group, for the Audience unless you implement custom code. It might be more efficient to use the property-based audience. In my scenario I used the Office field from the profile as the basis for my Audience.

Implications of Claims Mode Authentication on Service Applications

When designing a SharePoint Server 2010 environment to support claims authentication, it is vitally important to understand how your authentication architecture can affect the availability and function of the default SharePoint service applications. Failing to plan for service applications early in the environment design can potentially lead to the solution not being able to fully support certain SharePoint workloads. This section identifies the impact of different claims configurations on SharePoint service applications so that you an account for changes in expected functionality in your environment designs.

Secure Store

The Secure Store is an important service application to plan for if your environment will use claims-based authentication. The Secure Store enables service applications to map SharePoint users to credential sets which can then be used to access external data and services. Some scenarios, such as data refresh in PowerPivot, require credentials stored in the Secure Store to function. In other scenarios, such as authenticating with SQL Server, you may optionally need to use Secure Store credentials to use SQL Server authentication to connect to the SQL Server datasource.

Service applications that take advantage of the Secure Store include:

Excel Services – Excel Services can explicitly use the Secure Store in external data connections and use the Secure Store for unattended data refresh scenarios.

Visio Services – Visio Services can explicitly use the Secure Store in external data connections and use the Secure Store for unattended data refresh scenarios for drawings that use an ODC file to specify the connection. Additionally, Visio objects do not recognize relative URLs at the time of this writing. However, a fully qualified domain name as a link should work properly.

Business Connectivity Services – Business Connectivity Services can use Secure Store credentials with SQL Server, WCF, and .NET external content types. Secure Store credentials cannot be used when WCF external content types use outbound claims authentication.

PerformancePoint Services – Data source connections in PerformancePoint Services can use the Secure Store account with an administrator-configured unattended service account.

PowerPivot – PowerPivot will use Secure Store credentials for an administrator-configured unattended service account for scheduled data refreshes.

The Secure Store in claims-based authentication scenarios will continue to function as it does in traditional Classic mode authentication environments; no additional configuration specific to claims is required. For more information about the Secure Store, please refer to:

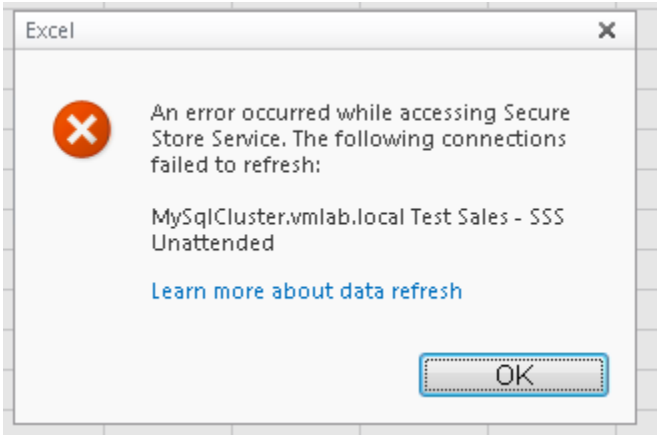
Plan the Secure Store Service (SharePoint Server 2010)
(http://go.microsoft.com/fwlink/?LinkId=229030)

Configure the Secure Store Service (SharePoint Server 2010)
(http://go.microsoft.com/fwlink/?LinkID=205440)

Secure Store and Claims Augmentation

If a custom claims provider has been deployed in the environment, ensure that the claims provider assembly is available on machines running the Secure Store Service. If the assembly is not available, the Secure Store will not be accessible when using claims. This condition can occur when solution packages default to "DeploymentServerType=WebFrontEnd" which results in application servers not receiving the .NET assembly containing the custom claims provider. Because it is common practice to run the Secure Store Service on a separate application server from the SharePoint front-end Web server, the Secure Store will not have access to the custom provider assembly.

As an example, attempting to refresh a connection in Excel Services which takes advantage of the Secure Store will result in a refresh error:



This condition can be detected by setting the SharePoint Foundation - Claims Authentication ULS logging category to verbose. Once configured, the ULS will contain log entries in the "Claims Authentication" category with a description "Error Loading the claim provider assembly…" . To correct the issue, deploy the custom claims provider assembly to the application servers running the Secure Store Service.

Business Connectivity Services

Microsoft Business Connectivity Services is a SharePoint service application which provides Web applications external data access in a consistent manner. Business Connectivity Services is also the only default SharePoint service application which supports outbound claims authentication. Business Connectivity Services supports three types of data connectors which provide Business Connectivity Services connectivity to various line-of-business system types. The choice to use claims authentication with Web applications can change the behavior of Business Connectivity Services depending on the data connector and the specified connection settings. For more information about Business Connectivity Services see Business Connectivity Services overview (SharePoint Server 2010) (http://go.microsoft.com/fwlink/?LinkId=229031).

The following sections outline the authentication behavior of Business Connectivity Services given a combination of Web application claims authentication type, external data source type, and data source connection settings.

External Data Source Type: SQL Data Source

The SQL Data Source provides data access to systems supporting SQL, including SQL Server, Oracle, and others which allow OLEDB and ODBC connections. When using SQL data sources, you have four authentication options, all of which are outlined in the table below:

Note: Business Connectivity Services does not support using outbound claims authentication with SQL data sources.

	User Identity	BDC Identity	Impersonate Windows	Impersonate Custom
Windows-Claims	Automatically attempts connection with Web app IIS application pool Identity	Will connect with Web app IIS application pool identity if RevertToSelf is allowed, error if not	Will use the Secure Store Application to perform NTLM logon. Supports all SSS application types	Will use the Secure Store Application to perform SQL logon Supports all SSS application types
SAML-Claims	Automatically attempts connection with Web app IIS application pool Identity	Will connect with Web app IIS application pool identity if RevertToSelf is allowed, error if not	Will use the Secure Store Application to perform NTLM logon. Supports all SSS	Will use the Secure Store Application to perform SQL logon Supports all SSS

			application types	application types
FBA-Claims	Automatically attempts connection with Web app IIS application pool Identity	Will connect with Web app IIS application pool identity if RevertToSelf is allowed, error if not	Will use the Secure Store Application to perform NTLM logon.	Will use the Secure Store Application to perform SQL logon
			Supports all SSS application types	Supports all SSS application types

Note: when configuring the connection to authenticate with the user's identity, the Business Connectivity Services runtime will use the identity of the hosting process (IIS worker process) and not the user's identity.

