Azure Active Directory Preview: Web Sign-On with Windows Azure Active Directory – JAVA Walkthrough

Abstract: This document provides instructions on setting up a JAVA web application and configuring it to leverage Windows Azure Active Directory to accept users from Office365 customers (the “WebSSO” initiative).

Table of Contents

[Overview 1](#_Toc320521208)

[Prerequisites 2](#_Toc320521209)

[Assets 2](#_Toc320521210)

[Generic Environment Requisites 2](#_Toc320521211)

[Java-specific Requisites 2](#_Toc320521212)

[Walkthrough 2](#_Toc320521213)

[ISV - Creating a Simple WAR Application 2](#_Toc320521214)

[Customers - Provisioning a New Application in the Directory Tenant 7](#_Toc320521215)

[ISV - Protect the Application via WS-Federation and Onboard the First Customer 9](#_Toc320521216)

[ISV - Configuring the Application to Handle Sign On with Multiple Tenants 19](#_Toc320521217)

[Document History 24](#_Toc320521218)

[Appendix: Deploying to JBOSS on Solaris 24](#_Toc320521219)

# Overview

This document provides instructions on setting up an ASP.NET MVC web application and configuring it to leverage Azure Active Directory (AAD) to accept users from Office365 customers.  
The code and instructions are based on the features of the “WebSSO” initiative as it exists today and are meant to accomplish a short-term goal: helping the Azure Active Directory Preview participations to quickly get up to speed with AAD and the use of WS-Federation in their development platform of choice.

The walkthrough accomplishes the task by guiding the reader thru four distinct tasks, performed alternatively by two roles:

* Fabrikam, an ISV exposing the web application
* Awesome Computers and Trey Research, playing the role of Office365 customers

The four tasks are:

* ISV: Create one simple JAVA web application
* Customer: Provision Fabrikam’s web application in their tenant
* ISV: Protect the application via WS-Federation and onboard the first customer
* ISV: Modify the application to handle sign on with multiple tenants

## Prerequisites

The walkthrough relies on various prerequisites being met by the software environment offered by the target machine.

Note that an expert Java developer should be able to easily apply the techniques shown here to any other application server. That holds for other platforms, too: the application created with this walkthrough has been successfully deployed and tested on JBOSS running on Solaris.

### Assets

The walkthrough comes with some scripts and sample code which helps with some of the most tedious tasks. All the assets are available in the archive WAAD.WebSSO.JAVA.ZIP.  
All the source code is available for the developer to study and tweak at will.

### Generic Environment Requisites

* Internet Information Services (IIS) 7.5 (SSL enabled)
* Windows Powershell
* Office 365 PS

### Java-specific Requisites

* Java Runtime Environment 1.6 ( <http://www.oracle.com/technetwork/java/javase/downloads/jre-6u31-download-1501637.html> )
* JBoss 7.1.1.Final ( <http://www.jboss.org/jbossas/downloads/> )
* JBoss Studio ( <https://devstudio.jboss.com/earlyaccess/> )

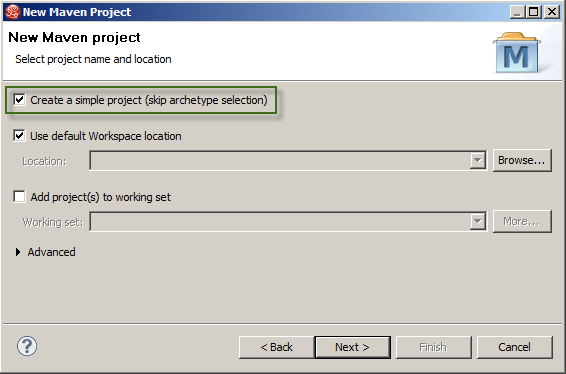
# Walkthrough

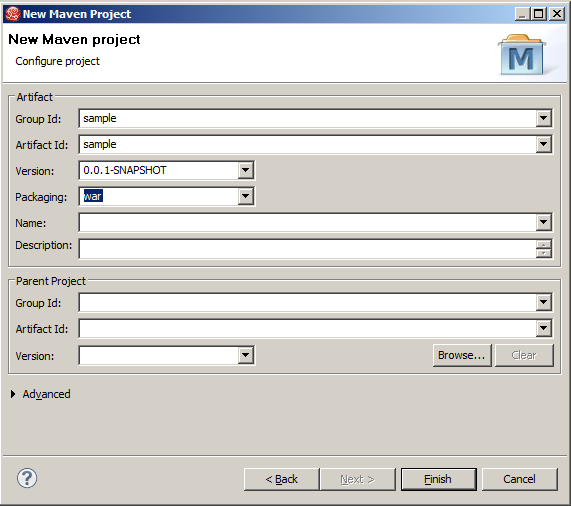
## ISV - Creating a Simple WAR Application

