Lab 4

Claims and Web Services

1. This lab extends the guidance in Chapter 8, "*Claims Enabling Web Services*" and Chapter 9, "*Securing REST Services*" of the book "*A Guide to Claims–based Identity and Access Control, 2nd Edition*" (<http://msdn.microsoft.com/en-us/library/ff423674.aspx>). It demonstrates how you can create a desktop client application that uses a claims-based infrastructure to authenticate with both SOAP and REST style services.
2. Both scenarios illustrate the use of active federation: the client is responsible for accessing the claims issuers and obtaining a token that it will present to the Web service (the relying party). The active federation in these lab exercises uses the WS-Trust protocol. This differs from passive federation (using the WS-Federation protocol) where the client (the web browser) is directed through redirect messages to the claims issuers.

# Objectives

After working though this lab, you will understand how to convert a WCF service that expects the desktop client application to send a user name and password into a claims-enabled service. You will also understand how to modify the client desktop client application to authenticate with the service using the WS-Trust protocol. You will see how to make the conversion for both SOAP and REST style services. The final optional exercise in this lab shows you how to configure Microsoft Active Directory Federation Services (ADFS) to support the SOAP-based, active federation scenario.

* 1. **Note:** Before you start working with these exercises, ensure you have run the dependency checking utility. You must also run Visual Studio as an administrator when opening the solutions in these exercises. If you simply double-click the solution file, Visual Studio may fail to load the projects. See the "*Introduction*" document for information about the dependency checker utility and how to run Visual Studio as an administrator.

This lab contains the following exercises:

* + [Exercise 1](#_Exercise_1:_Using): **Using Claims with SOAP Web Services**. In this exercise, you will modify the SOAP-based Adatum a-Order web service to use claims. You will also modify the desktop client application to work with the new version of the service.
  + [Exercise 2](#_Exercise_2:_Using): **Using Claims with REST Web Services**. In this exercise, you will modify the REST-based Adatum a-Order web service to use claims. You will also modify the desktop client application to work with the new version of the service.
  + [Exercise 3](#_Exercise_3:_XXXXX): **Federation with ADFS**. In this optional exercise, you will replace the custom Adatum federation provider with ADFS.

# Exercise 1: Using Claims with SOAP Web Services

In this exercise, you will modify the SOAP-based Adatum a-Order web application to trust the Adatum federation provider, and configure the Adatum federation provider to trust the Litware identity provider. You will modify the client to use federated authentication when it connects to the Web service.

This exercise contains the following tasks:

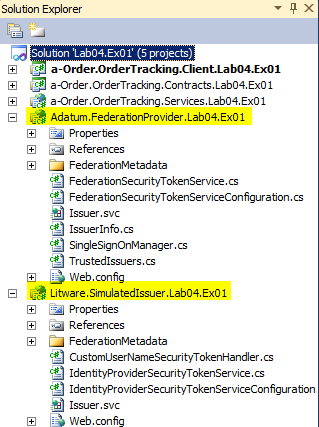
* + [Task 1](#_Task_1:_Modify): Modify the WCF binding configuration in the service.
  + [Task 2](#_Task_2:_Add): Add WIF to the service.
  + [Task 3](#_Task_3:_Change): Change the Authorization manger.
  + [Task 4](#_Task_4:_Modify): Modify the client binding.
  + [Task 5](#_Task_5:_Verify): Verify the solution

You should be able to complete this exercise in approximately 30 minutes.

## Task 1: Modify the Service Binding Configuration in the Service

* 1. In this task, you will first add the Adatum federation provider and Litware identity provider projects to the solution before modifying the binding information in the a-Order Web.config file. The modifications to the bindings in the Web.config file will federate the a-Order Web service with the Adatum federation provider that will be responsible for authenticating users on behalf of the a-Order application. In this federated scenario, the a-Order Web service will now receive a SAML token instead of a username and password.
  2. The Adatum federation provider and Litware identity provider are simulated issuers. In a 'real' scenario, they would already exist in these organizations.

To modify the service binding configuration

* 1. In Windows Explorer, copy the **Adatum.FederationProvider.Lab04.Ex01** folder and the **Litware.SimulatedIssuer.Lab04.Ex01** folder **(including all of their contents)** from the **Lab04-ActiveClient\Source\Ex01\Assets** folder to the **Lab04-ActiveClient\Source\Ex01\Begin** folder.
     1. The project in the **Adatum.FederationProvider.Lab04.Ex01** folder contains a sample STS that is configured to act as a federation provider, and the **Litware.SimulatedIssuer.Lab04.Ex01** folder contains a sample STS that is configured to act as an identity provider — these projects were created from the WIF **WCF Security Token Service** project template.
  2. Start Visual Studio as an administrator and open the solution named **Lab04.Ex01.sln** from the **Lab04- ActiveClient\Source\Ex01\Begin** folder.
  3. **In Visual Studio Solution Explorer, right-click on the top-level Lab04.Ex01 solution item, point to Add, and click Existing Project. Navigate to the folder   
     Lab04-ActiveClient\Source\Ex01\Begin\Adatum.FederationProvider.Lab04.Ex01, select the file named Adatum.FederationProvider.Lab04.Ex01.csproj, and click Open. The Adatum.FederationProvider.Lab4.Ex01** project now appears in Solution Explorer.
  4. **In Visual Studio Solution Explorer, right-click on the top-level Lab04.Ex01 solution item, point to Add, and click Existing Project. Navigate to the folder   
     Lab04-ActiveClient\Source\Ex01\Begin\Litware.SimulatedIssuer.Lab04.Ex01, select the file named Litware.SimulatedIssuer.Lab04.Ex01.csproj, and click Open. The Litware.SimulatedIssuer.Lab04.Ex01** project now appears in Solution Explorer.
     1. 
  5. In Visual Studio, open the file **Web.config** from the **a-Order.OrderTracking.Services.Lab04.Ex01** project into the code editor window.
  6. As shown in the highlights in the following snippet, replace the existing **wsHttpBinding** endpoint and bindings to federate with the Adatum federation provider. This will enable users to authenticate with the service by providing a SAML token issued by the Adatum issuer.
     1. XML
     2. <services>
     3. <service name="AOrder.OrderTracking.Services.OrderTrackingService"
     4. behaviorConfiguration="serviceBehavior">
     5. **<endpoint address="" binding="ws2007FederationHttpBinding"**
     6. **bindingConfiguration="WS2007FederationHttpBinding\_IOrderTrackingService"**
     7. contract="AOrder.OrderTracking.Contracts.IOrderTrackingService" />
     8. <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
     9. </service>
     10. </services>
     11. <bindings>
     12. **<ws2007FederationHttpBinding>**
     13. **<binding name="WS2007FederationHttpBinding\_IOrderTrackingService">**
     14. **<security mode="TransportWithMessageCredential">**
     15. **<message>**
     16. **<issuer**
     17. **address="https://localhost/Adatum.FederationProvider.Lab04.Ex03/Issuer.svc"**
     18. **binding="ws2007HttpBinding"></issuer>**
     19. **</message>**
     20. **</security>**
     21. **</binding>**
     22. **</ws2007FederationHttpBinding>**

</bindings>

* 1. Close all of the files open in the Visual Studio editor, making sure to save your changes, but keep the solution open so that you are ready to use it in the next task.
  2. You have now completed this task and you should know how to replace the **wsHttpBinding** in the Web.config file with a **ws2007FederationHttpBinding** that references a claims issuer.

## Task 2: Add WIF to the service

* 1. In this task, you will add WIF to the service to handle the token provided by the client. The service will read the claims in the user token and add the claims to the **IClaimsPrincipal** object in the service.

To add WIF to the service

* 1. Continue with the **Lab04.Ex01** solution you used in the previous task.
  2. In Visual Studio, in the **a-Order.OrderTracking.Services.Lab04.Ex01** project, right-click on the **References** folder, and then click on **Add Reference**.
  3. In the **Add Reference** dialog, select the **.NET** tab, and locate the **Microsoft.IdentityModel** component. Clicking on **Component Name** will sort the components alphabetically. Select **Microsoft.IdentityModel**, and click **OK**. This adds a reference to the WIF assembly to the project.
  4. In Visual Studio, open the file **Web.config** from the **a-Order.OrderTracking.Services.Lab04.Ex01** project into the code editor window.
  5. As shown in the following snippet, add the **federatedServiceHostConfiguration** behavior extension to the Web.config file.
     1. **Important**: The value of the type attribute that starts with "Microsoft.IdentityModel…" and is shown split over multiple lines, must be entered on a single in the Web.config file.
     2. XML
     3. </bindings>
     4. **<extensions>**
     5. **<behaviorExtensions>**
     6. **<add name="federatedServiceHostConfiguration" type=**
     7. **"Microsoft.IdentityModel.Configuration.ConfigureServiceHostBehaviorExtensionElement,**
     8. **Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral,**
     9. **PublicKeyToken=31bf3856ad364e35" />**
     10. **</behaviorExtensions>**
     11. **</extensions>**
     12. <behaviors>
     13. By adding this behavior extension, WIF attaches to WCF pipeline processing to verify the token.
  6. As shown in the following snippet, add the **federatedServiceHostConfiguration** element to the service behavior.
     1. XML
     2. <behaviors>
     3. <serviceBehaviors>
     4. <behavior name="serviceBehavior">
     5. **<federatedServiceHostConfiguration />**
     6. <serviceDebug includeExceptionDetailInFaults="true" />
     7. <serviceMetadata httpGetEnabled="true" />

</behavior>

* + 1. </serviceBehaviors>
    2. </behaviors>
  1. As shown in the following highlighted code, add a **microsoft.identityModel** section to the Web.config file to configure WIF in the a-Order application. This code configures the Adatum federation provider as a trusted issuer for the a-Order application.
     1. XML
     2. </system.serviceModel>
     3. **<microsoft.identityModel>**
     4. **<service>**
     5. **<issuerNameRegistry**
     6. **type="Microsoft.IdentityModel.Tokens.ConfigurationBasedIssuerNameRegistry,**
     7. **Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral,**
     8. **PublicKeyToken=31bf3856ad364e35">**
     9. **<trustedIssuers>**
     10. **<add thumbprint="f260042d59e14817984c6183fbc6bfc71baf5462" name="adatum" />**
     11. **</trustedIssuers>**
     12. **</issuerNameRegistry>**
     13. **<audienceUris>**

**<add value="https://localhost/a-Order.OrderTracking.Services.Lab04.Ex01/OrderTrackingService.svc"/>**

* + 1. **</audienceUris>**
    2. **<serviceCertificate>**
    3. **<certificateReference findValue="CN=adatum" storeLocation="LocalMachine"**
    4. **storeName="My" x509FindType="FindBySubjectDistinguishedName" />**
    5. **</serviceCertificate>**
    6. **</service>**
    7. **</microsoft.identityModel>**

</configuration>

* 1. Close all of the files open in the Visual Studio editor, making sure to save your changes, but keep the solution open so that you are ready to use it in the next task.
  2. You have now completed this task and you should know modify the Web.config file to use WIF to define a trusted issuer for the relying party application.

## Task 3: Change the Authorization manger

* 1. In this task, you will remove the existing authorization manager from the a-Order application and replace it with a claims-based authorization manager.

To change the Authorization manger

* 1. Continue with the **Lab04.Ex01** solution you used in the previous task.
  2. In Visual Studio, open the file **Web.config** from the **a-Order.OrderTracking.Services.Lab04.Ex01** project into the code editor window.
  3. Remove the existing configuration information for authorization from the Web.config file. The highlighted code shows the configuration that you should remove.
     1. XML
     2. <serviceBehaviors>
     3. <behavior name="serviceBehavior">
     4. <federatedServiceHostConfiguration />
     5. <serviceDebug includeExceptionDetailInFaults="true" />
     6. <serviceMetadata httpGetEnabled="true" />
     7. **<serviceCredentials>**
     8. **<userNameAuthentication**

**userNamePasswordValidationMode="Custom" …**

* + 1. **</serviceCredentials>**
    2. **<serviceAuthorization**
    3. **principalPermissionMode="Custom" …**
    4. **<authorizationPolicies>**
    5. **<add policyType="AOrder.OrderTracking.Services.SimpleAuthorizationPolicy, …**
    6. **</authorizationPolicies>**
    7. **</serviceAuthorization>**
    8. </behavior>
    9. </serviceBehaviors>
  1. In Solution Explorer, right-click the **SimpleAuthorizationManager.cs** file, click **Delete**, and then click **OK**. This file is no longer required.
  2. In Solution Explorer, right-click the **SimpleAuthorizationPolicy.cs** file, click **Delete**, and then click **OK**. This file is no longer required.
  3. In Solution Explorer, right-click the **SimpleUserNamePasswordValidator.cs** file, click **Delete**, and then click **OK**. This file is no longer required.
  4. In Solution Explorer, right-click the **UserInformation.cs** file, click **Delete**, and then click **OK**. This file is no longer required.
  5. In Solution Explorer, right-click the **UserRepository.cs** file, click **Delete**, and then click **OK**. This file is no longer required.
     1. The files you have just removed contained code to handle the user name and password authentication and authorization used in the original scenario before it was adapted to use claims.
  6. In Solution Explorer, right-click the **a-Order.OrderTracking.Services.Lab04.Ex01** project, click **Add**, and then click **Class**. In the **Name** field, type **SimpleClaimsAuthorizationManager.cs**, and then click **Add**.
  7. In the code editor, replace the contents of the file with the following code. This defines a **CheckAccess** method to check if the caller is in the **OrderTracking** role. Membership of this role is specified in the claims sent to the Web service by the client application.
     1. C#
     2. namespace AOrder.OrderTracking.Services
     3. {
     4. using Microsoft.IdentityModel.Claims;
     5. using Samples.Web.ClaimsUtilities;
     6. public class SimpleClaimsAuthorizationManager : ClaimsAuthorizationManager
     7. {
     8. public override bool CheckAccess(AuthorizationContext context)
     9. {
     10. return context.Principal.IsInRole(Adatum.Roles.OrderTracker);
     11. }
     12. }
     13. }
  8. In Visual Studio, open the file **OrderTrackingService.svc.cs** file from the **a-Order.OrderTracking.Services.Lab04.Ex01** project into the code editor window. Using the highlighted code in the following sample, replace the existing code that relied on the username and password authentication with code that retrieves the organization name from a claim.
     1. C#
     2. namespace AOrder.OrderTracking.Services
     3. {
     4. using System.Linq;
     5. using AOrder.OrderTracking.Contracts;
     6. using AOrder.OrderTracking.Contracts.Data;
     7. **using Samples.Web.ClaimsUtilities;**
     8. public class OrderTrackingService : IOrderTrackingService
     9. {
     10. public Order[] GetOrdersFromMyOrganization()
     11. {
     12. **string organization =**
     13. **ClaimHelper.GetCurrentUserClaim(Adatum.ClaimTypes.Organization).Value;**
     14. var repository = new OrderRepository();
     15. return repository.GetOrdersByCompanyName(organization).ToArray();
     16. }
     17. }
     18. }
  9. In Visual Studio, open the file **Web.config** from the a-Order.OrderTracking.Services.Lab04.Ex01 project into the code editor window.
  10. Using the highlighted code in the following snippet, add the **SimpleClaimsAuthorizationManager** class to the WIF configuration.
      1. XML
      2. <microsoft.identityModel>
      3. <service>
      4. …
      5. **<claimsAuthorizationManager**
      6. **type="AOrder.OrderTracking.Services.SimpleClaimsAuthorizationManager,**
      7. **AOrder.OrderTracking.Services" />**
      8. </service>
      9. </microsoft.identityModel>
  11. In Solution Explorer, right-click the **Lab04.Ex01** solution item and click **Rebuild solution**. Correct any errors.
  12. Close all of the files open in the Visual Studio editor, making sure to save your changes, but keep the solution open so that you are ready to use it in the next task.
  13. You have now completed this task. The changes that you made illustrate the changes that you need to make to a SOAP-based Web service that uses a legacy, username and password, authentication scheme to convert it to use claims in a federated identity environment.