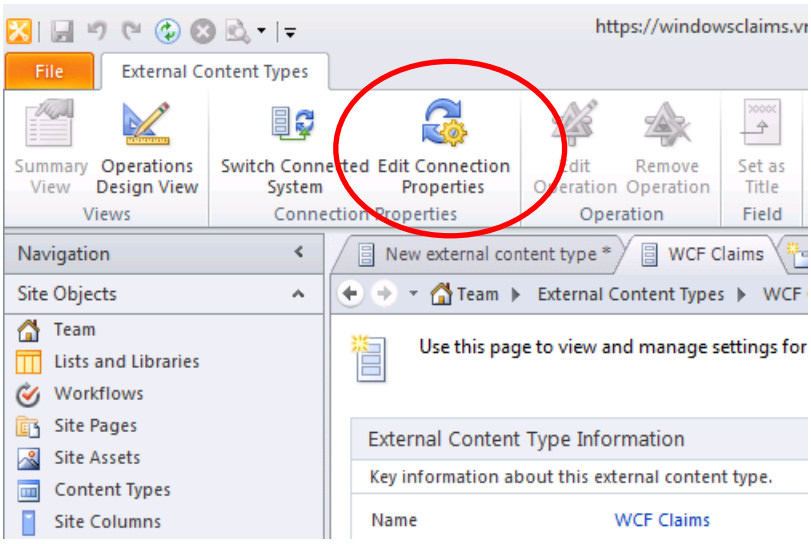
External Data Source Type: WCF Data Source

The WCF data source allows Business Connectivity Services to connect to WCF services by using a number of authentication methods. In addition to traditional Windows authentication models, WCF data sources can natively use outbound claims authentication to authenticate with claims-aware WCF services. To enable outbound claims authentication, you must enable claims in the connection properties of the WCF connection after creating the connection.

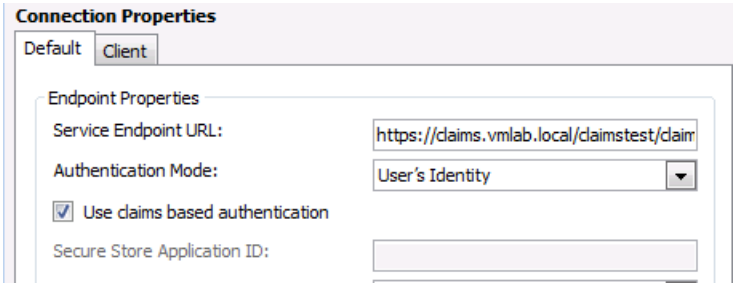
To change the connection settings of the WCF connector:

1) Open the site in Microsoft SharePoint Designer 2010.
2) Open an existing external content type which uses a WCF data source connection or create a new one.

3) Click **Edit Connection Properties**.



4) Select **Use claims based authentication** in the **Endpoint properties** section. Ensure that **Authentication Mode** is set to "User's Identity" to enable the claims authentication option.



*Note: Remember that the "**Use claims based authentication**" option is only available when you choose to edit the connection settings of an existing ECT. This option is not available when initially creating the ECT using a WCF connection.*

	Claims Enabled (in connection settings)	Claims Disabled (in connection settings)	Impersonate Windows	Impersonate Custom
Windows-Claims	Will pass all claims to the WCF service. Note that the Windows Group claims will not be sent as individual claims	Will attempt an anonymous logon to the service. Will result in an authentication error if the service requires	Will impersonate a Windows logon using the Secure Store credentials specified and attempt to authenticate using the method supported by	Will attempt an anonymous logon to the service. Will result in an authentication error if the service requires

	to the service, but instead are compressed into a single claim.	authentication.	the service.	authentication.
SAML-Claims	Will pass all claims to the WCF service. Note that the Windows Group claims will not be sent as individual claims to the service, but instead are compressed into a single claim.	Will attempt an anonymous logon to the service. Will result in an authentication error if the service requires authentication.	Will impersonate a Windows logon using the Secure Store credentials specified and attempt to authenticate using the method supported by the service.	Will attempt an anonymous logon to the service. Will result in an authentication error if the service requires authentication.
FBA-Claims	Will pass all claims to the WCF service. Note that the Windows Group claims will not be sent as individual claims to the service, but instead are compressed into a single claim.	Will attempt an anonymous logon to the service. Will result in an authentication error if the service requires authentication.	Will impersonate a Windows logon using the Secure Store credentials specified and attempt to authenticate using the method supported by the service.	Will attempt an anonymous logon to the service. Will result in an authentication error if the service requires authentication.

External Data Source Type: .NET Data Source

The .NET data source allows the Business Connectivity Services runtime to call static methods on .NET types that are based on a metadata definition (for example, BDC Model) of the type and Business Connectivity Services supported methods. Because the .NET connector data source runs the .NET type in process with the Business Connectivity Services runtime, your code will run under the identity of the process. For example, when calling an external content type by using an external list in a SharePoint site, your code will run with the following credentials:

	Thread.CurrentPrincipal.Identity	WindowsIdentity.Current Properties

	Properties	
Classic Mode Authentication (NTLM/Kerberos)	IsAuthenticated=true AuthenticatedType=Negotiate Name=User's Windows Account Name	IsAuthenticated=true ImpersonationLevel=Impersonation Name=User's Windows Account Name
Windows Claims	IsAuthenticated=true AuthenticatedType=Federation Name=claim representation of account name	IsAuthenticated=true ImpersonationLevel=Impersonation Name=IIS Anonymous User (typically IUSR)
SAML Claims	IsAuthenticated=true AuthenticatedType=Federation Name=claim representation of account name	IsAuthenticated=true ImpersonationLevel=Impersonation Name=IIS Anonymous User (typically IUSR)
FBA Claims	IsAuthenticated=true AuthenticatedType=Federation Name=claim representation of account name	IsAuthenticated=true ImpersonationLevel=Impersonation Name=IIS Anonymous User (typically IUSR)

The important thing to note is the difference in Windows identity between Classic mode and Claims mode. Classic mode will impersonate the client's identity and use this context for the thread executing the request. When accessing the same external content type from a Web application that is configured for claims-based authentication, the Windows identity for the

executing thread will impersonate the IIS anonymous account, typically the IUSR account. This difference in execution context can affect the external content type's ability to access resources.

.NET connectors and the Secure Store

You can also leverage secure store credentials with .NET data sources. For an example of how to leverage the Secure Store declaratively and with code, see: How to: Use the Secure Store Service from within a .NET Connectivity Assembly (http://go.microsoft.com/fwlink/?LinkId=229044).

.NET connectors and the Claims-to-Windows Token Service

The Claims-to-Windows Token Service (C2WTS) can also be leveraged in the .NET connectors but must be called manually in .NET code. The Business Connectivity Services do not natively support the use of the C2WTS by using configuration like other service applications such as Excel Services. For more information about how to use the C2WTS within code, see: Claims to Windows Token Service (c2WTS) Overview (http://go.microsoft.com/fwlink/?LinkId=229045) and S4UClient.UpnLogon Method (Microsoft.IdentityModel.WindowsTokenService) (http://go.microsoft.com/fwlink/?LinkId=229046).

Excel Services

The Web application's claims authentication mode will determine the Excel Services authentication modes that are supported for workbook data connection refresh. The primary design implication is that Excel Services can perform per-user delegated identity authentication only with Windows-Claims and Classic mode authentication. No other form of claims authentication can be used if per-user delegated authentication is required.