In this first task you will create a simple Java web application, which will become our mockup protected resource.

|  |
| --- |
| ***Note for the non-Java developer****:Maven is a dependency manager and build system. JBOSS Developer Studio, which is based on Eclipse, does not offer those capabilities out of the box.* |

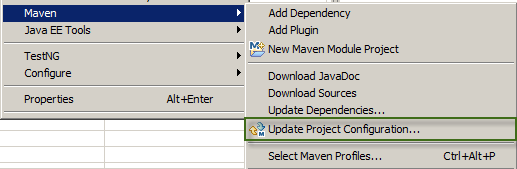
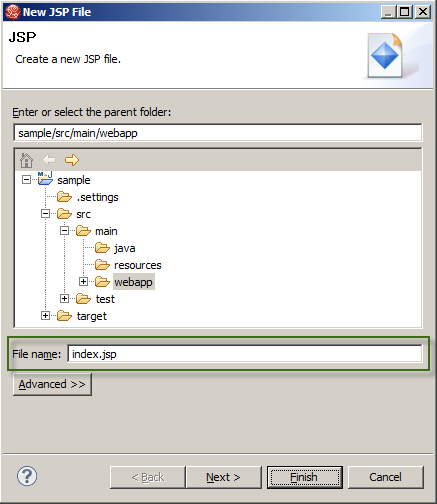
Fabrikam’s developers create a new **Maven Project** using JBoss Developer Studio by performing the following steps:

1. Open a new instance of JBoss Developer Studio.
2. Create a new project: ***File -> New Project -> Maven Project****.*
3. In the first wizard window select the ‘*Create a simple project*’ option and click **Next**.  
     
   
4. Provide a **Group Id**, an **Artifact Id**, and for Packaging select **war**. Click **Finish**.



1. Open the **pom.xml** file in the **sample** project and add the following xml inside the **project** node to configure the repositories for the external libraries and plugins and to target the project to Java 1.6.

|  |
| --- |
| <repositories>  <repository>  <id>jboss-public-repository-group</id>  <name>JBoss Public Maven Repository Group</name>  <url>https://repository.jboss.org/nexus/content/groups/public-jboss/</url>  <layout>default</layout>  <releases>  <enabled>true</enabled>  <updatePolicy>never</updatePolicy>  </releases>  <snapshots>  <enabled>true</enabled>  <updatePolicy>never</updatePolicy>  </snapshots>  </repository>  </repositories>  <pluginRepositories>  <pluginRepository>  <id>jboss-public-repository-group</id>  <name>JBoss Public Maven Repository Group</name>  <url>https://repository.jboss.org/nexus/content/groups/public-jboss/</url>  <layout>default</layout>  <releases>  <enabled>true</enabled>  <updatePolicy>always</updatePolicy>  </releases>  <snapshots>  <enabled>true</enabled>  <updatePolicy>always</updatePolicy>  </snapshots>  </pluginRepository>  </pluginRepositories>  <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>2.0.2</version>  <configuration>  <source>1.6</source>  <target>1.6</target>  </configuration>  </plugin>  </plugins>  </build> |

1. **Right-click** the **sample** project and select ***Maven -> Update Project Configuration…*** to refresh the project and apply **pom.xml** file changes. Select both projects and click **OK**.
2. 
3. Select the sample project, **right-click** and select ***New -> JSP File*** (name it index.jsp).  
     