## Task 4: Modify the client binding

* 1. In this task, you will modify the desktop client to work with your claims-enabled SOAP-based Web service. The only changes you need to make are to the application configuration. WIF and the WS-Trust protocol will handle the process of obtaining a token from an identity provider and passing this token to the Web service.

To modify the client binding

* 1. Continue with the **Lab04.Ex01** solution you used in the previous task.
  2. In Visual Studio, in the **a-Order.OrderTracking.Client.Lab04.Ex01** project, right-click on the **References** folder, and then click on **Add Reference**.
  3. In the **Add Reference** dialog, select the **.NET** tab, and locate the **Microsoft.IdentityModel** component. Clicking on **Component Name** will sort the components alphabetically. Select **Microsoft.IdentityModel**, and click **OK**. This adds a reference to the WIF assembly to the project.
  4. In Visual Studio, open the file **App.config** from the **a-Order.OrderTracking.Client.Lab04.Ex01** project into the code editor window.
  5. Delete the **wsHttpBinding** highlighted in the following snippet.
     1. XML
     2. <bindings>
     3. **<wsHttpBinding>**
     4. **<binding name="WSHttpBinding\_IOrderTrackingService">**
     5. **<security mode="TransportWithMessageCredential">**
     6. **<message clientCredentialType="UserName" establishSecurityContext="false" />**
     7. **</security>**
     8. **</binding>**
     9. **</wsHttpBinding>**
     10. </bindings>
  6. Add the new **ws2007FederationHttpBinding** highlighted in the following snippet to the App.config file. The client uses this binding to connect to the a-Order SOAP-based Web service.
     1. XML
     2. <bindings>
     3. **<ws2007FederationHttpBinding>**
     4. **<binding name="WS2007FederationHttpBinding\_IOrderTrackingService">**
     5. **<security mode="**TransportWithMessageCredential**">**
     6. **<message>**
     7. **<issuer**
     8. **address=**
     9. **"https://localhost/Adatum.FederationProvider.Lab04.Ex01/Issuer.svc"**
     10. **binding="customBinding" bindingConfiguration="AdatumIssuerIssuedToken">**
     11. **</issuer>**
     12. **</message>**
     13. **</security>**
     14. **</binding>**
     15. **</ws2007FederationHttpBinding>**
     16. </bindings>
  7. This binding uses a custom binding to connect to the Adatum federation provider that is responsible for converting tokens and claims from an identity provider into the token and claims that the a-Order Web service expects to see. Add the custom binding to the App.config as shown in the following highlighted snippet.
     1. XML
     2. <bindings>

…

* + 1. **<customBinding>**
    2. **<binding name="AdatumIssuerIssuedToken">**
    3. **<security authenticationMode="IssuedTokenOverTransport" messageSecurityVersion=**
    4. **"WSSecurity11WSTrust13WSSecureConversation13WSSecurityPolicy12BasicSecurityProfile10"**
    5. **>**
    6. **<issuedTokenParameters>**
    7. **<issuer**
    8. **address="https://localhost/Litware.SimulatedIssuer.Lab04.Ex01/Issuer.svc"**
    9. **binding="ws2007HttpBinding"**
    10. **bindingConfiguration="LitwareIssuerUsernameMixed"></issuer>**
    11. **</issuedTokenParameters>**
    12. **</security>**
    13. **<httpsTransport />**
    14. **</binding>**
    15. **</customBinding>**
    16. </bindings>
  1. This binding uses a **ws2007HttpBinding** binding to connect to the Litware identity provider that is responsible for authenticating the user and issuing a token. Add the **ws2007HttpBinding** binding to the App.config as shown in the following highlighted snippet.
     1. XML
     2. <bindings>

…

* + 1. **<ws2007HttpBinding>**
    2. **<binding name="LitwareIssuerUsernameMixed">**
    3. **<security mode="TransportWithMessageCredential">**
    4. **<message clientCredentialType="UserName" establishSecurityContext="false" />**
    5. **</security>**
    6. **</binding>**
    7. **</ws2007HttpBinding>**

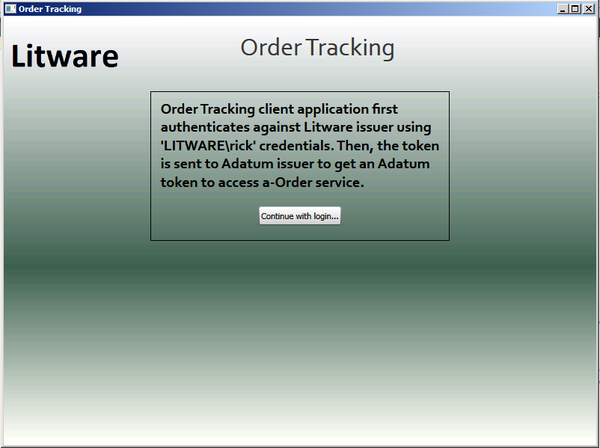
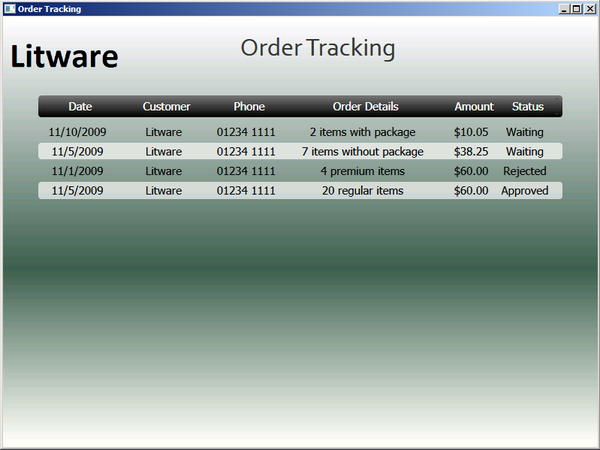
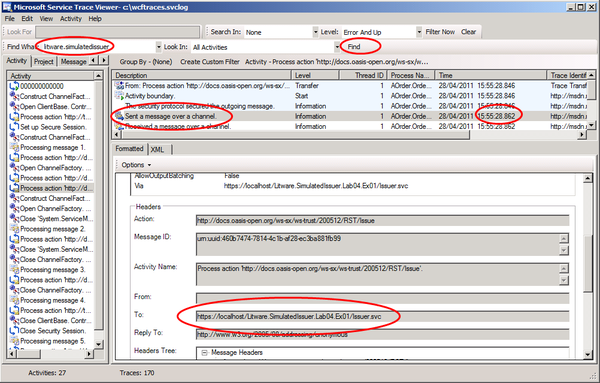
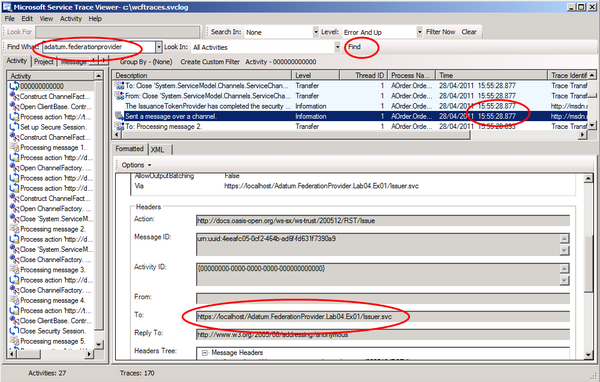
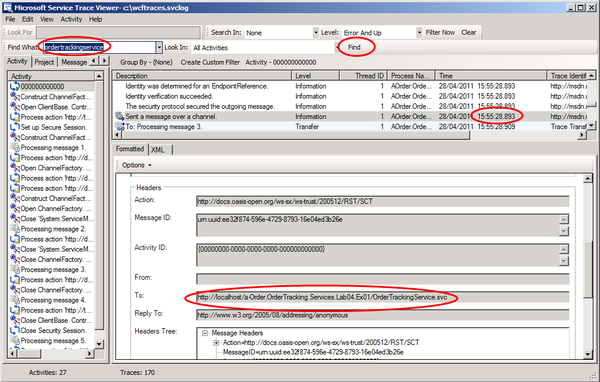
</bindings>

* + 1. The three bindings you have added to the App.config file implement an *active federation* scenario: the client first obtains a token from an identity provider, then the client send the token to a federation provider where the tokens and claims are transformed, and finally the client forwards the token to the relying party application (in this case, the a-Order application.
  1. Modify the a-Order endpoint to use the new bindings as shown in the following snippet.
     1. XML
     2. <client>
     3. **<endpoint**
     4. **address=**
     5. **"https://localhost/a-Order.OrderTracking.Services.Lab04.Ex01/OrderTrackingService.svc"**
     6. **binding="ws2007FederationHttpBinding"**
     7. **bindingConfiguration="WS2007FederationHttpBinding\_IOrderTrackingService"**
     8. **contract="OrderTrackingService.IOrderTrackingService"**
     9. **name="WS2007FederationHttpBinding\_IOrderTrackingService">**
     10. **<identity>**
     11. **<dns value="adatum" />**
     12. **</identity>**
     13. **</endpoint>**
     14. </client>
  2. In Solution Explorer, right-click the **Lab04.Ex01** solution item and click **Rebuild solution**. Correct any errors.
  3. Close all of the files open in the Visual Studio editor, making sure to save your changes, but keep the solution open so that you are ready to use it in the next task.
  4. You have now completed this task. The changes that you made to the client application configure the client to use WS-Trust protocol in an active federation scenario. The client will first authenticate with an identity provider (Litware in this scenario), then use the Adatum federation provider to convert the token and claims from Litware into a format that the a-Order application expects, and then forward the token (in a SOAP message) to the a-Order application.

## Task 5: Verify the Solution

* 1. In this task, you will verify the solution by running the solution and using WCF tracing to examine the interaction between the client, the identity provider, the federation provider, and the Web service.

To verify the solution

* 1. In Visual Studio, on the **Debug** menu, click **Start Without Debugging**. In the Order Tracking application, click **Continue with login**.
     1. 
  2. The client application authenticates with the Litware issuer, obtains a token from the Adatum federation provider, and connects to the Web service. The Web service returns Litware orders for the client application to display.
     1. 
  3. To examine the messages that the client application exchanges with the Web services, you can use the WCF trace facility. Continue with the **Lab04.Ex01** solution you used in the previous task. In Visual Studio, open the file **App.config** from the **a-Order.OrderTracking.Client.Lab04.Ex01** project into the code editor window.
  4. Add the **system.diagnostics** section highlighted in the following snippet, and save the changes.
     1. XML
     2. **<system.diagnostics>**
     3. **<sources>**
     4. **<source name="System.ServiceModel"**
     5. **switchValue="Information, ActivityTracing"**
     6. **propagateActivity="true">**
     7. **<listeners>**
     8. **<add name="xml"/>**
     9. **</listeners>**
     10. **</source>**
     11. **<source name="System.ServiceModel.MessageLogging">**
     12. **<listeners>**
     13. **<add name="xml"/>**
     14. **</listeners>**
     15. **</source>**
     16. **</sources>**
     17. **<sharedListeners>**
     18. **<add name="xml"**
     19. **type="System.Diagnostics.XmlWriterTraceListener"**
     20. **initializeData= "c:\WCFTraces.svclog" />**
     21. **</sharedListeners>**
     22. **</system.diagnostics>**
     23. </configuration>
     24. Make sure that you have permission to write to the location you have chosen to write your log file.
  5. Repeat steps 1 and 2 to run the application and collect the trace data.
  6. In Windows Explorer, double click on the **WCFTraces.svclog** file that contains your trace data.
     1. In the WS-Trust process, the application first connects to the identity provider — the Litware issuer. In the **Find What** box, type **Litware.simulatedissuer** and click **Find**. Note the time when the client application called the Litware issuer.  
        
     2. When entering values in the **Find What** text box make sure that you remove any trailing space after the value. If you copy and paste into the text box it may add a space that prevents the search from finding any matching entries.
  7. The client application next connects to the federation provider — the Adatum issuer. In the **Find What** box, type **Adatum.FederationProvider** and click **Find**, keep clicking **Find** until you find a **Sent a message over a channel** message (You may need to click on the first activity in the **Activity** list to start the search from the beginning of the trace file). Note the time when the client application called the Adatum federation provider: it will be later than the time that it called the Litware issuer.
     1. 
  8. The client application finally connects to the a-Order service. In the **Find What** box, type **ordertrackingservice** and click **Find**, keep clicking **Find** until you find a **Sent a message over a channel** message (You may need to click on the first activity in the **Activity** list to start the search from the beginning of the trace file). Note the time when the client application called the service: it will be later than the time that it called the Adatum federation provider.
     1. 
  9. You have now completed this task and this exercise. You have seen how to configure a federated identity environment: users from Litware can access the a-Order application using credentials from their own identity provider.
  10. Federated identity is a powerful feature of claims; users do not need to remember additional sets of credentials to access a partner's application, and it is easy for an organization such as Adatum to configure its federation provider to trust issuers from additional partners without the requirement to modify the relying party application. Furthermore, the relying party application no longer has the responsibility for verifying the user's username and password: this is now handled by an identity provider.

## Running the "End" Solution

1. If you did not complete all of the tasks in this exercise, you can run the provided "end" solution.

To run the end solution

* 1. Start Visual Studio 2010 as an administrator.
  2. Open the solution named **Lab04.Ex01.End** from the folder **Lab04-ActiveClient\Source\Ex01\End**.
  3. Open the **Build** menu and click **Rebuild Solution**.
  4. On the **Debug** menu, click **Start Without Debugging** to run the application.
  5. You can then simulate logging in as Rick from Litware.
  6. If you want to see the WCF trace and explore the message flow in the solution, follow the steps in [Task 5, Verify the Solution](#_Task_5:_Verify).

# Exercise 2: Using Claims with REST Web Services

In this exercise, you will modify the RESTful Adatum a-Order web application to trust the Adatum federation provider, and configure the Adatum federation provider to trust the Litware identity providers. You will modify the client to use federated authentication when it connects to the Web service.