Excel Services relies on the Windows Identity Foundation's Claims-to-Windows Service (C2WTS) to translate user claims to Window's tokens for Kerberos delegation. Before attempting to translate the claim by using C2WTS, Excel Services will check the identity provider claim in the user's claims token to determine if the provider is "Windows". If the claim value is Windows, Excel Services will pass the UPN claim value to the C2WTS, translate the claim token to a Windows token, and then use Kerberos delegation to delegate the identity to the data source. If the claim value is not Windows, Excel Services will not attempt to use the C2WTS, regardless of whether a UPN claim exists in the user's claim token or not.

For a step-by-step guide about how to configure Excel Services and the C2WTS to perform Windows integrated Kerberos delegation, refer to Excel Services scenario documented in the whitepaper Configuring Kerberos Authentication for Microsoft SharePoint Server 2010 Products and Technologies (http://go.microsoft.com/fwlink/?LinkId=196600).

Reference: Claims Modes and Data Connection Authentication Options for Excel Services

	Windows Authentication	Secure Store Authentication
Windows Claims	Will use the C2WTS to translate the	Supports all Secure Store

	user's claim into a windows token and perform Kerberos delegation	Application Types.
SamI-Claims	Not supported – Data refresh will result in error	Supports all Secure Store Application Types.
FBA-Claims	Not supported – Data refresh will result in error	Supports all Secure Store Application Types.

PowerPivot for SharePoint

PowerPivot for SharePoint currently does not support Web applications that use any form of claims authentication. You can upload a PowerPivot workbook to a claims-enabled site, but when you try to access PowerPivot functionality in the Web rendered workbook you will receive errors.

PerformancePoint Services

PerformancePoint Services can be used in environments that leverage claims-based authentication methods, but one must plan for how claims authentication affects dashboard authoring and data sources for PerformancePoint Services.

Dashboard Designer

PerformancePoint Dashboard Designer cannot authenticate to sites that leverage SAML or forms-based claims authentication. In addition, Dashboard Designer cannot authenticate to sites that leverage multiple forms of claims authentication on a single zone (for example, Windows-claims and SAML claims). To enable the use of Dashboard Designer in these sites, you must extend the Web application and create a new zone with only Windows Claims authentication. For more information about how to extend a SharePoint Web application, refer to Extend a Web application (SharePoint Server 2010) (http://go.microsoft.com/fwlink/?LinkId=229047).

Data Sources

PerformancePoint Services support multiple types of data sources, each with its own support for various authentication methods. This section will outline each data source, its supported authentication modes, and the effect of claims authentication on each data source.

Some configurations below require the Windows Identity Foundation's Claims-to-Windows Token Service (C2WTS). For more information about how to the configure the C2WTS and Kerberos delegation with PerformancePoint Services, see Configuring Kerberos Authentication for Microsoft SharePoint Server 2010 Products and Technologies (http://go.microsoft.com/fwlink/?LinkId=196600).

SQL Server Analysis Services

SQL Server Analysis Services only supports Windows authentication methods. Therefore, PerformancePoint Services must either connect to Analysis Services with a Secure Store credentials (unattended account) or transition the users' SharePoint claim token to a Windows token using the C2WTS.

Note that Analysis Services can accept the user's principal name in the connection string as custom data when used in conjunction with unattended account authentication. This allows scenarios where per-user identity must be used to filter Analysis Services data with SharePoint sites by using SAML and forms-based claims authentication.

	Windows-Claims	SAML-Claims	Forms-based authentication Claims
Unattended Account	Uses the domain account specified as the unattended account in PerformancePoint Services service application settings.	Uses the domain account specified as the unattended account in PerformancePoint Services service application settings.	Uses the domain account specified as the unattended account in PerformancePoint Services service application settings.
Unattended Account with Custom Data	Uses the domain account specified as the unattended account in PerformancePoint Services service application settings. Passes the current user's principal name to Analysis Services in the custom data field of the connection string.	Uses the domain account specified as the unattended account in PerformancePoint Services service application settings. Passes the current user's principal name to Analysis Services in the custom data field of the connection string.	Uses the domain account specified as the unattended account in PerformancePoint Services service application settings. Passes the current user's principal name to Analysis Services in the custom data field of the connection string.
Per-User Identity	Authenticates with the current user's identity. Requires C2WTS and Kerberos delegation	Not supported	Not supported

Excel Services

The target SharePoint site that hosts the source workbook for Excel Services data sources has to use Classic mode authentication. If the data source's SharePoint site uses any form of claims authentication PerformancePoint Services will not be able to authenticate with that site and connect to the data source regardless of the authentication method of the dashboard site.

	Windows-Claims	SAML-Claims	Forms-based authentication Claims
Unattended Account	Authenticates with the domain account specified as the unattended account in PerformancePoint Services service application settings.	Authenticates with the domain account specified as the unattended account in PerformancePoint Services service application settings.	Authenticates with the domain account specified as the unattended account in PerformancePoint Services service application settings.
Per-User Identity	Authenticates with the current user's identity. Requires C2WTS and Kerberos delegation	Not Supported	Not Supported

Import from Excel Workbook

Importing data from an excel workbook is not affected by the authentication mode of the dashboard site. Data in the workbook is considered static and will not be refreshed by PerformancePoint Services.

SharePoint List

The target SharePoint site that hosts the source list for SharePoint list data sources has to use Classic mode authentication. If the data source's SharePoint site uses any form of claims authentication, PerformancePoint Services will not be able to authenticate with that site and connect to the data source regardless of the authentication method of the dashboard site.

	Windows-Claims	SAML-Claims	Forms-based authentication Claims

Unattended Account	Authenticates with the domain account specified as the unattended account in PerformancePoint Services service application settings.	Authenticates with the domain account specified as the unattended account in PerformancePoint Services service application settings.	Authenticates with the domain account specified as the unattended account in PerformancePoint Services service application settings.
Per-User Identity	Authenticates with the current user's identity. Requires C2WTS and Kerberos delegation	Not Supported	Not Supported

SQL Server Table

SQL Server 2008 R2 or earlier does not natively support claims authentication. Therefore, Performance Point Services must take advantage of the Claims-to-Windows token service to transition claims tokens to Windows tokens for SQL Server authentication.

	Windows-Claims	SAML-Claims	Forms-based authentication Claims
Unattended Account	Uses the domain account specified as the unattended account in PerformancePoint Services service application settings.	Uses the domain account specified as the unattended account in PerformancePoint Services service application settings.	Uses the domain account specified as the unattended account in PerformancePoint Services service application settings.
Per-User Identity	Authenticates with the current user's identity. Requires C2WTS and Kerberos delegation	Not supported	Not supported

SQL Server R2 Reporting Services

SQL Server Reporting Services in SharePoint integration mode does support Web applications which leverage claims authentication, but there are certain constrains one must be aware of when designing SQL Server Reporting Services solution. For more information see: Claims Authentication and Reporting Services (http://go.microsoft.com/fwlink/?LinkId=229062).