   
4. Replace the generated code with the following:

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>  <title>Index</title>  </head>  <body>  <h3>Index Page</h3>  </body>  </html> |

1. **Right-click** the **sample** project and select ***Run As -> Run on Server.***
2. Open a Powershell console and run the following command to generate a new GUID for the application:

|  |
| --- |
| PS C:\Windows\system32> [guid]::NewGuid()  Guid  ----  9a822147-348b-4e0e-8edf-899fe8c117d4 |

|  |
| --- |
| ***Note****: This identifier will be the* ***AppPrincipalId*** *used in further customer provisioning that want to access the application using their Office 365 accounts.* |

## 

## Customers - Provisioning a New Application in the Directory Tenant

Awesome Computers, an Office 365 customer, acquired the right[[1]](#footnote-1) to access Fabrikam’s application (for example: they bought a license from Fabrikam). In this task we show how Awesome Computer’s administrators provision the application in their tenant and communicate back to Fabrikam the data they need in order to set up single sign on for Awesome Computers’ employees. The desired result is allowing Awesome Computers’ employees to authenticate with Fabrikam’s web application exactly in the same way in which they authenticate with their Office365 applications.

|  |
| --- |
| ***Note****: If you don’t have access to an Office365 tenant for this walkthrough, you can obtain one by applying for a FREE TRIAL subscription on Office 365’s Sign-up* [*page*](http://www.microsoft.com/en-us/office365/online-software.aspx#fbid=8qpYgwknaWN)*.* |

In a nutshell, in order to register for Fabrikam’s application Awesome Computers’ administrator needs to create a new Service Principal for it in their tenant.

Fabrikam somehow[[2]](#footnote-2) provides to Awesome Computers:

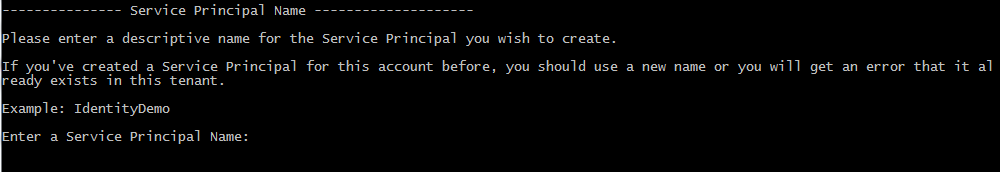
* The value of the **ServicePrincipalName** (*sample/localhost:8443*)
* the **AppPrincipalId** (*9a822147-348b-4e0e-8edf-899fe8c117d4*)
* the **ReplyUrl**

Awesome Computers’ administrator performs the following steps:

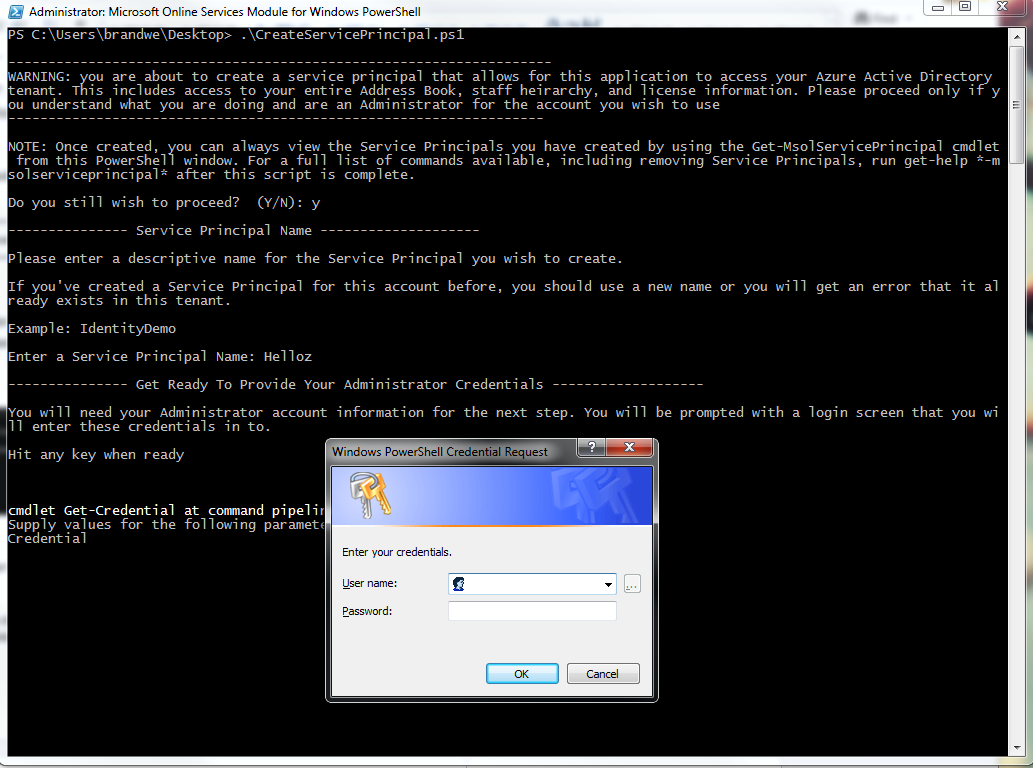
1. If you didn’t do it yet, download and install a set of Powershell scripts from Office 365’s online help [page](http://onlinehelp.microsoft.com/en-us/office365-enterprises/hh124998.aspx).
2. Locate the **CreateServicePrincipal.ps1** script in this code example set under WAAD.WebSSO.JAVA/Scripts
3. From the **Start Menu**, Open the *“Microsoft Online Services Module for Windows PowerShell”* console.
4. Run the SampleAppServicePrincipal.ps1 command from the Microsoft Online Services Module for Windows PowerShell Console