This exercise contains the following tasks:

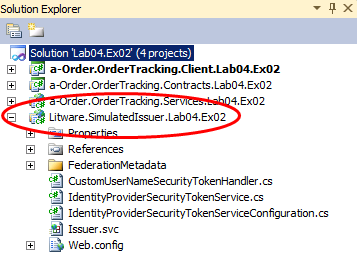
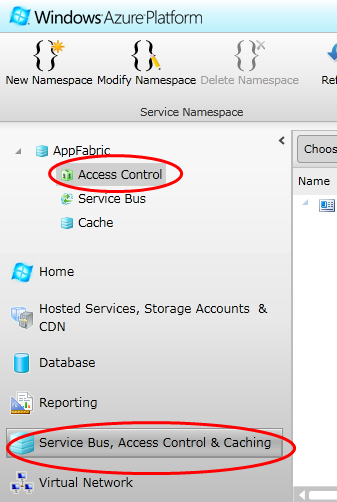
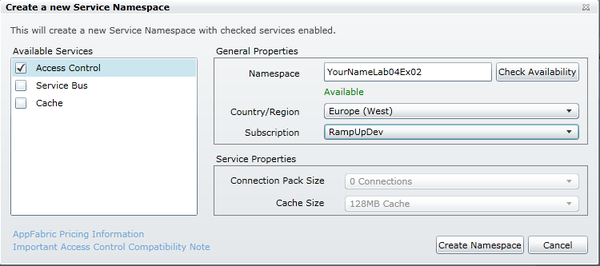
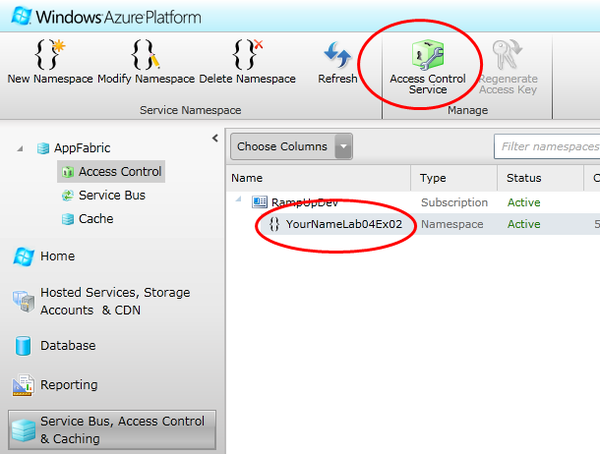
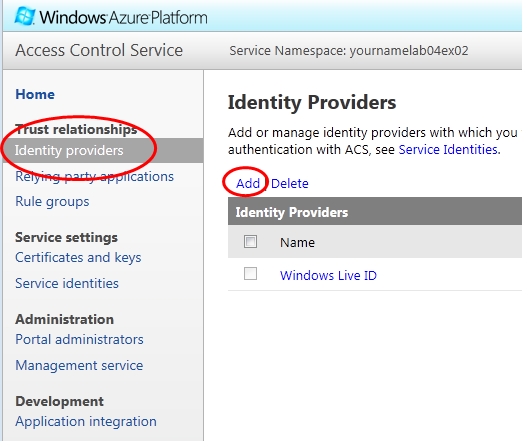
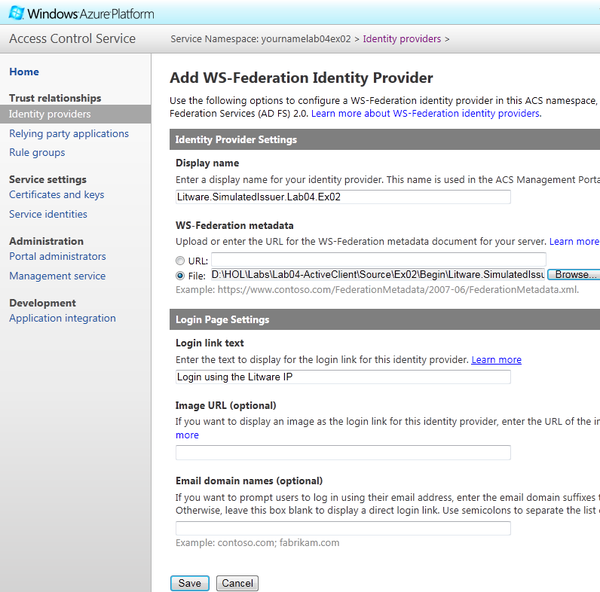
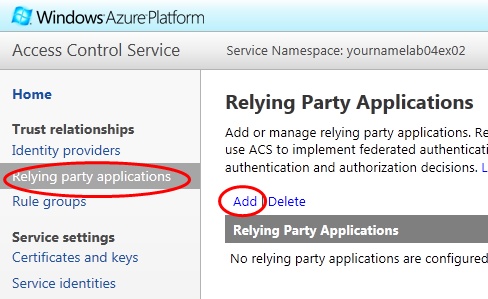
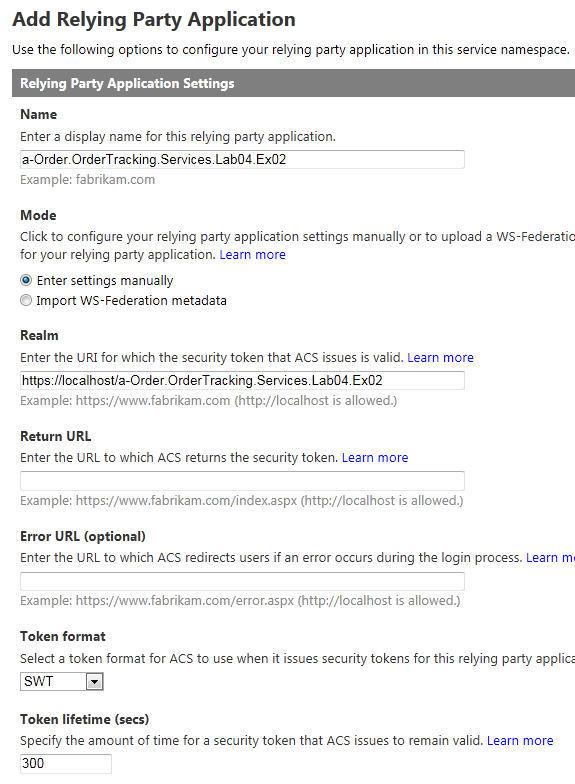
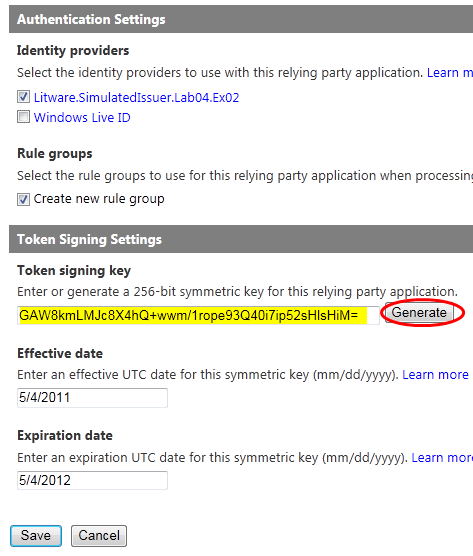
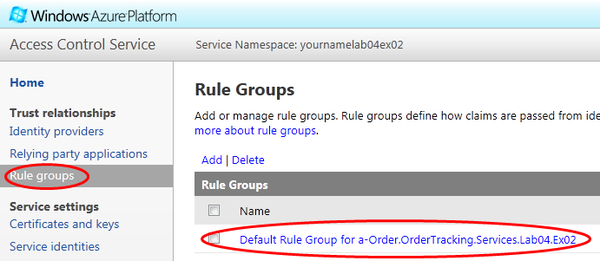
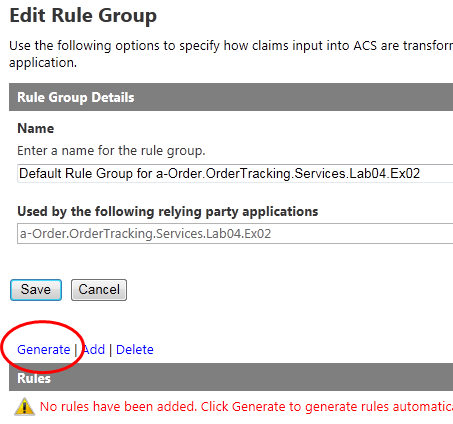
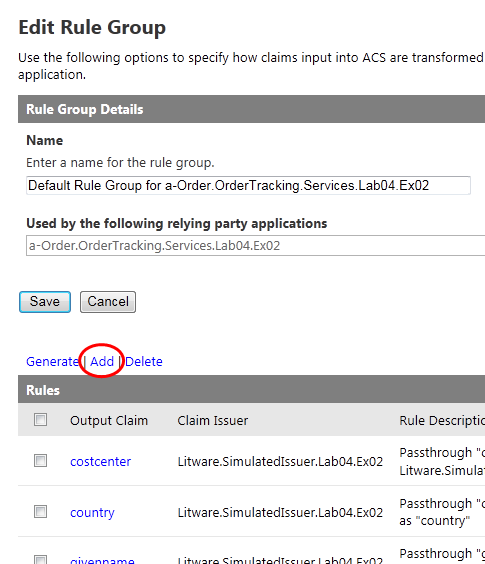
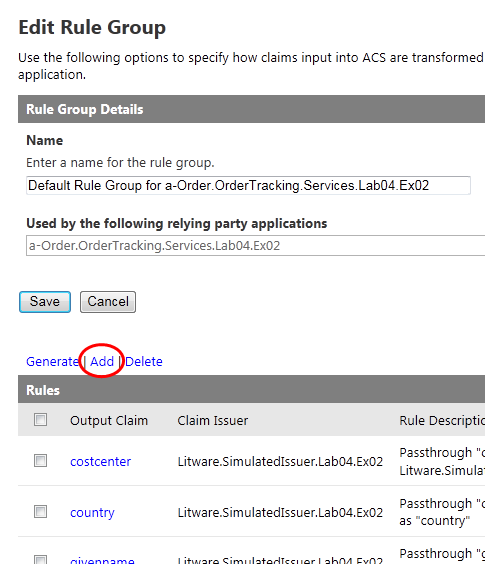
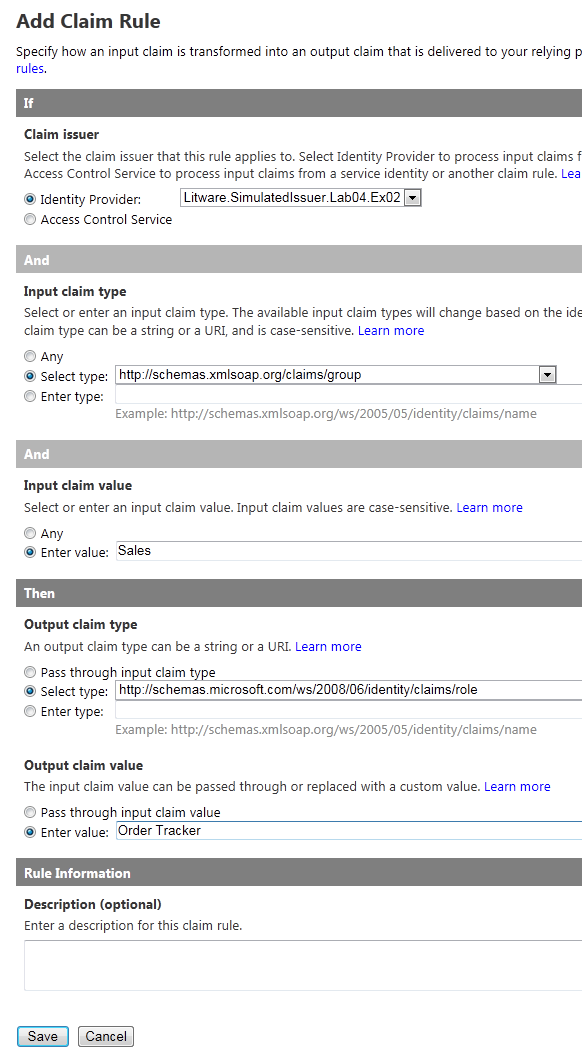
* + [Task 1](#_Task_1:_Add): Add the Litware simulated issuer and configure ACS.
  + [Task 2](#_Task_2:_Add_1): Add a WIF extension to the service.
  + [Task 3](#_Task_3:_Change_1): Add the authorization manger.
  + [Task 4](#_Task_4:_Modify_1): Modify the client to send an SWT token to the service.
  + [Task 5](#_Task_5:_Verify_1): Verify the solution

You should be able to complete this exercise in approximately 30 minutes.

## Task 1: Add the Litware Simulated Issuer and Configure ACS

* 1. In this task, you will first add the Litware identity provider project to the solution before creating an ACS namespace to use in this Lab. You will configure the ACS namespace with details of the a-Order relying party and the Litware identity provider.
  2. The Litware identity provider is a simulated issuer. In a 'real' scenario, it would already exist in the Litware organization.