Report Builder and Business Intelligence Development Studio

Report Builder and Business Intelligence Development Studio neither support publishing reports to Web applications that use SAML claims, forms-based authentication claims, or multiple claims providers. To publish a report to a library in one of these Web applications you can use one or more of the following alternatives:

- Extend the Web application and create a zone with only Windows-claims authentication.
- Manually copy and upload the report file and associated content to the claims-enabled site.
- Publish to a supported Web application, and then use the library's "send to" function to copy the report file and associated content to the claims-enabled site.

Identity Delegation and Reporting Services

Reporting Services does not natively support PassThrough authentication (delegation) with Web applications that take advantage of claims authentication. Reporting Services does not use the Claims-to-Windows Token Service and cannot perform the needed translation from claims tokens to Windows tokens to perform delegation. Instead, Reporting Services will take advantage of a trusted account as outlined in Claims Authentication and Reporting Services (http://go.microsoft.com/fwlink/?LinkID=229062).

Service Applications and the C2WTS

The Claims-to-Windows Token Service is a Windows Identity Foundation component which enables an application to perform a Windows logon for identity impersonation Services for User (S4U) logon with only a user's UPN. SharePoint Server 2010 takes advantage of the C2WTS to transition Claims-to-Windows token by extracting the user's UPN claim and calling the C2WTS UpnLogon() function. This follows the classic guidelines related to Windows security tokens in that all accounts must be in the same forest or have the appropriate two way Active Directory Domain Services trusts to generate this type of token. There are also important things to understand about the C2WTS in the context of SharePoint Server 2010.

Supported Service Applications

Some service applications do not natively support the C2WTS. Therefore, some service applications are not able to transition to Claims-to-Windows tokens. The following service applications support the C2WTS:

- Excel Services
- Visio Graphics Service

- PerformancePoint Services
- InfoPath Services

In addition, Business Connectivity Services can also use the Claims-to-Windows token service, but this can only be done with custom code within a .NET data source.

C2WTS and SAML/Forms-based authentication claims

Although it is true that the C2WTS only requires a user's UPN to perform the S4ULgon, SharePoint Server 2010 service applications will only allow claims provided by SharePoint's Windows-claims provider to be used with the C2WTS. This means that when users authenticate with Web applications that use SAML claims, service applications that support the C2WTS will not attempt to transition those user's claims, and delegation will fail. SharePoint Server validates the user's claims provider by checking the "IdentityProvider" claim in the user's claims collection. If the IdentityProvider does not equal "Windows", default service applications will not attempt to call the C2WTS.

Using Active Authentication for Custom Development in SharePoint Server 2010 Claims Authentication Web Applications

Authentication in SharePoint Web applications that are secured by claims is done by a process called passive authentication. Passive federation is simply browser-based authentication to an identity provider (IP) security token service (STS). They are based on the WS-Federation Passive Requestor Profile. The authentication request is made to an IP-STS by using the browser with a number of query string parameters that help the IP-STS process the request. They include things like what realm is being used, how to authenticate the person, where to redirect the person after logging in successfully, and so on. The browser is typically redirected between different pages and sites to complete the login process, usually several times, before the SharePoint site is rendered.

Active authentication is done outside the browser, such as from a Windows forms application. In that case, you do not have the same process for logging on, because you are not sending query string parameters nor being redirected between the different pages used in a passive authentication process. As a result, there is a different process to authenticate successfully. Getting authenticated actively is needed before you work with data in a SharePoint site by using the different remote data access features, such as the Client Side Object Model (CSOM), Representational State Transfer (REST), and so on.

Why Claims Authentication is Different

Both Windows and forms-based authentication use a very well understood set of standards for the authentication process. That is why CSOM works when you are logged into a Windows workstation and hit a Windows-secured site. That is also why you there is specific support for forms-based authentication in the CSOM model. So why isn't there specific support for claims authentication also in CSOM? The reality is that when using claims authentication, there are a virtually limitless number of different directions it can go. Any given identity provider along the

way can choose to redirect you, prompt you for credentials, prompt you for a secret key (like two-factor authentication), and so on. So there really isn't a fixed path to get authenticated.

Use CSOM with a claims-secured site is still quite possible. This section will describe how to do so and use CSOM or the REST data interfaces from a Windows forms application.

Working with AD FS 2.0

As described earlier, there are many possible authentication choices when using claims authentication in SharePoint Server 2010. This paper describes the most common approach, which is a SharePoint Server 2010 site using SAML claims that uses AD FS 2.0 (AD FS) as the IP STS. The steps to configure SharePoint Server and AD FS are not described here; instead we assume that a working environment has already been created.

When developing the custom application, the examples described here are going to take advantage of the Windows Identity Foundation (WIF) to connect to and authenticate with AD FS to obtain a SAML token that can then be programmatically presented to SharePoint Server 2010. This is done to obtain a special authentication cookie referred to as the FedAuth cookie. That cookie contains an encrypted version of a user's claims that SharePoint Server uses to determine the resources that a user has rights to access in site. Because AD FS is built on top of WIF, it makes a good choice to incorporate into custom development that needs to authenticate with a system secured by AD FS.

The first step that is necessary to add support for creating the FedAuth cookie programmatically is to enable an additional endpoint in AD FS. To do this, log on to a server on which AD FS is running and do the following:

1. Start the AD FS 2.0 Management application.
2. Expand the **Service** node, and click the **Endpoints** node.
3. Right-click the /ADFS/services/trust/2005/windowstransport node, and then click **Enable** on the shortcut menu.
4. Restart the AD FS 2.0 Windows Service on each server in the AD FS farm.

Creating the Custom Application

For this scenario, start by creating a new Windows forms application project in Visual Studio based on the .NET Framework 3.5. Examples described in this paper should work for both Visual Studio 2008 and Visual Studio 2010. After the project is open, add references to the following assemblies:

- Microsoft.IdentityModel

  This is installed with WIF. The default location is the C:\Program Files\Reference Assemblies\Microsoft\Windows Identity Foundation\v3.5 directory, and the assembly is Microsoft.IdentityModel.dll.

- Microsoft.SharePoint.Client

  This is installed with SharePoint Server 2010 and can be copied from a server that is running SharePoint Server 2010. The default location is the C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\ISAPI directory.

- Microsoft.SharePoint.Client.Runtime

This is installed with SharePoint Server 2010 and can be copied from a server that is running SharePoint Server 2010. The default location is the C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\ISAPI directory.

- System.Web
- System.Runtime.Serialization
- System.IdentityModel
- System.Net
- System.Runtime.Serialization
- System.ServiceModel

In addition to the references above, there is a helper class that is needed to get the SAML token from AD FS. The following code sample represents the helper class.