|  |
| --- |
| PS C:\Windows\system32> ./CreateServicePrincipal.ps1 |

You will be asked to pick a name for your Service Principal. This just needs to be a descriptive name that you can remember in case you wish to inspect or remove the Service Principal later on.



1. You will be prompted to enter your administration credentials for your Office365 tenant:



1. If the script is successful, you will get output like indicated below. **You will want to capture these items for use later in the walkthrough**:

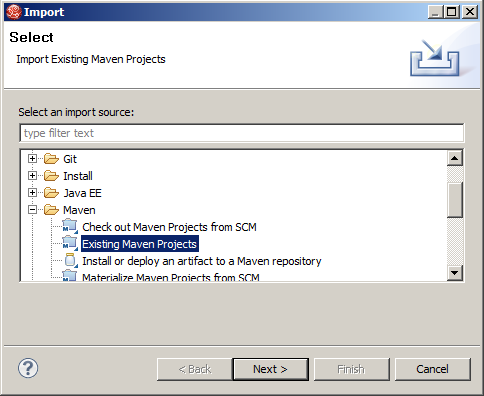
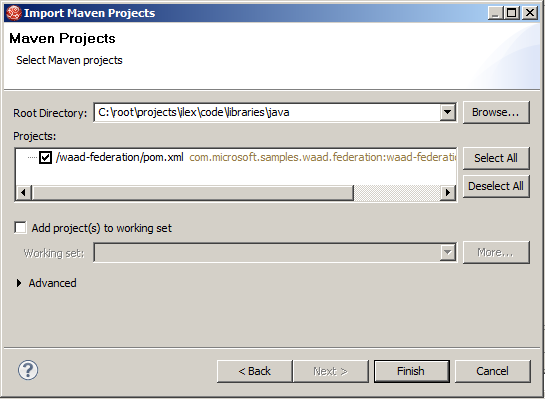
|  |
| --- |
| Setting permissions to allow the Service Principal to have Read Only access to your Azure Active Directory tenant. See the P  owerShell script to see how this is done.  --------------- Script is complete ----------------------  Company ID: eda7500a-ad60-4648-9019-89dddd37673b  **AppPrincipal ID: a4ba298c-0d1c-4332-b85f-7ab34743d377**  App Principal Secret: 0guCAQnwLuTtpa+jWkam1Y/gcGsRzEOOOA2+Illt5bQ=  **Audience URI: a4ba298c-0d1c-4332-b85f-7ab34743d377@eda7500a-ad60-4648-9019-89dddd37673b**  Usage : Verify |
|  |
| ***Note****: In the command shown here,* ***AppPrincipalId*** *values are those provided by Fabrikam.* |

That’s it: Fabrikam’s application has been provisioned in the directory tenant of Awesome Computer.