To add the Litware simulated issuer and configure ACS

* 1. In Windows Explorer, copy the **Litware.SimulatedIssuer.Lab04.Ex02** folder **(including all of its contents)** from the **Lab04-ActiveClient\Source\Ex02\Assets** folder to the **Lab04-ActiveClient\Source\Ex02\Begin** folder.
     1. The **Litware.SimulatedIssuer.Lab04.Ex02** folder contains a sample STS that is configured to act as an identity provider — this project was created from the WIF **WCF Security Token Service** project template.
  2. Start Visual Studio as an administrator and open the solution named **Lab04.Ex02.sln** from the **Lab04-ActiveClient\Source\Ex02\Begin** folder.
  3. **In Visual Studio Solution Explorer, right-click on the top-level Lab04.Ex02 solution item, point to Add, and click Existing Project. Navigate to the folder   
     Lab04-ActiveClient\Source\Ex02\Begin\Litware.SimulatedIssuer.Lab04.Ex02, select the file named Litware.SimulatedIssuer.Lab04.Ex02.csproj, and click Open. The Litware.SimulatedIssuer.Lab04.Ex02** project now appears in Solution Explorer.
     1. 
  4. Sign-in to your Windows Azure account at [**http://windows.azure.com**](http://windows.azure.com). In the portal, click **Service Bus, Access Control & Caching**, then click **Access Control**.
     1. 
  5. Click **New Namespace**, to create an ACS namespace to use in this Lab. You will need to select a unique name for your namespace, a suitable **Country/Region**, and one of your subscriptions. Then click on **Create Namespace** and wait until the namespace has been activated.
     1. 
  6. Select your new namespace and click on **Access Control Service** to visit the ACS portal.
     1. 
  7. In the ACS portal, select **Identity providers** and click **Add**.
     1. 
  8. On the **Add Identity Provider** page, ensure that **WS-Federation identity provider** is selected and click **Next**.
  9. On the **Add WS-Federation Identity Provider** page, enter **Litware.SimulatedIssuer.Lab04.Ex02** in the **Display name** field. Then click the browse button and navigate to the **Lab04-ActiveClient\Source\Ex02\Assets\Litware.SimulatedIssuer.Lab04.Ex02\FederationMetadata\2007-06** folder, then select the **FederationMetadata.xml** file and click **Open**. In the **Login link text** field, enter **Login using the Litware IP**. Then click **Save**.
     1. 
  10. In the ACS portal, select **Relying party applications** and click **Add**.
      1. 
  11. On the **Add Relying Party Application** page, in the **Name** field enter **a-Order.OrderTracking.Services.Lab04.Ex02**, select **Enter settings manually**, in the **Realm** field enter **https://localhost/a-Order.OrderTracking.Services.Lab04.Ex02**, select **SWT** as the **Token format**, in the **Token lifetime** field enter **300**, make sure that only **Litware.SimulatedIssuer.Lab04.Ex02** is selected as an **Identity provider**, click **Generate** to create a new token signing key, and then click **Save**.
      1. You should make a note of the **Token signing key** because you will need it later.
      2. 
      3. 
  12. In the ACS portal, select **Rule groups** and click on **Default Rule Group for a-Order.OrderTracking.Services.Lab04.Ex02**.
      1. 
  13. On the **Edit Rule Group** page, click **Generate**. Then, on the **Generate Rules** page, ensure that only **Litware.SimulatedIssuer.Lab04.Ex02** is selected and then click **Generate**. This will generate a default set of pass-through rules.
      1. 
  14. On the **Edit Rule Group** page, click **Add**. You will add an additional rule to set the value of the organization claim.
      1. 
  15. On the **Add Claim Rule** page, select **Litware.SimulatedIssuer.Lab04.Ex02** as the **Claim issuer**. Select **Any** as the **Input claim type** and select **Any** as the **Input claim value**. Select **Enter type** in the **Output claim type** section and enter **http://schemas.adatum.com/claims/2009/08/organization**. Select **Enter value** in the **Output claim value** section and enter **Litware**. Then click **Save**.
      1. 
  16. On the **Edit Rule Group** page, click **Add**. You will add an additional rule to map the value Sales for the group claim to the value Order Tracker for the role claim.
      1. 
  17. On the Add Claim Rule page, select Litware.SimulatedIssuer.Lab04.Ex02 as the Claim issuer. Select http://schemas.xmlsoap.org/claims/group as the Input claim type and enter Sales as the Input claim value. Select **http://schemas.microsoft.com/ws/2008/06/identity/claims/role** in the Output claim type section. Select Enter value in the Output claim value section and enter Order Tracker. Then click Save.
      1. 
  18. Click the **Save** button above the list to update the rule group with the new rules.
  19. You have now completed this task. You have added the Litware simulated issuer to the solution and configured the ACS namespace that Adatum will use as a federation provider.

## Task 2: Add a WIF extension to the service

* 1. In this task, you will add a WIF extension to the service to handle the SWT token provided by the client. The service will read the claims in the user token and add the claims to the **IClaimsPrincipal** object in the service.

To add a WIF extension to the service

* 1. Continue with the **Lab04.Ex02** solution you used in the previous task.
  2. In Visual Studio, in the **a-Order.OrderTracking.Services.Lab04.Ex02** project, right-click on the **References** folder, and then click on **Add Reference**.
  3. In the **Add Reference** dialog, select the **.NET** tab, and locate the **Microsoft.IdentityModel** component. Clicking on **Component Name** will sort the components alphabetically. Select **Microsoft.IdentityModel**, and click **OK**. This adds a reference to the WIF assembly to the project.
  4. In Visual Studio, in the **a-Order.OrderTracking.Services.Lab04.Ex02** project, right-click on the **References** folder, and then click on **Add Reference**.
  5. In the **Add Reference** dialog, select the **Browse** tab, and navigate to the **Lab04-ActiveClient\Source\Shared Code\Libs** folder. Select **Microsoft.Samples.DPE.OAuth.dll**, and click **OK**. This adds a reference to the WIF extension assembly that adds support for the OAuth protocol and SWT tokens to the project.
  6. In Visual Studio, open the file **Web.config** from the **a-Order.OrderTracking.Services.Lab04.Ex02** project into the code editor window.
  7. As shown in the following snippet, uncomment the **configSections** section.
     1. XML
     2. **<configSections>**
     3. <section name="microsoft.identityModel" …
     4. **</configSections>**
  8. As shown in the following snippet, add the **ProtectedResourceModule**, **WSFederationAuthenticationModule**, and the **SessionAuthenticationModule** modules to the Web.config file.
     1. **Important**: The value of the **type** attributes that are shown split over multiple lines, must be entered on single lines in the Web.config file.
     2. XML
     3. <modules runAllManagedModulesForAllRequests="true">
     4. <add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule, System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
     5. **<add name="ProtectedResourceModule" type="Microsoft.Samples.DPE.OAuth.ProtectedResource.ProtectedResourceModule, Microsoft.Samples.DPE.OAuth, Version=1.0.0.0, Culture=neutral" />**
     6. **<add name="WSFederationAuthenticationModule" type="Microsoft.IdentityModel.Web.WSFederationAuthenticationModule, Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" preCondition="managedHandler" />**
     7. **<add name="SessionAuthenticationModule" type="Microsoft.IdentityModel.Web.SessionAuthenticationModule, Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" preCondition="managedHandler" />**
     8. </modules>
     9. By adding these modules, the application can use the WIF infrastructure to handle the OAuth protocol and SWT tokens.
  9. As shown in the following highlighted code, add a **microsoft.identityModel** section to the Web.config file to configure WIF in the a-Order application. This code configures the Adatum ACS instance as a trusted issuer for the a-Order application. *Notice* that you must add the Token signing key from ACS as the value of the **serviceKey** attribute and modify the **issuerIdentifier** attribute with your ACS Service Namespace name (see Task 1, Step 5).
     1. **Important**: The value of the **type** attributes that are shown split over multiple lines, must be entered on single lines in the Web.config file.
     2. XML
     3. </system.serviceModel>
     4. **<microsoft.identityModel>**
     5. **<service name="OAuth">**
     6. **<audienceUris>**
     7. **<add value="https://localhost/a-Order.OrderTracking.Services.Lab04.Ex02" />**
     8. **</audienceUris>**
     9. **<securityTokenHandlers>**
     10. **<add type="Microsoft.Samples.DPE.OAuth.Tokens.SimpleWebTokenHandler, Microsoft.Samples.DPE.OAuth" />**
     11. **</securityTokenHandlers>**
     12. **<issuerTokenResolver type=**
     13. **"Microsoft.Samples.DPE.OAuth.ProtectedResource.ConfigurationBasedIssuerTokenResolver, Microsoft.Samples.DPE.OAuth">**
     14. **<serviceKeys>**
     15. **<add serviceName=**
     16. **"https://localhost/a-Order.OrderTracking.Services.Lab04.Ex02"**
     17. **serviceKey="{token signing key from Task 1, Step 11}" />**
     18. **</serviceKeys>**
     19. **</issuerTokenResolver>**
     20. **<issuerNameRegistry type=**
     21. **"Microsoft.Samples.DPE.OAuth.ProtectedResource.SimpleWebTokenTrustedIssuersRegistry, Microsoft.Samples.DPE.OAuth">**
     22. **<trustedIssuers>**
     23. **<add issuerIdentifier=**
     24. **"https://{Your ACS Service Namespace name}.accesscontrol.windows.net/"**
     25. **name="ACS" />**
     26. **</trustedIssuers>**
     27. **</issuerNameRegistry>**
     28. **</service>**
     29. **</microsoft.identityModel>**

</configuration>

* 1. Close all of the files open in the Visual Studio editor, making sure to save your changes, but keep the solution open so that you are ready to use it in the next task.
  2. You have now completed this task and you should know how to modify the Web.config file to use WIF to define a trusted issuer for the relying party application.

## Task 3: Add the Authorization Manger

* 1. In this task, you will add a claims-based authorization manager to the a-Order application.

To add the authorization manger

* 1. Continue with the **Lab04.Ex02** solution you used in the previous task.
  2. In Solution Explorer, right-click the **a-Order.OrderTracking.Services.Lab04.Ex02** project, click **Add**, and then click **Class**. In the **Name** field, type **SimpleClaimsAuthorizationManager.cs**, and then click **Add**.
  3. In the code editor, replace the contents of the file with the following code. This defines a **CheckAccess** method to check if the caller is in the **OrderTracking** role. Membership of this role is specified in the claims sent to the Web service by the client application.
     1. C#
     2. namespace AOrder.OrderTracking.Services
     3. {
     4. using Microsoft.IdentityModel.Claims;
     5. using Samples.Web.ClaimsUtilities;
     6. public class SimpleClaimsAuthorizationManager : ClaimsAuthorizationManager
     7. {
     8. public override bool CheckAccess(AuthorizationContext context)
     9. {
     10. return context.Principal.IsInRole(Adatum.Roles.OrderTracker);
     11. }
     12. }
     13. }
  4. In Visual Studio, open the file **OrderTrackingService.cs** file from the **a-Order.OrderTracking.Services.Lab04.Ex02** project into the code editor window. Using the highlighted code in the following sample, replace the existing code that hardcoded the name of the organization with code that retrieves the organization name from a claim.
     1. C#
     2. namespace AOrder.OrderTracking.Services
     3. {
     4. using System.Linq;
     5. using System.ServiceModel;
     6. using System.ServiceModel.Activation;
     7. **using System.Web;**
     8. using AOrder.OrderTracking.Contracts;
     9. using AOrder.OrderTracking.Contracts.Data;
     10. using Samples.Web.ClaimsUtilities;
     11. [AspNetCompatibilityRequirements(RequirementsMode =
     12. AspNetCompatibilityRequirementsMode.Allowed)]
     13. [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
     14. public class OrderTrackingService : IOrderTrackingService
     15. {
     16. public Order[] GetOrdersFromMyOrganization()
     17. {
     18. **string organization = ClaimHelper.GetClaimsFromPrincipal(**
     19. **HttpContext.Current.User, Adatum.ClaimTypes.Organization).Value;**
     20. var repository = new OrderRepository();
     21. return repository.GetOrdersByCompanyName(organization).ToArray();
     22. }
     23. }
     24. }
  5. In Visual Studio, open the file **Web.config** from the **a-Order.OrderTracking.Services.Lab04.Ex02** project into the code editor window.
  6. Using the highlighted code in the following snippet, add the **SimpleClaimsAuthorizationManager** class to the WIF configuration.
     1. XML
     2. <microsoft.identityModel>
     3. <service name="OAuth">
     4. …
     5. **<claimsAuthorizationManager**
     6. **type="AOrder.OrderTracking.Services.SimpleClaimsAuthorizationManager,**
     7. **AOrder.OrderTracking.Services" />**
     8. </service>
     9. </microsoft.identityModel>
     10. **Important**: The value of the **type** attribute that is shown split over multiple lines, must be entered on a single line in the Web.config file.
  7. In Solution Explorer, right-click the **Lab04.Ex02** solution item and click **Rebuild solution**. Correct any errors.
  8. Close all of the files open in the Visual Studio editor, making sure to save your changes, but keep the solution open so that you are ready to use it in the next task.
  9. You have now completed this task. The changes that you made illustrate the changes that you need to make to a REST-based Web service so that it can use a WIF extension to handle SWT tokens and the OAuth protocol. You have also seen how you use claims to implement authorization rules in the service.

## Task 4: Modify the Client to Send an SWT Token to the Service

* 1. In this task, you will modify the desktop client to work with your claims-enabled REST-based Web service. You will modify the client to attach an SWT token to the outgoing request to the a-Order Web service. The client application obtains the SWT token from the Adatum ACS namespace, and to obtain an SWT token from ACS the client must send it a SAML token. The client obtains the SAML token from the Litware simulated issuer.
  2. To handle this process you will add a custom header message inspector class to the client. This class will inspect every outgoing request to the a-Order REST-based service. The custom header message inspector class will first obtain a SAML token by authenticating with the Litware issuer; it will send the SAML token to ACS and obtain an SWT token that it will attach to the outgoing request.

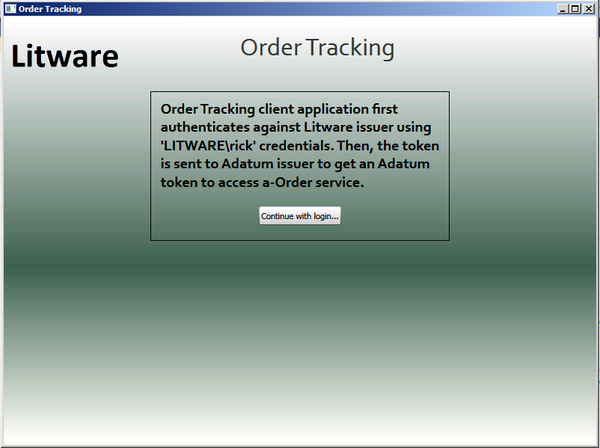
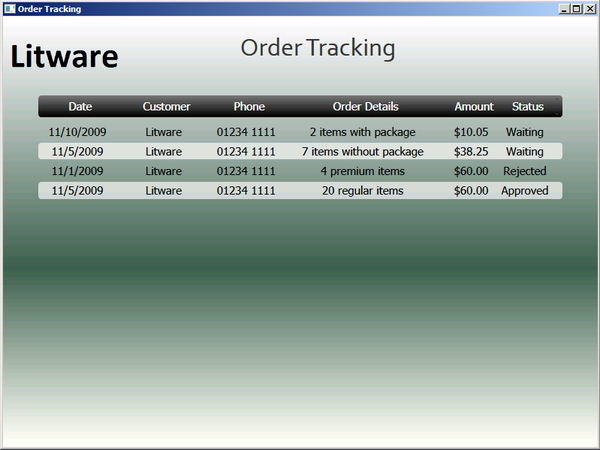
To modify the client to send an SWT token to the service