C# Code

```
using System;
using System.Net.Security;
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.Xml;
using Microsoft.IdentityModel.Protocols.WSTrust;
//**********************************************
//change the namespace to match your application
//**********************************************
namespace ClientOmAuth
{
    [ServiceContract]
    public interface IWSTrustFeb2005Contract
    {
        [OperationContract(ProtectionLevel = ProtectionLevel.EncryptAndSign, Action =
"http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Issue", ReplyAction =
"http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Issue", AsyncPattern = true)]
        IAsyncResult BeginIssue(System.ServiceModel.Channels.Message request, AsyncCallback
callback, object state);
        System.ServiceModel.Channels.Message EndIssue(IAsyncResult asyncResult);
    }

    public partial class WSTrustFeb2005ContractClient : ClientBase<IWSTrustFeb2005Contract>,
IWSTrustFeb2005Contract
    {
        public WSTrustFeb2005ContractClient(Binding binding, EndpointAddress remoteAddress)
            : base(binding, remoteAddress)
        {
        }
        public IAsyncResult BeginIssue(Message request, AsyncCallback callback, object state)
        {
            return base.Channel.BeginIssue(request, callback, state);
        }
        public Message EndIssue(IAsyncResult asyncResult)
        {
            return base.Channel.EndIssue(asyncResult);
        }
    }
    class RequestBodyWriter : BodyWriter
```

```
    {
        WSTrustRequestSerializer _serializer;
        RequestSecurityToken _rst;

        /// <summary>
        /// Constructs the Body Writer.
        /// </summary>
        /// <param name="serializer">Serializer to use for serializing the rst.</param>
        /// <param name="rst">The RequestSecurityToken object to be serialized to the
outgoing Message.</param>
        public RequestBodyWriter(WSTrustRequestSerializer serializer, RequestSecurityToken
rst)
            : base(false)
        {
            if (serializer == null)
                throw new ArgumentNullException("serializer");

            this._serializer = serializer;
            this._rst = rst;
        }
        /// <summary>
        /// Override of the base class method. Serializes the rst to the outgoing stream.
        /// </summary>
        /// <param name="writer">Writer to which the rst should be written.</param>
        protected override void OnWriteBodyContents(XmlDictionaryWriter writer)
        {
            _serializer.WriteXml(_rst, writer, new WSTrustSerializationContext());
        }
    }
}
```

Getting the FedAuth Cookie and Retrieving Data with CSOM

Start by adding the following using statements to the code behind of the form that was added automatically when the project was created.

C# Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.Net;
using System.IO;
using Microsoft.SharePoint.Client;
using System.Web;
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.Security.Principal;
using Microsoft.IdentityModel.Protocols.WSTrust;
using System.Xml;
To demonstrate using CSOM, the following code would be used to retrieve all of the lists
```

```
being used in the site:
ClientContext ctx = new ClientContext(SamlTxt.Text);
//use default credentials
ctx.Credentials = CredentialCache.DefaultCredentials;
//*********************************************************
//this is the important part – where we get the auth cookie
//*********************************************************
ctx.ExecutingWebRequest += new EventHandler<WebRequestEventArgs>(ctx_ExecutingWebRequest);
//get the web
Web w = ctx.Web;
//load lists with all properties
var lists = ctx.LoadQuery(w.Lists);
//execute the query
ctx.ExecuteQuery();
//this is just a simple listbox on a Windows form
SamlList.Items.Clear();
foreach (List theList in lists)
{
SamlList.Items.Add(theList);
}
```

As noted in the code comments, there is one line in particular where the work happens to retrieve the FedAuth cookie. The CSOM ClientContext has an event for which a handler is added – ExecutingWebRequest. That event fires when the call is made to the ExecuteQuery() method. The code there will execute first, it will retrieve the FedAuth cookie, and then the request will be made to the SharePoint site via CSOM. The following sample code examines the ExecutingWebRequest event handler.

C# Code

```
void ctx_ExecutingWebRequest(object sender, WebRequestEventArgs e)
{
try
{
string sharepointUrl = "https://spsServer";
string AD FSUrl = "https://AD FSServer";
//*********************************************
//this is where the FedAuth cookie is retrieved
//*********************************************
string samlToken = GetSamlToken(sharepointUrl, AD FSUrl);
       if (!string.IsNullOrEmpty(samlToken))
       {
CookieContainer cc = new CookieContainer();
Cookie samlAuth = new Cookie("FedAuth", samlToken);
samlAuth.Expires = DateTime.Now.AddHours(1);
samlAuth.Path = "/";
samlAuth.Secure = true;
samlAuth.HttpOnly = true;
Uri samlUri = new Uri(sharepointUrl);
samlAuth.Domain = samlUri.Host;
cc.Add(samlAuth);
e.WebRequestExecutor.WebRequest.CookieContainer = cc;
}
}
catch (Exception ex)
{
//your error handling here
```

```
}
}
```

IN the previous code, a call is made to get the SAML token, which gets the token and a FedAuth cookie from SharePoint Server. After the value of the FedAuth cookie is retrieved from SharePoint Server, a new cookie is created for the request. The same name – FedAuth – is provided that SharePoint Server uses, and the value put into it is the value of the FedAuth cookie retrieved from SharePoint Server.

The GetSamlToken method is invoked to retrieve the SAML token, which is then used to generate the FedAuth cookie from the SharePoint site as shown in the following code sample.

C# Code

```
private string GetSamlToken(string sharepointUrl, string AD FSUrl)
{
string ret = string.Empty;
try
{
//get the SharePoint site address with a trailing slash
string samlServer =
sharepointUrl.Text.EndsWith("/") ? sharepointUrl.Text : sharepointUrl.Text + "/";

       //create a type on the fly to hold information about the SharePoint urls
var sharepointSite = new
       {
Wctx = samlServer + "_layouts/Authenticate.aspx?Source=%2F",
Wtrealm = samlServer,
Wreply = samlServer + "_trust/"
};
       //get the STS server address with a trailing slash
       string stsServer = AD FSUrl.EndsWith("/") ? AD FSUrl : AD FSUrl + "/";
       //this is the endpoint we enabled earlier in the paper
       string stsUrl = stsServer + "AD FS/services/trust/2005/windowstransport";
       //get token from STS
       string stsResponse = GetResponse(stsUrl, sharepointSite.Wreply);
       //make a request to the SharePoint site with the SAML token we
       //got from AD FS (stsResponse) so that it will generate a FedAuth
       //cookie that we can then pass back and use when calling other
       //methods in the site
       string stringData = String.Format("wa=wsignin1.0&wctx={0}&wresult={1}",
          HttpUtility.UrlEncode(sharepointSite.Wctx),
          HttpUtility.UrlEncode(stsResponse));
          HttpWebRequest sharepointRequest =
HttpWebRequest.Create(sharepointSite.Wreply) as HttpWebRequest;
       sharepointRequest.Method = "POST";
       sharepointRequest.ContentType = "application/x-www-form-urlencoded";
       sharepointRequest.CookieContainer = new CookieContainer();
       sharepointRequest.AllowAutoRedirect = false; // This is important
       Stream newStream = sharepointRequest.GetRequestStream();
       byte[] data = Encoding.UTF8.GetBytes(stringData);
       newStream.Write(data, 0, data.Length);
       newStream.Close();
       HttpWebResponse webResponse = sharepointRequest.GetResponse() as HttpWebResponse;

       //get the FedAuth cookie from the response so we can reuse it
       ret = webResponse.Cookies["FedAuth"].Value;
```

```
}
catch (Exception ex)
{
//your error handling here
}
return ret;
}
```

The GetResponse() method should be included in custom code when using the previous sample to retrieve the SAML token from AD FS (see below).

C# Code