Now Fabrikam needs to provision Awesome Computers as a customer of the application: all it takes is for Fabrikam to know that users from the Office365 tenant with domain *awesomecomputers.onmicrosoft.com* should be granted access. How that information reaches Fabrikam will depend on how subscriptions are handled, in this walkthrough we just assume it happened without detailing how.

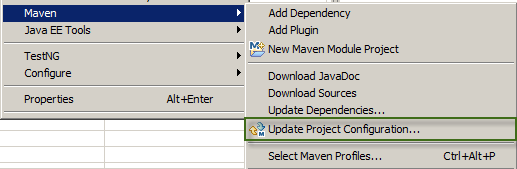
## ISV - Protect the Application via WS-Federation and Onboard the First Customer

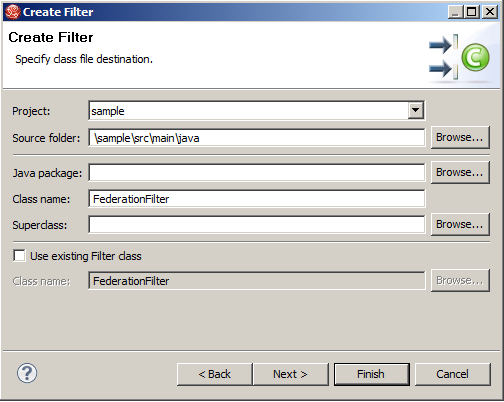
Let’s get back to Fabrikam. The application created in step1 is still unfinished: we need to add support for federated login. We’ll do it by using the **waad-federation** library and adding some extra artifacts, like a login page. With the app ready to authenticate requests using the WS-Federation protocol, we’ll add the WAAD tenant of Awesome Computers as a trusted provider.

1. Import the **waad-federation** library from JBoss Developer Studio in the same workspace you created the **sample** application: ***File -> Import -> Existing Maven Projects***.   
     
   
2. Select the folder where the **waad-federation** library is located and click **Finish**.  
     
   
3. Open the **pom.xml** file in the **sample** project and add the following xml inside **project** node to configure the project’s dependencies.

|  |
| --- |
| <dependencies>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>3.0-alpha-1</version>  </dependency>  <dependency>  <groupId>com.microsoft.samples.waad.federation</groupId>  <artifactId>waad-federation</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  </dependencies> |

1. **Right-click** the **sample** project and select ***Maven -> Update Project Configuration…*** to refresh the project and apply **pom.xml** file changes. Select both projects and click **OK**.



1. Create a Filter. **Right-click** the **sample** project and select ***New -> Filter.*** For “Class name” type **FederationFilter** and click **Finish**.  
     
   
2. Replace the generated code with the following:

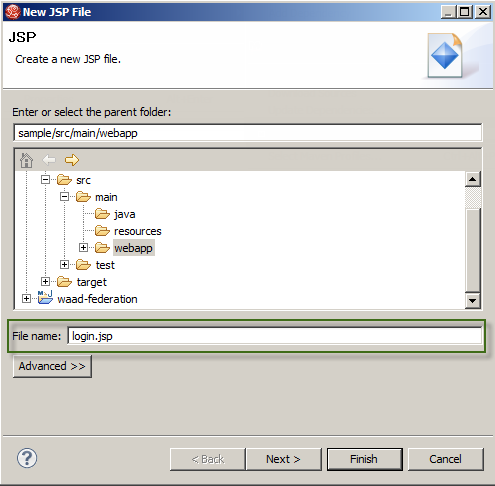
|  |
| --- |
| **import** java.io.IOException;  **import** javax.servlet.Filter;  **import** javax.servlet.FilterChain;  **import** javax.servlet.FilterConfig;  **import** javax.servlet.ServletException;  **import** javax.servlet.ServletRequest;  **import** javax.servlet.ServletResponse;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** java.util.regex.\*;  **import** com.microsoft.samples.federation.FederatedLoginManager;  **import** com.microsoft.samples.federation.URLUTF8Encoder;  **public** **class** FederationFilter **implements** Filter {  **private** String loginPage;  **private** String allowedRegex;  **public** **void** init(FilterConfig config) **throws** ServletException {  **this**.loginPage = config.getInitParameter("login-page-url");  **this**.allowedRegex = config.getInitParameter("allowed-regex");  }  **public** **void** doFilter(ServletRequest request, ServletResponse response,  FilterChain chain) **throws** IOException, ServletException {  **if** (request **instanceof** HttpServletRequest) {  HttpServletRequest httpRequest = (HttpServletRequest) request;  **if** (!httpRequest.getRequestURL().toString().contains(**this**.loginPage)) {  FederatedLoginManager loginManager = FederatedLoginManager.*fromRequest*(httpRequest);  **boolean** allowedUrl = Pattern.*compile*(**this**.allowedRegex).matcher(httpRequest.getRequestURL().toString()).find();  **if** (!allowedUrl && ! loginManager.isAuthenticated()) {  HttpServletResponse httpResponse = (HttpServletResponse) response;  String encodedReturnUrl = URLUTF8Encoder.*encode*(httpRequest.getRequestURL().toString());  httpResponse.setHeader("Location", **this**.loginPage + "?returnUrl=" + encodedReturnUrl);  httpResponse.setStatus(302);  **return**;  }  }  }  chain.doFilter(request, response);  }  **public** **void** destroy() {  }  } |