* 1. Continue with the **Lab04.Ex02** solution you used in the previous task.
  2. In Visual Studio, in the **a-Order.OrderTracking.Client.Lab04.Ex02** project, right-click on the **References** folder, and then click on **Add Reference**.
  3. In the **Add Reference** dialog, select the **.NET** tab, and locate the **Microsoft.IdentityModel** component. Clicking on **Component Name** will sort the components alphabetically. Select **Microsoft.IdentityModel**, and click **OK**. This adds a reference to the WIF assembly to the project.
  4. In Solution Explorer, right-click the **a-Order.OrderTracking.Client.Lab04.Ex02** project, click **Add**, and then click **Class**. In the **Name** field, type **CustomHeaderMessageInspector.cs**, and then click **Add**.
  5. In the code editor, replace the contents of the file with the following code. This declares a several class level variables, modifies the class to implement the **IClientMessageInspector** interface, and adds a constructor.
     1. C#
     2. namespace AOrder.OrderTracking.Client
     3. {
     4. using System;
     5. using System.Collections.Generic;
     6. using System.Collections.Specialized;
     7. using System.IdentityModel.Tokens;
     8. using System.Net;
     9. using System.ServiceModel;
     10. using System.ServiceModel.Channels;
     11. using System.ServiceModel.Description;
     12. using System.ServiceModel.Dispatcher;
     13. using System.ServiceModel.Security;
     14. using System.Text;
     15. using System.Web.Script.Serialization;
     16. using Microsoft.IdentityModel.Protocols.WSTrust;
     17. using Microsoft.IdentityModel.Protocols.WSTrust.Bindings;
     18. using Microsoft.IdentityModel.SecurityTokenService;
     19. public class CustomHeaderMessageInspector : IClientMessageInspector
     20. {
     21. private const string ServiceEndpointFormat =
     22. "https://{0}.accesscontrol.windows.net";
     23. private readonly string acsRelyingParty;
     24. private readonly ClientCredentials clientCredentials;
     25. private readonly string serviceEndpoint;
     26. private readonly string stsEndpoint;
     27. private NameValueCollection oauthToken;
     28. private DateTime oauthTokenValidUntil = DateTime.MinValue;
     29. private GenericXmlSecurityToken samlToken;
     30. public CustomHeaderMessageInspector(ClientCredentials clientCredentials,
     31. string acsNamespace, string acsRelyingParty, string stsEndpoint)
     32. {
     33. this.clientCredentials = clientCredentials;
     34. this.acsRelyingParty = acsRelyingParty;
     35. this.stsEndpoint = stsEndpoint;
     36. this.serviceEndpoint = string.Format(ServiceEndpointFormat,
     37. acsNamespace);
     38. }
     39. }
     40. }
  6. Add the **GetSamlToken** method shown in the following code sample to the **CustomHeaderMessageInspector** class. This method will send the user's credentials to the Litware identity provider and receive a SAML token in return.
     1. C#
     2. private static SecurityToken GetSamlToken(string realm, string stsEndpoint,
     3. ClientCredentials clientCredentials)
     4. {
     5. using (var factory = new WSTrustChannelFactory(
     6. new UserNameWSTrustBinding(SecurityMode.TransportWithMessageCredential),
     7. new EndpointAddress(new Uri(stsEndpoint))))
     8. {
     9. factory.Credentials.UserName.UserName = clientCredentials.UserName.UserName;
     10. factory.Credentials.UserName.Password = clientCredentials.UserName.Password;
     11. factory.TrustVersion = TrustVersion.WSTrust13;
     12. WSTrustChannel channel = null;
     13. try
     14. {
     15. var rst = new RequestSecurityToken
     16. {
     17. RequestType = WSTrust13Constants.RequestTypes.Issue,
     18. AppliesTo = new EndpointAddress(realm),
     19. KeyType = KeyTypes.Bearer,
     20. };
     21. channel = (WSTrustChannel)factory.CreateChannel();
     22. return channel.Issue(rst);
     23. }
     24. finally
     25. {
     26. if (channel != null)
     27. {
     28. channel.Abort();
     29. }
     30. factory.Abort();
     31. }
     32. }
     33. }
  7. Add the **GetOAuthToken** method shown in the following code sample to the **CustomHeaderMessageInspector** class. This method will send the SAML token from the Litware identity provider to ACS and receive a SWT token in return.
     1. C#
     2. private static NameValueCollection GetOAuthToken(string xmlSamlToken, string
     3. serviceEndpoint, string acsRelyingParty)
     4. {
     5. var values = new NameValueCollection
     6. {
     7. { "grant\_type", "urn:oasis:names:tc:SAML:2.0:assertion" },
     8. { "assertion", xmlSamlToken },
     9. { "scope", acsRelyingParty }
     10. };
     11. var client = new WebClient { BaseAddress = serviceEndpoint };
     12. byte[] acsTokenResponse = client.UploadValues("v2/OAuth2-13", "POST", values);
     13. string acsToken = Encoding.UTF8.GetString(acsTokenResponse);
     14. var tokens = new NameValueCollection();
     15. var json = new JavaScriptSerializer();
     16. var parsed = json.DeserializeObject(acsToken) as Dictionary<string, object>;
     17. foreach (var item in parsed)
     18. {
     19. tokens.Add(item.Key, item.Value.ToString());
     20. }
     21. return tokens;
     22. }
  8. Add implementations of the **BeforeSendRequest** and **AfterReceiveReply** methods to the **CustomHeaderMessageInspector** class. The **BeforeSendRequest** method orchestrates the process of obtaining the SAML token from the Litware issuer, the SWT token from ACS, and adding the SWT token to the outgoing authorization header.
     1. C#
     2. public void AfterReceiveReply(ref Message reply, object correlationState)
     3. {
     4. }
     5. public object BeforeSendRequest(ref Message request, IClientChannel channel)
     6. {
     7. // Making sure we have a HttpRequestMessageProperty
     8. HttpRequestMessageProperty httpRequestMessageProperty;
     9. if (request.Properties.ContainsKey(HttpRequestMessageProperty.Name))
     10. {
     11. httpRequestMessageProperty = request.Properties[HttpRequestMessageProperty.Name]
     12. as HttpRequestMessageProperty;
     13. if (httpRequestMessageProperty == null)
     14. {
     15. httpRequestMessageProperty = new HttpRequestMessageProperty();
     16. request.Properties.Add(HttpRequestMessageProperty.Name,
     17. httpRequestMessageProperty);
     18. }
     19. }
     20. else
     21. {
     22. httpRequestMessageProperty = new HttpRequestMessageProperty();
     23. request.Properties.Add(HttpRequestMessageProperty.Name,
     24. httpRequestMessageProperty);
     25. }
     26. // Get SWT token from ACS if no token has been requested before or the saved one
     27. // has expired
     28. if (oauthToken == null || oauthTokenValidUntil == DateTime.MinValue ||
     29. DateTime.UtcNow > oauthTokenValidUntil)
     30. {
     31. // Get SAML token from Litware
     32. if (samlToken == null)
     33. {
     34. samlToken = GetSamlToken(this.serviceEndpoint, this.stsEndpoint,
     35. this.clientCredentials) as GenericXmlSecurityToken;
     36. }
     37. int timesTriedCallingAcs;
     38. const int maxTimesToTryCallingAcs = 2;
     39. for (timesTriedCallingAcs = 1; timesTriedCallingAcs <= maxTimesToTryCallingAcs;
     40. timesTriedCallingAcs++)
     41. {
     42. try
     43. {
     44. // Get OAuth from ACS authenticating with the SAML token from Litware
     45. oauthToken = GetOAuthToken(samlToken.TokenXml.OuterXml, this.serviceEndpoint,
     46. this.acsRelyingParty);
     47. break;
     48. }
     49. catch (WebException)
     50. {
     51. if (timesTriedCallingAcs == 1)
     52. {
     53. // Refresh the SAML token from Litware because it may be expired before
     54. // calling ACS again
     55. samlToken = GetSamlToken(this.serviceEndpoint, this.stsEndpoint,
     56. this.clientCredentials) as GenericXmlSecurityToken;
     57. }
     58. }
     59. }
     60. if (timesTriedCallingAcs > maxTimesToTryCallingAcs)
     61. {
     62. throw new ApplicationException("Unable to get a token from ACS with the token issued by Litware. This may be caused by an incorrect date and time in this computer.");
     63. }
     64. var tokenExpiresIn = double.Parse(oauthToken["expires\_in"]);
     65. oauthTokenValidUntil = DateTime.UtcNow + TimeSpan.FromSeconds(tokenExpiresIn);
     66. }
     67. // Add the token to the request's authorization header
     68. var oauthAuthorizationHeader = string.Format("Bearer {0}",
     69. oauthToken["access\_token"]);
     70. httpRequestMessageProperty.Headers.Add(HttpRequestHeader.Authorization,
     71. oauthAuthorizationHeader);
     72. return null;
     73. }
     74. This code also caches the SWT token from ACS so the application does not have to visit the Litware issuer or ACS for every request.
  9. In Solution Explorer, right-click the **a-Order.OrderTracking.Client.Lab04.Ex02** project, click **Add**, and then click **Class**. In the **Name** field, type **CustomHeaderBehavior.cs**, and then click **Add**.
  10. In the code editor, replace the contents of the file with the following code. This declares a several class level variables, modifies the class to implement the **IEndpointBehavior** interface, adds a constructor, and adds a **CustomHeaderMessageInspector** instance to the client runtime.
      1. C#
      2. namespace AOrder.OrderTracking.Client
      3. {
      4. using System.ServiceModel.Channels;
      5. using System.ServiceModel.Description;
      6. using System.ServiceModel.Dispatcher;
      7. public class CustomHeaderBehavior : IEndpointBehavior
      8. {
      9. private readonly string acsNamespace;
      10. private readonly string acsRelyingParty;
      11. private readonly ClientCredentials clientCredentials;
      12. private readonly string stsEndpoint;
      13. public CustomHeaderBehavior(ClientCredentials clientCredentials, string
      14. acsNamespace, string acsRelyingParty, string stsEndpoint)
      15. {
      16. this.clientCredentials = clientCredentials;
      17. this.acsNamespace = acsNamespace;
      18. this.acsRelyingParty = acsRelyingParty;
      19. this.stsEndpoint = stsEndpoint;
      20. }
      21. public void AddBindingParameters(ServiceEndpoint endpoint,
      22. BindingParameterCollection bindingParameters)
      23. {
      24. }
      25. public void ApplyClientBehavior(ServiceEndpoint endpoint,
      26. ClientRuntime clientRuntime)
      27. {
      28. var customHeaderMessageInspector =
      29. new CustomHeaderMessageInspector(this.clientCredentials, this.acsNamespace,
      30. this.acsRelyingParty, this.stsEndpoint);
      31. clientRuntime.MessageInspectors.Add(customHeaderMessageInspector);
      32. }
      33. public void ApplyDispatchBehavior(ServiceEndpoint endpoint,
      34. EndpointDispatcher endpointDispatcher)
      35. {
      36. }
      37. public void Validate(ServiceEndpoint endpoint)
      38. {
      39. }
      40. }
      41. }
  11. In Visual Studio, open the file **OrderTrackingServiceClient.cs** file from the **a-Order.OrderTracking.Client.Lab04.Ex02** project into the code editor window. Using the highlighted code in the following sample, modify the class to add a new **CustomHeaderBehavior** instance to the channel factory.
      1. C#
      2. public class OrderTrackingServiceClient : ClientBase<IOrderTrackingService>,
      3. IOrderTrackingService
      4. {
      5. **private readonly string acsNamespace;**
      6. **private readonly string acsRelyingParty;**
      7. **private readonly string stsEndpoint;**
      8. **public OrderTrackingServiceClient(string acsNamespace, string acsRelyingParty,**
      9. **string stsEndpoint)**
      10. **{**
      11. **this.acsNamespace = acsNamespace;**
      12. **this.acsRelyingParty = acsRelyingParty;**
      13. **this.stsEndpoint = stsEndpoint;**
      14. **}**
      15. public Order[] GetOrdersFromMyOrganization()
      16. {
      17. return this.Channel.GetOrdersFromMyOrganization();
      18. }
      19. **protected override IOrderTrackingService CreateChannel()**
      20. **{**
      21. **this.ChannelFactory.Endpoint.Behaviors.Add(**
      22. **new CustomHeaderBehavior(this.ClientCredentials, this.acsNamespace,**
      23. **this.acsRelyingParty, this.stsEndpoint));**
      24. **return base.CreateChannel();**
      25. **}**
      26. }
  12. In Visual Studio, open the file **OrderTrackingController.cs** file from the **a-Order.OrderTracking.Client.Lab04.Ex02** project into the code editor window. Using the highlighted code in the following sample, modify the **ShowOrders** method to instantiate an **OrderTrackingServiceClient** object with the addresses of the Litware IP and your ACS namespace (see Task 1, Step 5).
      1. C#
      2. private void ShowOrders()
      3. {
      4. **var stsEndpoint =**
      5. **"https://localhost/Litware.SimulatedIssuer.Lab04.Ex02/Issuer.svc";**
      6. **var acsNamespace = "{your-ACS-namespace-name}";**
      7. **var acsRelyingParty =**
      8. **"https://localhost/a-Order.OrderTracking.Services.Lab04.Ex02";**
      9. **var client = new OrderTrackingServiceClient(acsNamespace, acsRelyingParty,**
      10. **stsEndpoint);**
      11. client.ClientCredentials.UserName.UserName = "LITWARE\\rick";
      12. client.ClientCredentials.UserName.Password = "thisPasswordIsNotChecked";
      13. …
      14. }
  13. In Solution Explorer, right-click the **Lab04.Ex02** solution item and click **Rebuild solution**. Correct any errors.
  14. Close all of the files open in the Visual Studio editor, making sure to save your changes, but keep the solution open so that you are ready to use it in the next task.
  15. You have now completed this task. You have modified the client application to attach an SWT token from ACS to the request sent to the RESTful a-Order Web service. The client application first authenticates with the Litware IP to obtain a SAML token, it then sends the SAML token to ACS to get the SWT token with the claims that the a-Order application expects. For performance reasons, the client application caches the SWT token to reuse for subsequent requests.

## Task 5: Verify the Solution

* 1. In this task, you will verify the solution by running the client application.

To verify the solution

* 1. In Visual Studio, on the **Debug** menu, click **Start Without Debugging**. In the Order Tracking application, click **Continue with login**.
     1. 
  2. The client application authenticates with the Litware issuer, obtains an SWT token from the Adatum ACS namespace, and connects to the Web service. The Web service returns Litware orders for the client application to display.
     1. 
  3. You have now completed this task and this exercise. You have seen how to configure a federated identity environment: users from Litware can access the a-Order application using credentials from their own identity providers. In this scenario, the Web service used a REST style API and ACS provided the federation services.
  4. Federated identity is a powerful feature of claims; users do not need to remember additional sets of credentials to access a partner's application, and it is easy for an organization such as Adatum to configure its federation provider to trust issuers from additional partners without the requirement to modify the relying party application. Furthermore, the relying party application no longer has the responsibility for verifying the user's username and password: this is now handled by an identity provider. Using ACS as a federation provider means that Adatum does not need to deploy its own federation provider. ACS also makes it easy to federate with social identities such as Windows Live IDs, Google IDs, and Yahoo! IDs.