```
private string GetResponse(string stsUrl, string realm)
{
RequestSecurityToken rst = new RequestSecurityToken();
rst.RequestType = WSTrustFeb2005Constants.RequestTypes.Issue;
//bearer token, no encryption
rst.AppliesTo = new EndpointAddress(realm);
rst.KeyType = WSTrustFeb2005Constants.KeyTypes.Bearer;
WSTrustFeb2005RequestSerializer trustSerializer =
new WSTrustFeb2005RequestSerializer();
WSHttpBinding binding = new WSHttpBinding();
binding.Security.Mode = SecurityMode.Transport;
binding.Security.Message.ClientCredentialType = MessageCredentialType.None;
binding.Security.Message.EstablishSecurityContext = false;
binding.Security.Transport.ClientCredentialType = HttpClientCredentialType.Windows;
EndpointAddress address = new EndpointAddress(stsUrl);
WSTrustFeb2005ContractClient trustClient =
new WSTrustFeb2005ContractClient(binding, address);

trustClient.ClientCredentials.Windows.AllowNtlm = true;
trustClient.ClientCredentials.Windows.AllowedImpersonationLevel =
    TokenImpersonationLevel.Impersonation;
trustClient.ClientCredentials.Windows.ClientCredential =
CredentialCache.DefaultNetworkCredentials;
System.ServiceModel.Channels.Message response =
trustClient.EndIssue(trustClient.BeginIssue(
System.ServiceModel.Channels.Message.CreateMessage(
MessageVersion.Default, WSTrustFeb2005Constants.Actions.Issue,
new RequestBodyWriter(trustSerializer, rst)), null, null));
trustClient.Close();
XmlDictionaryReader reader = response.GetReaderAtBodyContents();
return reader.ReadOuterXml();
}
```

Calling the GetResponse() method results in a SAML token for the current user that is based on AD FS authenticating the user based on his or her current Windows credentials. After the SAML token has been retrieved, it can then be used in the preceding function (GetSamlToken) to make a connection to SharePoint Server and retrieve the FedAuth cookie. After the FedAuth cookie has been obtained, it can be reused to connect to various endpoints available in the SharePoint site, such as CSOM or REST.

Retrieving REST Data

SharePoint Server 2010 also provides the ability to use a REST interface to retrieve list data. In this example, the code to retrieve the FedAuth cookie is reused and the cookie is reused to make a call to retrieve list data by using REST. The call will be slightly different in that HTTP GETs will be performed directly against the listdata service in SharePoint Server. The listdata service is available within any site by navigating to the _vti_bin directory. For example, to get all of the items in the Contacts list for a site at https://claims, a request could be made to https://claims/_vti_bin/listdata.svc/Contacts. The data is returned as XML, which can then be processed as needed for the application.

The following code sample reuses the method to obtain a FedAuth ticket and then retrieves a list of all items in the Contacts list.

C# Code

```
private void GetRestDataBtn_Click(object sender, EventArgs e)
{
try
{
       //this is the REST endpoint I want to use to get all Contacts
       string endpoint = "https://fc1/_vti_bin/listdata.svc/Contacts";
       //get the FedAuth cookie
       string FedAuth = GetSamlToken();
       //make a request to the REST interface for the data
       HttpWebRequest webRqst = (HttpWebRequest)WebRequest.Create(endpoint);
       webRqst.UseDefaultCredentials = true;
       webRqst.Method = "GET";
       webRqst.Accept = "*/*";
       webRqst.KeepAlive = true;
       //create the FedAuth cookie that will go with our request
       CookieContainer cc = new CookieContainer();
       Cookie samlAuth = new Cookie("FedAuth", FedAuth);
       samlAuth.Expires = DateTime.Now.AddHours(1);
       samlAuth.Path = "/";
       samlAuth.Secure = true;
       samlAuth.HttpOnly = true;
       Uri samlUri = new Uri(SamlTxt.Text);
       samlAuth.Domain = samlUri.Host;
       cc.Add(samlAuth);
       //plug our cookie into the request
       webRqst.CookieContainer = cc;
       //read the response now
       HttpWebResponse webResp = webRqst.GetResponse() as HttpWebResponse;

       //make the request and get the response
       StreamReader theData = new StreamReader(webResp.GetResponseStream(), true);
       string payload = theData.ReadToEnd();
       theData.Close();
       webResp.Close();
       //create the Xml classes for working with the results
//xml doc, loaded with the results
       XmlDocument xDoc = new XmlDocument();
       xDoc.LoadXml(payload);
       //namespace manager, used for querying
       XmlNamespaceManager ns = new XmlNamespaceManager(xDoc.NameTable);
```