1. Open the **web.xml** located in the **src/main/webapp/WEB-INF** folder. Add the following piece inside the **web-app** node:

|  |
| --- |
| <filter>  <filter-name>FederationFilter</filter-name>  <filter-class>FederationFilter</filter-class>  <init-param>  <param-name>login-page-url</param-name>  <param-value>/sample/login.jsp</param-value>  </init-param>  <init-param>  <param-name>allowed-regex</param-name>  <param-value>(\/sample\/login.jsp|\/sample\/wsfed-saml|\/sample\/oauth)</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>FederationFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

|  |
| --- |
| ***Note****: The filter will handle the secured and unsecured pages and will also redirect users to the login page (defined as* ***login-page-url*** *filter’s parameter) if they are not authenticated.*  *However, the filter will not apply to the incoming Urls that match the* ***allow-regex*** *regular expression parameter.* |

1. Create a login page. Select the sample project, **right-click** and select ***New -> JSP File*** (name it login.jsp)



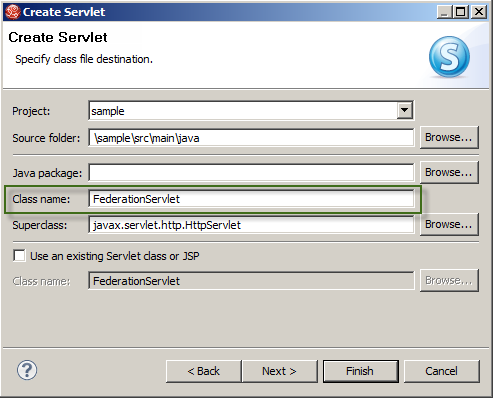
1. Replace the generated code with the following:

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <%@ page import=*"com.microsoft.samples.federation.\*"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>  <title>Login Page</title>  </head>  <body>  <h3>Login Page</h3>  <a href=*"*<%=FederatedLoginManager.getFederatedLoginUrl(request.getParameter("returnUrl"))%>*"*><%=FederatedConfiguration.getInstance().getStsFriendlyName()%></a>  </body>  </html> |

1. From JBoss Developer Studio, **right-click** the **src/main/resources** folder in the **sample** project, select ***New -> Properties file***, name it **federation** and provide the following configuration:

**NOTE:** audienceuris= and realm= are the values you retrieved from the PowerShell command above. Remember that you must add **spn:** to be beginning of this value. Use the audienceuri for both values below.

|  |
| --- |
| federation.trustedissuers.issuer=https://accounts.accesscontrol.windows.net/v2/wsfederation  federation.trustedissuers.thumbprint=3F5DFCDF4B3D0EAB9BA49BEFB3CFD760DA9CCCF1  federation.trustedissuers.friendlyname=Awesome Computers  federation.audienceuris=spn:9a822147-348b-4e0e-8edf-899fe8c117d4@495c4a5e-38b7-49b9-a90f-4c0050b2d7f7  federation.realm=spn:9a822147-348b-4e0e-8edf-899fe8c117d4@495c4a5e-38b7-49b9-a90f-4c0050b2d7f7  federation.reply=https://localhost:8443/sample/wsfed-saml |

1. Create the new Servlet. **Right-click** the **sample** project and select ***New -> Other -> Servlet.***Name it **FederationServlet**, click **Next** and then **Finish**.  
     