## Running the "End" Solution

1. If you did not complete all of the tasks in this exercise, you can run the provided "end" solution.

To run the end solution

* 1. Start Visual Studio 2010 as an administrator.
  2. Open the solution named **Lab04.Ex02.End** from the folder **Lab04-ActiveClient\Source\Ex02\End**.
  3. Open the **Build** menu and click **Rebuild Solution**.
  4. You will need to configure an ACS service namespace before you can run the end solution. To simplify this you can use the ACS.Setup.Lab04.Ex02.End project to add the Litware identity provider, the a-Order relying party, and the rules to your ACS namespace.
  5. Before you can run the ACS.Setup program you must customize it for your ACS namespace. You will need two pieces of information from ACS: the name of your ACSservice namespace, and the password for your **ManagementClient** credential. You can find the password here:
     1. 
  6. In Visual Studio, in the **ACS.Setup.Lab04.Ex02.End** project, open the **Program.cs** file. Modify the file as shown in the following highlighted code.
     1. C#
     2. private const string AcsPassword = "**{your-acs-password}**";
     3. private const string AcsServiceNamespace = "**{your-namespace-name}**";
     4. ...
     5. private const string TokenKey = "**{token signing key from Task 1, Step 11}**";
     6. This code uses the default management account in ACS named **ManagementClient**. For help locating the correct values to use for the password and namespace, see the section "*Locating Your ACS Password Key and Namespace*" at the start of Lab 3.
  7. In Visual Studio, right-click the **ACS.Setup.Lab04.Ex02.End** project and click **Rebuild**.
  8. In Visual Studio, right-click the **ACS.Setup.Lab04.Ex02.End** project, point to **Debug**, and click **Start new instance**.
  9. You must also update the a-Order.OrderTracking.Client.Lab04.Ex02.End project with the name of your ACS namespace service. In Visual Studio, in the **a-Order.OrderTracking.Client.Lab04.Ex02.End** project, open the **OrderTrackingController.cs** file, and modify the **ShowOrders** method as shown in the following highlighted code:
     1. C#
     2. private void ShowOrders()
     3. {
     4. var stsEndpoint =
     5. "https://localhost/Litware.SimulatedIssuer.Lab04.Ex02.End/Issuer.svc";
     6. var acsNamespace = "**{your-namespace-name}**";
     7. var acsRelyingParty =
     8. "https://localhost/a-Order.OrderTracking.Services.Lab04.Ex02.End";
     9. …
     10. }
  10. In Visual Studio, in the **a-Order.OrderTracking.Services.Lab04.Ex02.End** project, open the **Web.config** file. Scroll down to the **<microsoft.identityModel>** section and modify the **serviceKey** attribute to include the token signing key from ACS that you identified in Task 1, Step 11.
      1. XML
      2. <serviceKeys>
      3. <add serviceName=
      4. "https://localhost/a-Order.OrderTracking.Services.Lab04.Ex02.End"
      5. serviceKey="**{token signing key from Task 1, Step 11}**" />
      6. </serviceKeys>
  11. You must also update the **a-Order.OrderTracking.Services.Lab04.Ex02.End** project with the name of your ACS namespace service. Modify the **trustedIssuers** section as shown in the following highlighted code:
      1. XML
      2. <issuerNameRegistry type="Microsoft.Samples…
      3. <trustedIssuers>
      4. <add issuerIdentifier=
      5. "https://**{your-namespace-name}**.accesscontrol.windows.net/"
      6. name="ACS" />
      7. </trustedIssuers>
      8. </issuerNameRegistry>
  12. Save all of the modified files.
  13. In Solution Explorer, right-click the **Lab04.Ex02.End** solution item and click **Rebuild solution**. Correct any errors. Close all of the files open in the Visual Studio editor, making sure to save your changes.
  14. On the **Debug** menu, click **Start Without Debugging** to run the application.
  15. You can then simulate logging in as Rick from Litware.

# Exercise 3: Federation with ADFS

* 1. In this optional exercise, you will modify the SOAP-based scenario from exercise 1 to replace the mock Adatum federation provider and identity provider projects in the solution and configure ADFS to perform the same roles. This will be a more realistic scenario. The changes to the a-Order application will be just configuration changes so that it points to your ADFS installation instead of the mock issuers in the Visual Studio solution. The remainder of the Lab will show you how to configure ADFS to support the Adatum federated identity scenario.
  2. The Lab Introduction contains instructions for installing and configuring a base ADFS environment. You should ensure that you have read the section, "Information Required for the Lab Exercises," in the Introduction before attempting this Lab Exercise.

This exercise contains the following tasks:

* + [Task 1](#_Task_1:_Modify_1): Modify the a-Order application to trust ADFS
  + [Task 2](#_Task_3:_Add): Add the a-Order application as a Relying Party in ADFS
  + [Task 3](#_Task_4:_Add): Add a-Order application claims rules to ADFS
  + [Task 4](#_Task_5:_Add): Add the Litware issuer as Claim Trust Provider in ADFS
  + [Task 5](#_Task_6:_Add): Add Litware claims rules to ADFS
  + [Task 6](#_Task_6:_Modify): Modify the active client application
  + [Task 7](#_Task_9:_Verify): Verify the solution

You should be able to complete this exercise in approximately 20 minutes.

## Task 1: Modify the a-Order Application to Trust ADFS

* 1. In this task, you will modify the a-Order application to trust your ADFS issuer instead of the mock Adatum federation provider. You will also remove the Adatum federation provider project from the solution as it is no longer required and will be replaced by ADFS.

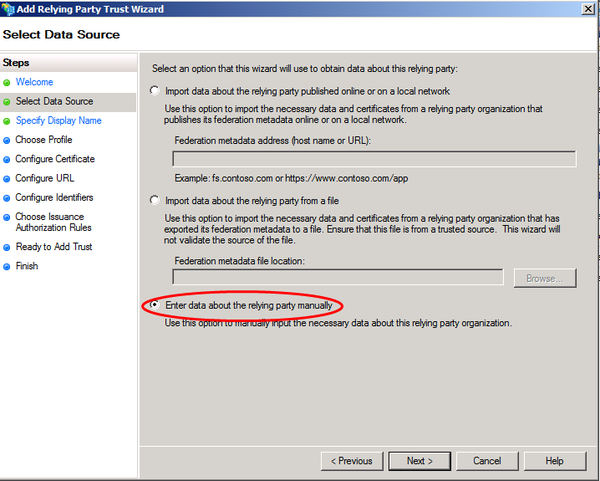
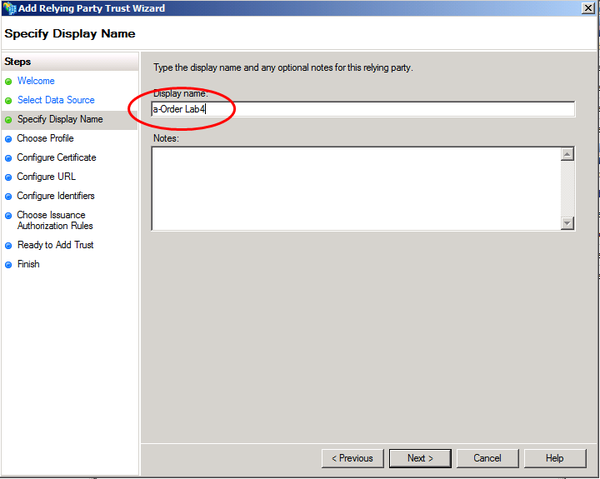
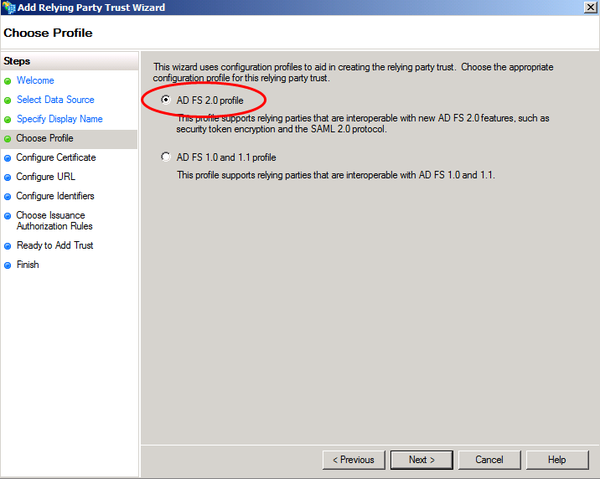
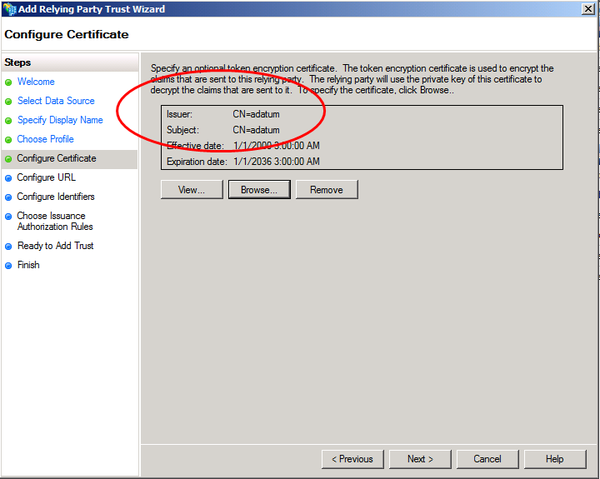
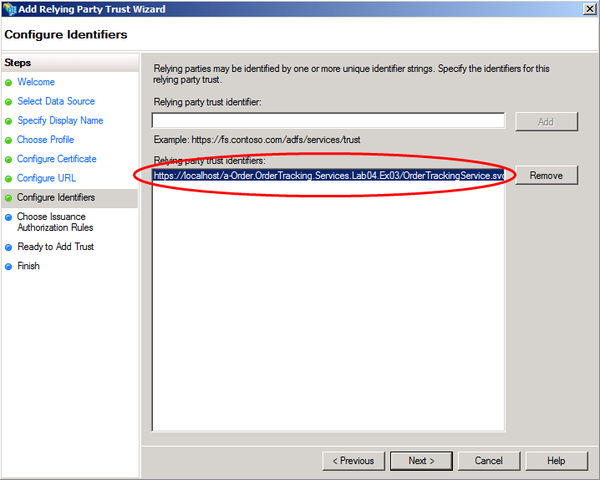
To modify the a-Order application to trust ADFS

* 1. Start Visual Studio as an administrator and open the solution named **Lab04.Ex03.sln** from the **Lab04-ActiveClient\Source\Ex03\Begin** folder.
  2. In Solution Explorer, right-click on the **Adatum.FederationProvider.Lab04.Ex03** project node, and select **Remove**. Then click **OK**.
  3. In Solution Explorer expand the **a-Order.OrderTracking.Services.Lab04.Ex03** project, and double-click on the file **Web.config** to open it in the code editor.
  4. Using the values for the name and thumbprint of your ADFS issuer that you recorded when you read the section "Integrating With Active Directory Federation Services" in the Introduction to these Lab exercises, modify the highlighted values in the **trustedIssuers** section as shown in the following snippet:
     1. XML
     2. <trustedIssuers>
     3. <add thumbprint="**{your-thumbprint}**"
     4. name="**{your-name-identifier}**" />
     5. </trustedIssuers>
  5. Using the host name of your ADFS instance that you recorded when you read the section "Integrating With Active Directory Federation Services" in the Introduction to these Lab exercises, modify the **issuer** element in the **ws2007FederationHttpBinding** element as shown in the following snippet:
     1. XML
     2. <ws2007FederationHttpBinding>
     3. <binding name="WS2007FederationHttpBinding\_IOrderTrackingService">
     4. <security mode="TransportWithMessageCredential">
     5. <message>
     6. **<issuer address=**
     7. **"https://{issuer-url}/adfs/services/trust/13/issuedtokenmixedsymmetricbasic256" binding="ws2007FederationHttpBinding"></issuer>**
     8. </message>
     9. </security>
     10. </binding>
     11. </ws2007FederationHttpBinding>
  6. Close all of the files open in the Visual Studio editor, making sure to save your changes, but keep the solution open so that you are ready to use it in the next task.
  7. You have now completed this task to configure the a-Order application to trust your ADFS instance to issue claims instead of using the mock federation provider.

## Task 2: Add the a-Order Application as a Relying Party in ADFS

* 1. In this task, you will modify the ADFS configuration to define the a-Order application as a relying party.

To add the a-Order application as a Relying Party in ADFS

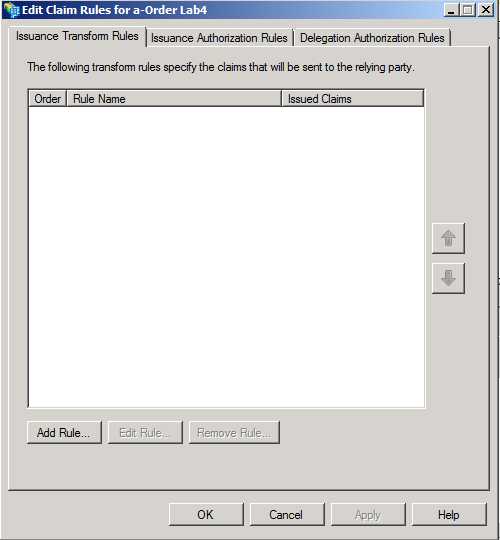
* 1. On the machine where you have installed ADFS, start the **AD FS 2.0 Management** console.
  2. Expand **Trust Relationships**, then right-click on **Relying Party Trusts**, and then select **Add Relying Party Trust**.
  3. In the **Add Relying Party Trust Wizard** click **Start**.
     1. 
  4. On the **Select Data Source** page of the wizard, select **Enter data about the relying party manually** and click **Next**.
     1. 
  5. On the **Specify Display Name** page of the wizard, type **a-Order Lab4** in the **Display name** textbox, then click **Next**.
     1. 
  6. On the **Choose Profile** page of the wizard, ensure that the **AD FS 2.0 profile** option is selected, and then click **Next**.
     1. 
  7. On the **Configure Certificate** page of the wizard, click **Browse** to add the **adatum.cer** certificate. You can find the **Adatum.cer** file in the **Lab04-ActiveClient\Source\Ex03\Assets** folder. If you receive a message, "**The certificate key length is less than 2048 bits. Certificates with key sizes less than 2048 bits might present security risk and are not recommended. Do you want to continue?**" click **Yes** to continue. Then click **Next**.
     1. 
  8. On the **Configure URL** page of the wizard, click **Next**.
  9. On the **Configure Identifiers** page of the wizard, in the **Relying party trust identifier** text box, type **https://localhost/a-Order.OrderTracking.Services.Lab04.Ex03/OrderTrackingService.svc** and click **Add**, then click **Next**.
     1. 
  10. On the **Choose Issuance Authorization Rules** page of the wizard, click **Next**.
  11. On the **Ready to Add Trust** page of the wizard, click **Next**.
  12. On the **Finish** page of the wizard, click **Close**. The **Edit Claim Rules for a-Order Lab4** dialog will open ready for you to complete the next task.

You have now completed this task to define the a-Order application as a relying party in ADFS.