```
        ns.AddNamespace("b",
            "http://www.w3.org/2005/Atom");
        ns.AddNamespace("m",
            "http://schemas.microsoft.com/ado/2007/08/dataservices/metadata");
        ns.AddNamespace("d",
            "http://schemas.microsoft.com/ado/2007/08/dataservices");
        //query for items
        XmlNodeList nl = xDoc.SelectNodes("/b:feed/b:entry/b:content/m:properties", ns);
        //create a list to hold the results
        List<Contact> Contacts = new List<Contact>();
        //enumerate the results
        foreach (XmlNode xNode in nl)
        {
            Contacts.Add(new Contact(
xNode.SelectSingleNode("d:FirstName", ns).InnerText,
                xNode.SelectSingleNode("d:LastName", ns).InnerText,
                xNode.SelectSingleNode("d:Company", ns).InnerText,
                xNode.SelectSingleNode("d:JobTitle", ns).InnerText,
                xNode.SelectSingleNode("d:EMailAddress", ns).InnerText));
}
        //bind to the DataGridView on my form
        ContactGrd.DataSource = Contacts;
}
    catch (Exception ex)
    {
//your error handling here
    }
}
public class Contact
{
public string FirstName { get; set; }
public string LastName { get; set; }
public string Company { get; set; }
public string JobTitle { get; set; }
public string Email { get; set; }
public Contact(string First, string Last, string Company, string Title, string Email)
{
this.FirstName = First;
        this.LastName = Last;
        this.Company = Company;
        this.JobTitle = Title;
        this.Email = Email;
}
}
```

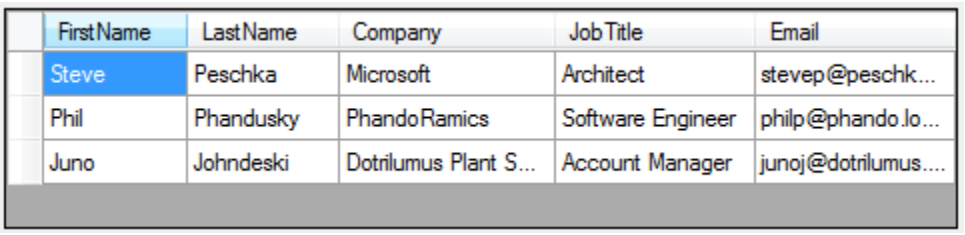
In the previous code sample, a FedAuth cookie is retrieved from SharePoint Server. After it is obtained, a new HttpWebRequest will be used to call the REST interface in SharePoint Server to retrieve all items in the Contacts list. A new cookie is created where the FedAuth cookie that was retrieved can be placed; this is what allows SharePoint Server to view our request as being authenticated. After the cookie has been added, the request is made to the REST interface in SharePoint Server to retrieve the data; the results are put into the string variable payload.

The results returned from SharePoint Server are in XML format as illustrated in the following XML snippet.

XML Snippet

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<feed xml:base="https://fc1/_vti_bin/listdata.svc/"
xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
xmlns="http://www.w3.org/2005/Atom">
  <title type="text">Contacts</title>
  <id>https://fc1/_vti_bin/listdata.svc/Contacts/</id>
  <updated>2010-09-25T14:16:56Z</updated>
  <link rel="self" title="Contacts" href="Contacts" />
  <entry m:etag="W/&quot;1&quot;">
    <id>https://fc1/_vti_bin/listdata.svc/Contacts(1)</id>
    <title type="text">Peschka</title>
    <updated>2010-09-24T16:03:07-07:00</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="ContactsItem" href="Contacts(1)" />
    <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/CreatedBy"
type="application/atom+xml;type=entry" title="CreatedBy" href="Contacts(1)/CreatedBy" />
    <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/ModifiedBy"
type="application/atom+xml;type=entry" title="ModifiedBy" href="Contacts(1)/ModifiedBy" />
    <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Attachments"
type="application/atom+xml;type=feed" title="Attachments" href="Contacts(1)/Attachments" />
    <category term="Microsoft.SharePoint.DataService.ContactsItem"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    <content type="application/xml">
      <m:properties>
        <d:Id m:type="Edm.Int32">1</d:Id>
        <d:ContentTypeID>0x0106003310ED85589F044E96358B975F3381CB</d:ContentTypeID>
        <d:ContentType>Contact</d:ContentType>
        <d:LastName>Jump</d:LastName>
        <d:Modified m:type="Edm.DateTime">2010-09-24T16:03:07</d:Modified>
        <d:Created m:type="Edm.DateTime">2010-09-24T16:03:07</d:Created>
        <d:CreatedById m:type="Edm.Int32">1</d:CreatedById>
        <d:ModifiedById m:type="Edm.Int32">1</d:ModifiedById>
        <d:Owshiddenversion m:type="Edm.Int32">1</d:Owshiddenversion>
        <d:Version>1.0</d:Version>
        <d:Path>/Lists/Contacts</d:Path>
        <d:FirstName>Dan</d:FirstName>
        <d:FullName>Dan Jump</d:FullName>
        <d:EMailAddress>dan.jump@contoso.local</d:EMailAddress>
        <d:Company>Microsoft</d:Company>
        <d:JobTitle>Architect</d:JobTitle>
        <d:BusinessPhone m:null="true" />
        <d:HomePhone m:null="true" />
        <d:MobileNumber m:null="true" />
        <d:FaxNumber m:null="true" />
        <d:Address m:null="true" />
        <d:City m:null="true" />
        <d:StateProvince m:null="true" />
        <d:ZIPPostalCode m:null="true" />
        <d:CountryRegion m:null="true" />
        <d:WebPage>http://www.microsoft.com, http://www.microsoft.com</d:WebPage>
        <d:Notes>&lt;div&gt;&lt;/div&gt;</d:Notes>
      </m:properties>
    </content>
  </entry>
</feed>
```

To process that data, several XML classes are used. An XmlDocument is created to load the XML and provide query services. An XmlNamespaceManager class is used to add the various namespaces that are used in the XML. An XmlNodeList is used to select all the items from the list. An XPath query is executed to populate the XmlNodeList, and then the code enumerates through the results and creates a new instance, named Contact, of a custom class that was created for this sample. The new Contact instance goes in a List<T> of Contacts, and then finally the list is bound to a DataGridView control that is on the Windows form. The results after executing this code is a bound grid of contacts that would appear as follows.



FirstName	LastName	Company	JobTitle	Email
Steve	Peschka	Microsoft	Architect	stevep@peschk...
Phil	Phandusky	PhandoRamics	Software Engineer	philp@phando.lo...
Juno	Johndeski	Dotrilumus Plant S...	Account Manager	junoj@dotrilumus....

Conclusion

SharePoint Server 2010 claims authentication provides a valuable abstraction of the authentication process, but accessing resources in claims-secure sites requires additional steps from Windows authentication sites. This paper provides the following:

- Steps that configure a claims provider trust.
- Steps that create a custom security token service.
- Code to enhance the user experience with a custom people/object picker.
- Environmental nuances depending on identity provider selection and type.
- Code necessary to work with enhancing the rich internet client experience.
- Details useful in migration to this new mode.

Additional Resources

Plan authentication methods (SharePoint Server 2010)
(http://go.microsoft.com/fwlink/?LinkID=190219)

Configure Forms-based Authentication for a Claims-based Web Application
(http://go.microsoft.com/fwlink/?LinkID=184190)

Claims Walkthrough: Creating Claims Providers for Trusted Login Providers for SharePoint Server 2010 (http://go.microsoft.com/fwlink/?LinkId=229068)

Configure the Security Token Service (http://go.microsoft.com/fwlink/?LinkID=205441)

SharePoint and Claims-based Identity (http://go.microsoft.com/fwlink/?LinkID=196647)

A Guide to Claims-Based Identity and Access Control
(http://go.microsoft.com/fwlink/?LinkId=229069)

Claims-Based Identity for Windows (http://go.microsoft.com/fwlink/?LinkID=196776)

Appendix: Overview of a Federation Trust between AD FS 1.X and SharePoint Server 2010

Many organizations have adopted identity federation technologies, specifically AD FS 1.X, and utilize it to provide authentication to Office SharePoint Server 2007. SharePoint Server 2010 includes default support for claims-based authentication (WS-Federation). Additional software, such as the AD FS claims-aware Web agent, is not needed. In addition, the methods of configuring SharePoint Server 2010 to establish an identity federation trust with AD FS 1.X are distinctly different from the methods of configuring AD FS 1.X to work with Office SharePoint Server 2007. Existing reference material is not applicable to SharePoint Server 2010. This section details the methods that are used to establish a federation trust with AD FS 1.X and SharePoint Server 2010 and how to test its operation.

Configuration Steps

The following steps describe how to configure SharePoint Server 2010 to accept identities and authentication from AD FS 1.X in a federation trust. Several assumptions are made about the environment where these steps are being performed. The steps assume that an environment has already been created where both AD FS 1.X and SharePoint Server 2010 have been successfully installed and initially configured. Also, all servers and services to have network connectivity and DNS name resolution and certificate services (token signing and SSL) are assumed.