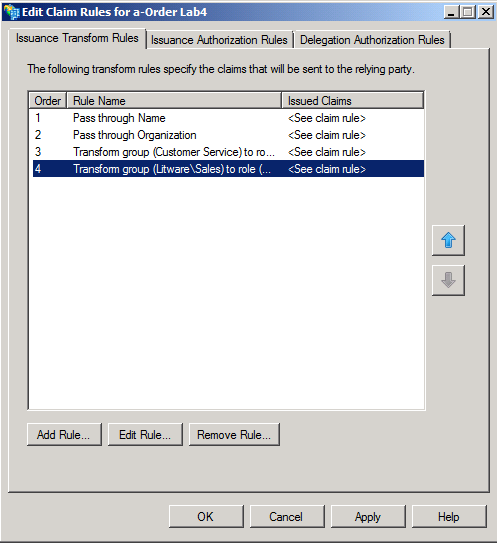
## Task 3: Add a-Order Application Claims Rules to ADFS

* 1. In this task, you will add the claims transformation rules to the **a-Order Lab4** relying party trust in ADFS. These rules will map the claims from the **Active Directory** and **Litware** claims provider trusts to a set of claims that the a-Order application will recognize.

To add a-Order application claims rules to ADFS

* 1. If you do not have the **Edit Claim Rules for a-Order Lab4** dialog open from the previous task, then on the machine where you have installed ADFS, start the AD FS 2.0 Management tool. Expand **Trust Relationships**, then click on **Relying Party Trusts**, then right-click on **a-Order Lab4**, and then select **Edit Claim Rules**. You should be able to see the dialog shown in the following screenshot:
     1. 
  2. Now you can add the claim transformation rules you need to test the scenario. In the **Edit Claim Rules for a-Order Lab4** dialog, click **Add Rule.** Then, in **Add Transform Claim Rule Wizard** dialog, in the **Claim rule template** dropdown list, select **Send Claims Using a Custom Rule**, and then click **Next**.
  3. On the **Configure Rule** page of the wizard, in the **Claim rule name** textbox, type **Pass through Name**. In the **Custom rule** textbox, add the following rule:
     1. c:[Type == "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"] => issue(claim = c);
  4. On the **Configure Rule** page of the wizard, click **Finish**.
  5. Using the information in the following table, repeat steps 2, 3, and 4 to add the remaining rules to ADFS.

|  |  |
| --- | --- |
| Claim rule name | Custom rule |
| Pass through Organization | c:[Type == "http://schemas.adatum.com/claims/2009/08/organization"] => issue (claim = c); |
| Transform group (Customer Service) to role (Order Tracker) | c:[Type == "http://schemas.xmlsoap.org/claims/group", Value =~ "^(?i)Customer\ Service$"]  => issue(Type = "http://schemas.microsoft.com/ws/2008/06/identity/claims/role", Issuer = c.Issuer, OriginalIssuer = c.OriginalIssuer, Value = "Order Tracker", ValueType = c.ValueType); |
| Transform group (Litware\Sales) to role (Order Tracker) | c:[Type == "http://schemas.xmlsoap.org/claims/group", Value =~ "^(?i)Litware\\Sales$"]  => issue(Type = "http://schemas.microsoft.com/ws/2008/06/identity/claims/role", Issuer = c.Issuer, OriginalIssuer = c.OriginalIssuer, Value = "Order Tracker", ValueType = c.ValueType); |

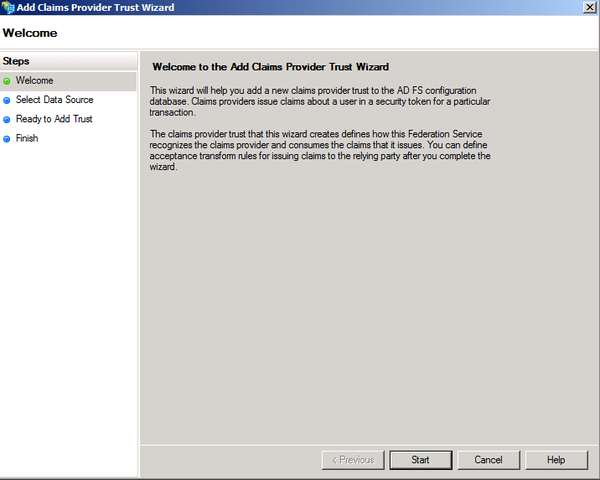
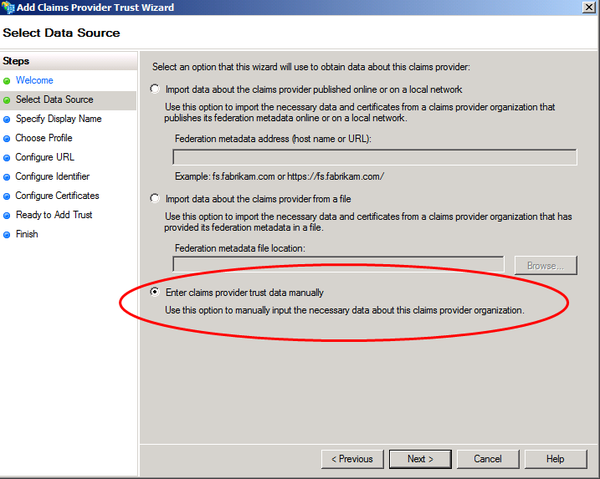
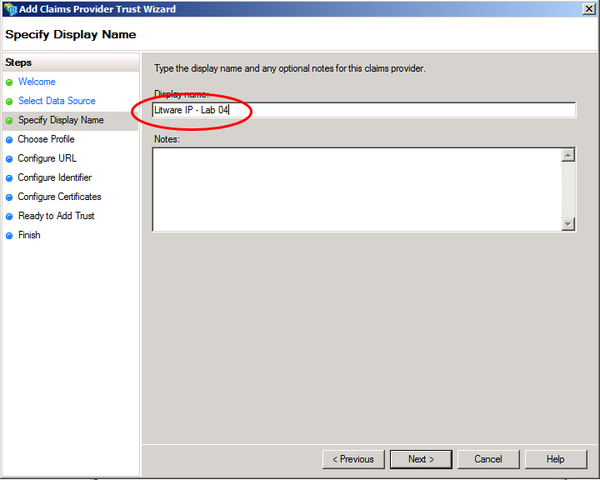
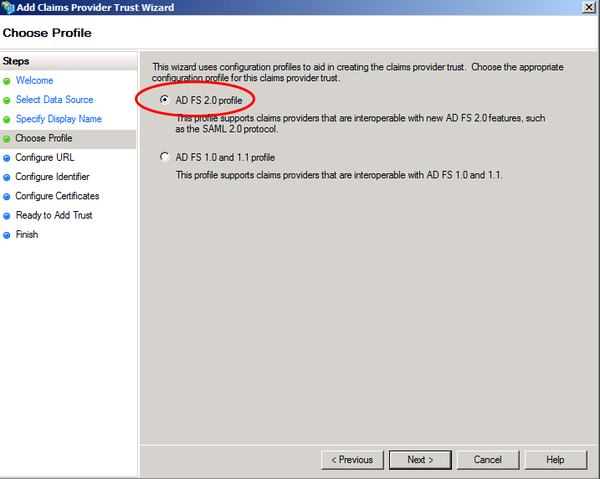
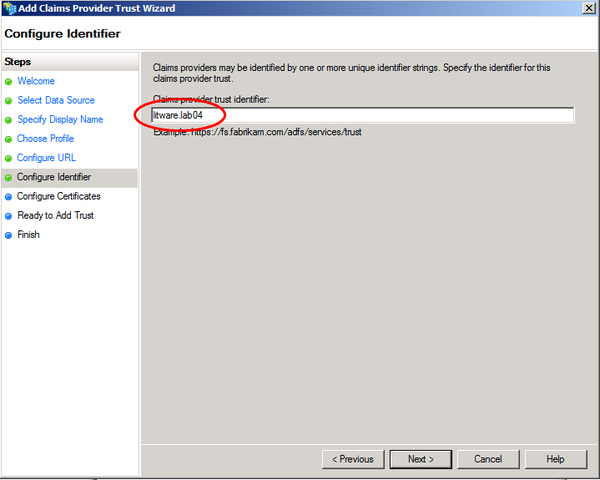
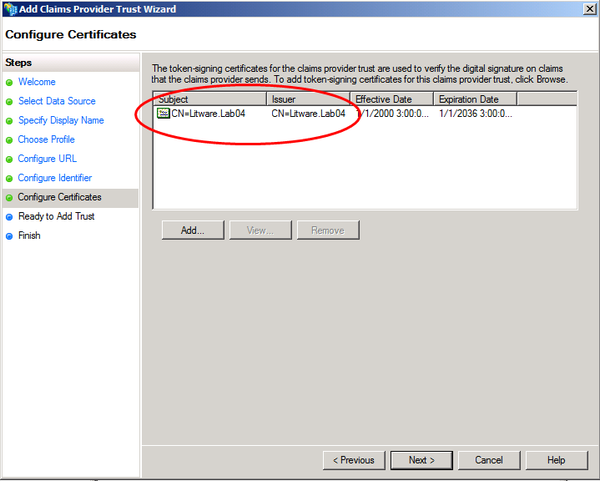
* 1. When you have finished, the **Edit Claim Rules for a-Order Lab4** dialog should look like the following screenshot. Click **OK** to close the dialog.
     1. 

You have now completed this task and added a set of claim transformation rules for the a-Order relying party application in ADFS.

## Task 4: Add the Litware Issuer as Claim Trust Provider in ADFS

* 1. In this task, you will add a claims provider trust for the Litware IP to the ADFS configuration. This will enable users to authenticate with Litware to gain access to the a-Order application.

To add the Litware issuer as Claim Trust Provider in ADFS

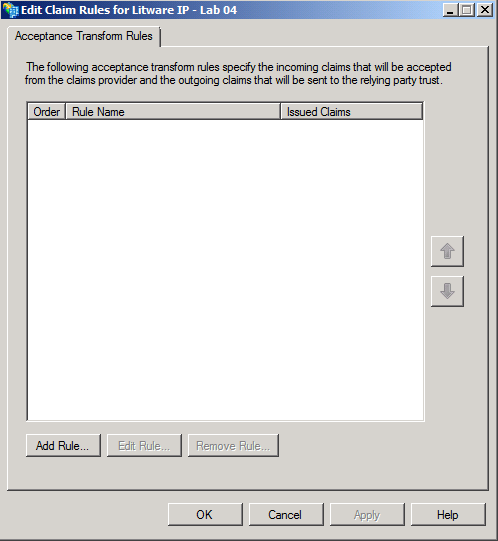
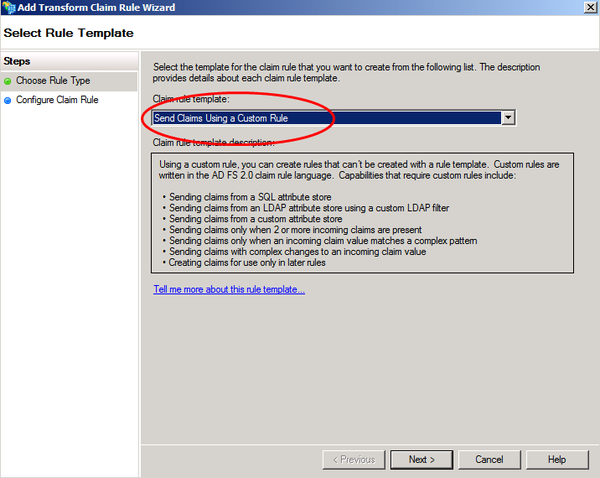
* 1. On the machine where you have installed ADFS, start the AD FS 2.0 Management tool. Expand **Trust Relationships**, then right-click on **Claims Provider Trusts**, and then select **Add Claims Provider Trust**. The **Add Claims Provider Trust Wizard** will launch. Click **Start**.
     1. 
  2. On the **Select Data Source** page of the wizard, click **Enter claims provider trust data manually**, and then click **Next**.
     1. 
  3. On the **Specify Display Name** page, in **Display name** type **Litware IP – Lab 04**. Then click **Next**.
     1. 
  4. On the **Choose Profile** page, select **AD FS 2.0 Profile**, and then click **Next**.
     1. 
  5. On the **Configure URL** page, select the **Enable support for the WS-Federation Passive** protocol check box. Under **Claims provider WS-Federation Passive protocol URL**, type **https://localhost/Litware.SimulatedIssuer.Lab04.Ex03/**, and then click **Next**.
     1. 
  6. On the **Configure Identifier** page, under **Claims provider trust identifier**, type **litware.lab04**, and then click **Next**.
     1. 
  7. On the **Configure Certificates** page, click **Add** to locate the **litware.Lab04.cer** file and add it to the list of certificates, and then click **Next**. If you receive a message, "**The certificate key length is less than 2048 bits. Certificates with key sizes less than 2048 bits might present security risk and are not recommended. Do you want to continue?**" click **Yes** to continue.
     1. You can find the **litware.Lab04.cer** file in the **\Lab04-ActiveClient\Source\Ex03\Assets** folder. If ADFS is installed on a different machine from where the Lab files are located, you will need to copy the file from the machine where the Lab source files are located to the machine where you have ADFS installed.
     2. This is a test certificate — a certificate for use in production should have a key-size of 2048 bits or more.
     3. 
  8. On the **Ready to Add Trust** page, click **Next** to save your claims provider trust information.
  9. On the **Finish** page, click **Close**. This action automatically displays the **Edit Claim Rules** dialog box that you will use in the next task.

You have now completed this task to configure the Litware issuer as a claims provider in ADFS.

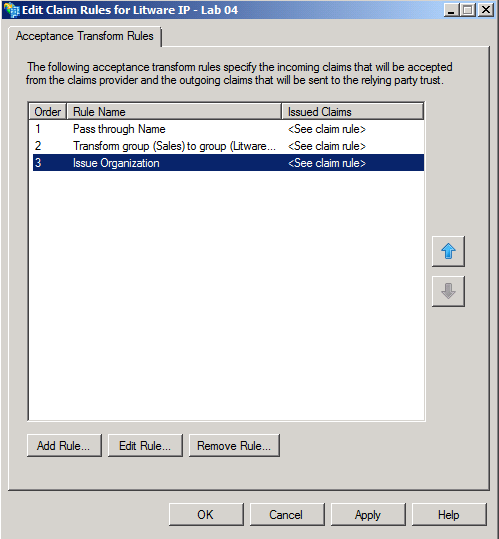
## Task 5: Add Litware Claims Rules to ADFS

* 1. In this task, you will modify the Litware IP claims provider trust in ADFS to accept claims from the Litware issuer. You will use the AD FS 2.0 Management tool to add the rules that you can use to test the a-Order application.

To add Litware claims rules to ADFS

* 1. If you do not have the **Edit Claim Rules for Litware IP** dialog open from the previous task, then on the machine where you have installed ADFS, start the AD FS 2.0 Management tool. Expand **Trust Relationships**, then click on **Claims Provider Trusts**, then right-click on **Litware IP – Lab 04**, and then select **Edit Claim Rules**. You should be able to see the dialog shown in the following screenshot.
     1. 
  2. Now you can add the rules you need to test the scenario. In the **Edit Claim Rules for Litware IP – Lab 04** dialog, click **Add Rule.** Then, in **Add Transform Claim Rule Wizard** dialog, in the **Claim rule template** dropdown list, select **Send Claims Using a Custom Rule**, and then click **Next**.
     1. 
  3. On the **Configure Rule** page of the wizard, in the **Claim rule name** textbox, type **Pass through Name**. In the **Custom rule** textbox, add the following rule:
     1. c:[Type == "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"] => issue (claim = c);
  4. On the **Configure Rule** page of the wizard, click **Finish**.
  5. Using the information in the following table, repeat steps 2, 3 and 4 to add the remaining rules to ADFS.

|  |  |
| --- | --- |
| Claim rule name | Custom rule |
| Transform group (Sales) to group (Litware\Sales) | c:[Type == "http://schemas.xmlsoap.org/claims/group", Value =~ "^(?i)Sales$"]  => issue(Type = "http://schemas.xmlsoap.org/claims/group", Issuer = c.Issuer, OriginalIssuer = c.OriginalIssuer, Value = "Litware\Sales", ValueType = c.ValueType); |
| Issue Organization | => issue(Type = "http://schemas.adatum.com/claims/2009/08/organization", Value = "Litware"); |

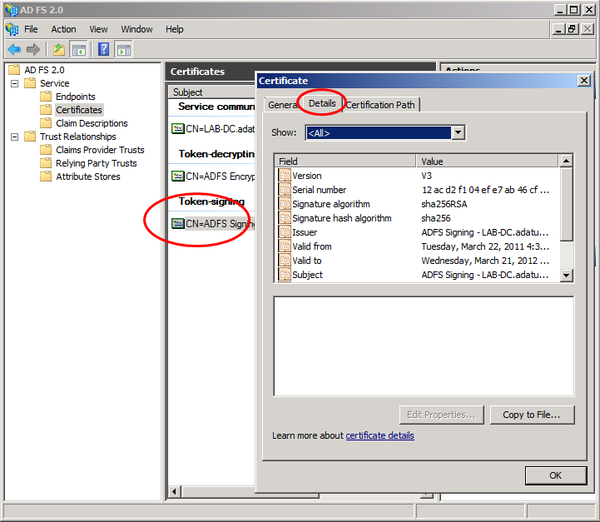
* 1. When you have finished, the **Edit Claim Rules for Litware IP – Lab 04** dialog should look like the following screenshot. Click **OK** to close the dialog.
     1. 

You have now completed this task to add the Litware IP claims rules to the Litware IP claims provider trust in ADFS that you can use to test the a-Order application.

## Task 6: Modify the Active Client Application

* 1. In this task, you will modify the active client application to work with ADFS instead of the simulated Adatum federation provider.

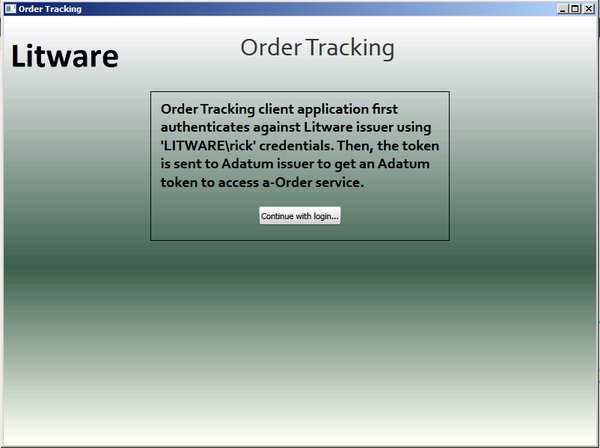
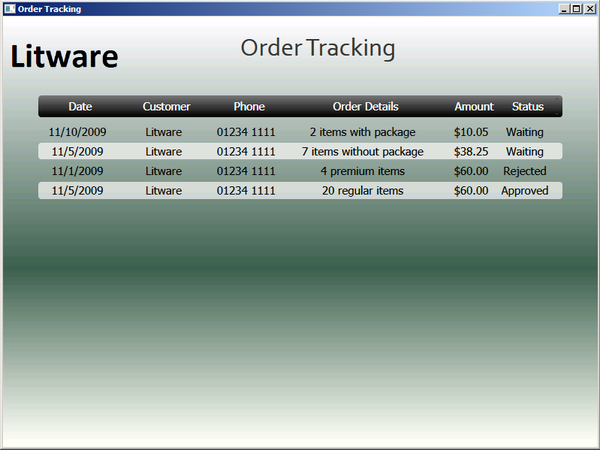
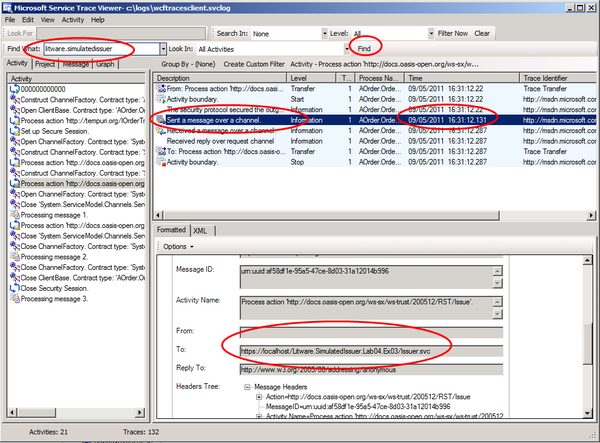
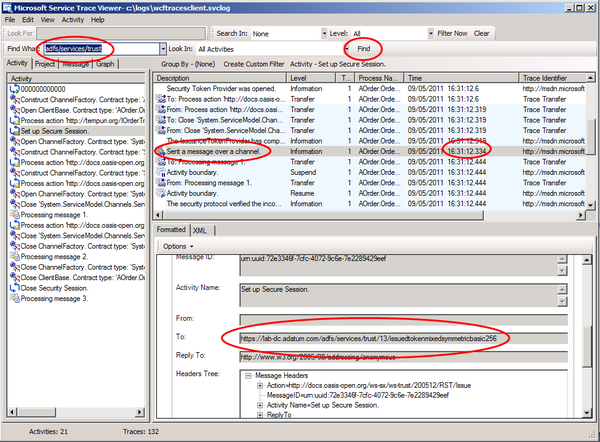
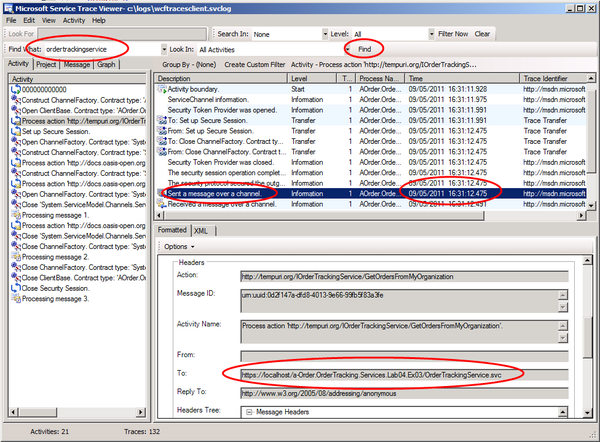
To modify the active client application

* 1. Start Visual Studio as an administrator and open the solution named **Lab04.Ex03.sln** from the **Lab04-ActiveClient\Source\Ex03\Begin** folder.
  2. In Visual Studio, open the file **App.config** from the **a-Order.OrderTracking.Client.Lab04.Ex03** project into the code editor window.
  3. Modify the **ws2007FederationHttpBinding** section to add details of your ADFS instance to the dictionary. Modify the following highlighted **issuer** element, and replace the **{issuer-url}** with the host name of your ADFS server.
     1. XML
     2. <ws2007FederationHttpBinding>
     3. <binding name="WS2007FederationHttpBinding\_IOrderTrackingService">
     4. <security mode="TransportWithMessageCredential">
     5. <message>
     6. **<issuer address=**
     7. **"https://{issuer-url}/adfs/services/trust/13/issuedtokenmixedsymmetricbasic256"**
     8. **binding="customBinding"**
     9. **bindingConfiguration="AdatumIssuerIssuedToken" />**
     10. </message>
     11. </security>
     12. </binding>
     13. </ws2007FederationHttpBinding>
     14. You should enter the **hostname** in all lower-case characters.
  4. In Solution Explorer, right-click the top-level **Lab04.Ex03** solution item and click **Rebuild solution**. Correct any errors.
  5. Close all of the files open in the Visual Studio editor, making sure to save your changes, but keep the solution open so that you are ready to use it in the next task.
  6. Finally, you must add the certificate that ADFS uses to sign the tokens that it issues to the **Trusted People** certificate store on the machine where you are running the client application so that the client application can authenticate the token. On the machine where you have installed ADFS, start the **AD FS 2.0 Management** console.
  7. Expand the **Service** node and then click **Certificates**. Right-click the **Token-signing** certificate and click **View Certificate**. The click the **Details** tab.
     1. 
  8. Click **Copy to File**, on the welcome page of the **Certificate Export Wizard** click **Next**, select **DER encoded binary X.509 (.CER)**,and click **Next**. On the **File to Export** page, click **Browse**, select a suitable folder, and use **ADFSSigningCert.cer** as the file name. Click **Next** and then click **Finish**.
  9. If you are running ADFS on a separate machine, copy the **ADFSSigningCert.cer** file to the machine where you will run the service.
  10. On the machine where you will run the service, start the **MMC** console and add the **Certificates** snap-in for the local computer. In the **Trusted People** node, right-click **Certificates**, point to **All Tasks**, and then click **Import**.
  11. On the welcome page of the **Certificate Import Wizard,** click **Next**. On the **File to Import** page, click **Browse** and locate the **ADFSSigningCert.cer** file. In the **Certificate Import Wizard**, click **Next**, ensure that **Trusted People** is the selected store, click **Next**, and then click **Finish**.
  12. You have now completed this task to modify the Litware issuer to work with your ADFS instance.

## Task 7: Verify the Solution

* 1. In this task, you will verify the solution by running the solution and using WCF tracing to examine the interaction between the client, the identity provider, ADFS (the federation provider), and the Web service.
  2. If you are running ADFS on a separate machine (or virtual machine) from the development machine where you are completing the labs, you must make sure that you have installed the ADFS service certificate on the development machine. See the section "*Integrating with Active Directory Federation Services*" in the *Introduction* document for these labs for details.

To verify the solution

* 1. In Visual Studio, on the **Debug** menu, click **Start Without Debugging**. In the Order Tracking application, click **Continue with login**.
     1. 
  2. The client application authenticates with the Litware issuer, obtains a token from the Adatum federation provider, and connects to the Web service. The Web service returns Litware orders for the client application to display.
     1. 
  3. To examine the messages that the client application exchanges with the Web services, you can use the WCF trace facility. Continue with the **Lab04.Ex03** solution you used in the previous task. In Visual Studio, open the file **App.config** from the **a-Order.OrderTracking.Client.Lab04.Ex03** project into the code editor window.
  4. Add the **system.diagnostics** section highlighted in the following snippet, and save the changes.
     1. XML
     2. **<system.diagnostics>**
     3. **<sources>**
     4. **<source name="System.ServiceModel"**
     5. **switchValue="Information, ActivityTracing"**
     6. **propagateActivity="true">**
     7. **<listeners>**
     8. **<add name="xml"/>**
     9. **</listeners>**
     10. **</source>**
     11. **<source name="System.ServiceModel.MessageLogging">**
     12. **<listeners>**
     13. **<add name="xml"/>**
     14. **</listeners>**
     15. **</source>**
     16. **</sources>**
     17. **<sharedListeners>**
     18. **<add name="xml"**
     19. **type="System.Diagnostics.XmlWriterTraceListener"**
     20. **initializeData= "c:\WCFTraces.svclog" />**
     21. **</sharedListeners>**
     22. **</system.diagnostics>**
     23. </configuration>
     24. Make sure that you have permission to write to the location you have chosen to write your log file.
  5. Repeat steps 1 and 2 to run the application and collect the trace data.
  6. In Windows Explorer, double click on the **WCFTraces.svclog** file that contains your trace data.
     1. In the WS-Trust process, the application first connects to the identity provider — the Litware issuer. In the **Find What** box, type **Litware.simulatedissuer** and click **Find**. Note the time when the client application called the Litware issuer.  
        
  7. The client application next connects to the federation provider — ADFS. In the **Find What** box, type **adfs/services/trust** and click **Find**, keep clicking **Find** until you find a **Sent a message over a channel** message (You may need to click on the first activity in the **Activity** list to start the search from the beginning of the trace file). Note the time when the client application called ADFS: it will be later than the time that it called the Litware issuer.
     1. 
  8. The client application finally connects to the a-Order service. In the **Find What** box, type **ordertrackingservice** and click **Find**, keep clicking **Find** until you find a **Sent a message over a channel** message (You may need to click on the first activity in the **Activity** list to start the search from the beginning of the trace file). Note the time when the client application called the service: it will be later than the time that it called the Adatum federation provider.
     1. 
  9. You have now completed this task and this exercise. You have seen how to configure a federated identity environment using ADFS as the federation provider: users from Litware can access the a-Order application using credentials from their own identity provider.
  10. Federated identity is a powerful feature of claims; users do not need to remember additional sets of credentials to access a partner's application, and it is easy for an organization such as Adatum to configure its federation provider to trust issuers from additional partners without the requirement to modify the relying party application. Furthermore, the relying party application no longer has the responsibility for verifying the user's username and password: this is now handled by an identity provider. ADFS handles all of the federation tasks for Adatum.

## Running the "End" Solution

1. If you did not complete all of the tasks in this exercise, you can run the "end" solution we provide.

To run the end solution

* 1. You must complete task 1, starting at step3, to configure the a-Order application to trust your ADFS instance. Use the solution **Lab04.Ex03.End.sln** from the **Lab04-ActiveClient\Source\Ex03\End** folder, the project name in the "end" solution is **Adatum.FederationProvider.Lab04.Ex03.End**
  2. You should run the PowerShell script, **Add-aOrder-RelyingParty.ps1** in the Lab04-ActiveClient \Source\Ex03\Assets folder to add the a-Order application as a relying party in ADFS.
  3. You can run the PowerShell script, **Add-aOrder-Rules.ps1** in the Lab04-ActiveClient \Source\Ex03\Assets folder to add the a-Order application claims rules to ADFS.
  4. You can run the PowerShell script, **Add-Litware-ClaimsProviderTrust.ps1** in the Lab04-ActiveClient \Source\Ex03\Assets folder to add the Litware issuer as a claims trust provider in ADFS.
     1. To run this script, you must first **cd** to the **Lab04-ActiveClient\Source\Ex03\Assets** folder at the **PowerShell** prompt.
  5. You can run the PowerShell script, **Add-Litware-Rules.ps1** in the Lab04-ActiveClient \Source\Ex03\Assets folder to add the Litware claims rules to ADFS.
  6. You must complete task 6 to configure the client application to work with your ADFS instance.
  7. You can follow the steps in task 7 to verify your solution.