Prepare SharePoint Server 2010 for Claims-Based Authentication

1. Create a claims-based SharePoint Server 2010 Web application.
    a. On the server running SharePoint Server, create a new Web application by using **Claims Mode Authentication** with a **Fully Qualified Domain Name (FQDN)** and **SSL** support. *(Keep NTLM/KERBEROS [Windows Claims] Authentication. In a production deployment of SharePoint Server 2010, it is recommended to extend the Web application with an extra zone by using NTLM/KERBEROS [Windows Claims] authentication to enable the SharePoint Server 2010 search crawler to access content without requiring a home realm discovery page.)*
    b. Create a top-level site collection for the Web application that you created, and assign a Site Administrator by using Windows Claims authentication (that is, Domain/User).
2. Import the AD FS 1.X token-signing certificate.
    a. Export the AD FS 1.X token signing certificate from the AD FS 1.X federation server.
    b. Copy the token-signing certificate from the federation server to the server running SharePoint Server.
    c. On the server running SharePoint Server, click **Start**, pint to **All programs**, and then click **Microsoft SharePoint Server 2010 Products and SharePoint Management Shell** to run the following Windows PowerShell commands.
        i. Type the following Windows PowerShell command to get the AD FS root certificate:

Windows PowerShell Command

```
$root = New-Object
System.Security.Cryptography.X509Certificates.X509Certificate2("Locatio
n of token signing Certificate Export ")
```

ii. Type the following Windows PowerShell command to add the certificate to the list of SharePoint trusted root authorities:

Windows PowerShell Command

```
New-SPTrustedRootAuthority -Name "AD FS 1.X Token Signing Root
Authority" -Certificate $root
```

d. The server running SharePoint Server should now be able to decrypt tokens that come from the AD FS 1.X service.

NOTE

If the token-signing certificate was created by using a certificate authority root certificate, all certificates in the certificate path, including the root, must be imported into the SharePoint trusted root authority for the token signing cert to be trusted by SharePoint Server 2010. For more information, see Root of Certificate Chain Not Trusted Error with Claims Authentication (http://go.microsoft.com/fwlink/?LinkId=229074).

3. Add a SPTrustedIdentityTokenIssuer entity to SharePoint Server:
   a. To configure SharePoint Server 2010 to accept SAML claims (WS-Federation only), you must create and configure a new SPTrustedIdentityTokenIssuer. On the server running SharePoint Server, click **Start**, point to **All Programs**, and then click **Microsoft SharePoint Server 2010 Products and SharePoint Management Shell**:
      i. Type the following Windows PowerShell commands:

Windows PowerShell Command

```
$siteName = "<SHAREPOINT SERVER 2010 site name>"
$domainFqdn = "<domain host FQDN>"
$AD FSServer = "<AD FS 1.X site name>"
$certPublicKeyFile = "Location of AD FS 1.X token signing Certificate
Export "
$cert = New-Object
System.Security.Cryptography.X509Certificates.X509Certificate2($certPub
licKeyFile)
$map1 = New-SPClaimTypeMapping
"http://schemas.xmlsoap.org/claims/ADFS1/email" -
IncomingClaimTypeDisplayName "EmailAddress"  -SameAsIncoming
```

NOTE

The string for the incoming claim-type description is defined in AD FS v1.X. Custom claims are defined as follows "http:// schemas.xmlsoap.org /claims/<outgoing claim mapping string>". In the previous case, in AD FS v1.X the outgoing custom claim was mapped to the string “AD FS1/email”.

Windows PowerShell Command

```
$map2 = New-SPClaimTypeMapping  "http:// schemas.xmlsoap.org
/claims/ADFS1/role"  -IncomingClaimTypeDisplayName "Role" –
SameAsIncoming
```

NOTE

The string for the incoming claim-type description is defined in AD FS v1.X. Custom claims are defined as follows "http:// schemas.xmlsoap.org /claims/<outgoing claim mapping string>". In the previous case, in AD FS v1.X the outgoing custom claim was mapped to the string “AD FS1/role”.

Windows PowerShell Command

```
New-SPTrustedIdentityTokenIssuer -Name $AD FSServer -Description "AD
FSv1 on $AD FSServer" -Realm "https://$siteName.$domainFqdn/" -
SignInUrl "https://$AD FSServer.$domainFqdn/ADFS/ls/" -
ImportTrustCertificate $cert -ClaimsMappings $map1,$map2  -
IdentifierClaim $map1.InputClaimType
```

b. Close the SharePoint Management Shell after the SPTrustedIdentityTokenIssuer settings have been reviewed.

4. Enable the SharePoint Web application to use SAML Claims AuthN:
   a. On the server running SharePoint Server, click **Start**, point to **All Programs**, click **Microsoft SharePoint Server 2010 Products**, and then click **SharePoint Server 2010 Central Administration**.
   b. In the **Application Management** section, click **Manage web applications**.
   c. Click the Web application created earlier, and then click **Authentication Providers**.
   d. On the Authentication Provider page, click **Default** Zone.
   e. On the Edit Authentication page, scroll to the **Claims Authentication Types** section.
   f. Select **Trusted Identity Provider**, and then **Select Trusted Identity Provider – <AD FS Server name>**.
   g. Leave Windows Authentication selected.
   h. Click **Save** to save changes.
   i. Close SharePoint Server 2010 Central Administration.

Configure AD FS 1.X for Federation Trust with SharePoint Server 2010

1. On the AD FS 1.X server, start the AD FS 1.X Management console. On the **Start** menu, click **Administrative Tools**, and then click **AD FS 1.X Management**.
2. In the AD FS 1.X Management console, select **Organizational Claims**.
3. Create two new custom organizational claims:
    a. Email
    b. Role
4. In the Account Stores/Active Directory, create a custom claim extraction for the two new custom organization claims created earlier.
    a. Map to this Organizational Claim: Email – Attribute: mail
    b. Map to this Organizational Claim: Role – Attribute: memberOf
5. In the Partner Organization/Resource Partner, create a new resource partner by clicking **New** to start new resource partner wizard.
6. On the Import Policy file page, select **NO**, and click **Next**.
7. On the Resource Partner Details page, enter the following:
    a. Display Name: <Enter desired Display Name>
    b. Federation Service URI: <Enter realm value from SHAREPOINT SERVER 2010 SPTrustedIdentityTokenIssuer script> most likely site name i.e. https://<site FQDN> /
    c. Federation Service Endpoint URL: < https://<site FQDN> /_Trust/>
    d. Click **Next**.
8. Choose E-mail as the enabled identity claim.
9. Create Outgoing claim mappings for SharePoint resource partner by right-clicking the resource partner and selecting create outgoing claim mappings. Create the following claim mappings *Note: AD FS v1.X Group and Custom claim mappings have different claim description type generated by AD FS v1.x. All group claim extractions are mapped to "http:// schemas.xmlsoap.org /claims/Group" with the value of the claim equal to the group common name as a string. Custom claims are mapped to the claim description formed by concatenating "http://schemas.xmlsoap.org/claims/" &<Outgoing custom claim name>.
    a. Organizational custom claims: Role - Outgoing custom claims name: AD FS1/role
    b. Organizational custom claims: Email - Outgoing custom claims name: AD FS1/email
10. Close AD FS 1.X management console.