   
2. Open the **FederationServlet.java** file and replace the generated code with the following:

|  |
| --- |
| **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** com.microsoft.samples.federation.FederatedAuthenticationListener;  **import** com.microsoft.samples.federation.FederatedLoginManager;  **import** com.microsoft.samples.federation.FederatedPrincipal;  **import** com.microsoft.samples.federation.FederationException;  **public** **class** FederationServlet **extends** HttpServlet {  **private** **static** **final** **long** *serialVersionUID* = 1L;    **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  String token = request.getParameter("wresult").toString();    **if** (token == **null**) {  response.sendError(400, "You were supposed to send a wresult parameter with a token");  }    FederatedLoginManager loginManager = FederatedLoginManager.*fromRequest*(request, **new** SampleAuthenticationListener());  **try** {  loginManager.authenticate(token, response);  } **catch** (FederationException e) {  response.sendError(500, "Oops! and error occurred.");  }  }    **private** **class** SampleAuthenticationListener **implements** FederatedAuthenticationListener {  @Override  **public** **void** OnAuthenticationSucceed(FederatedPrincipal principal) {  // \*\*\*  // do whatever you want with the principal object that contains the token's claims  // \*\*\*  }  }  } |

1. Open the **web.xml** located inside the **src/main/webapp/WEB-INF** folder and replace the url-pattern **“/FederationServlet**” with “**/ws-saml**”.

|  |
| --- |
| <servlet>  <description></description>  <display-name>FederationServlet</display-name>  <servlet-name>FederationServlet</servlet-name>  <servlet-class>FederationServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>FederationServlet</servlet-name>  <url-pattern>/wsfed-saml</url-pattern>  </servlet-mapping> |

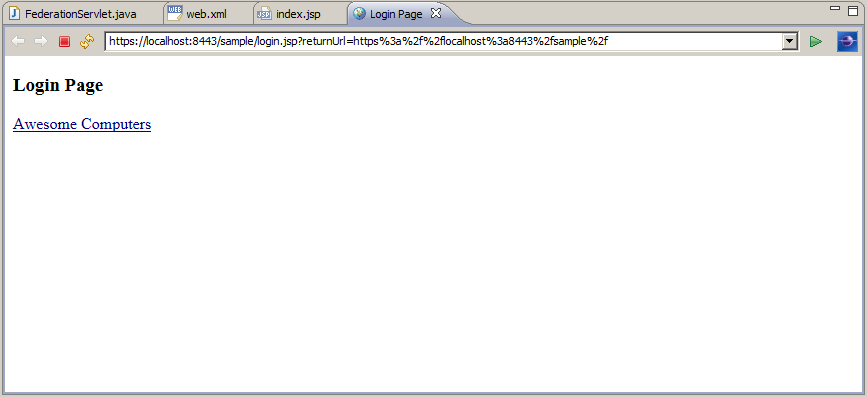
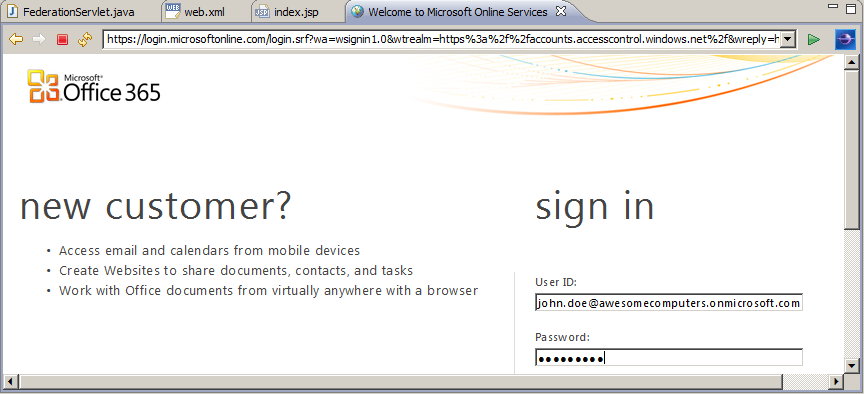
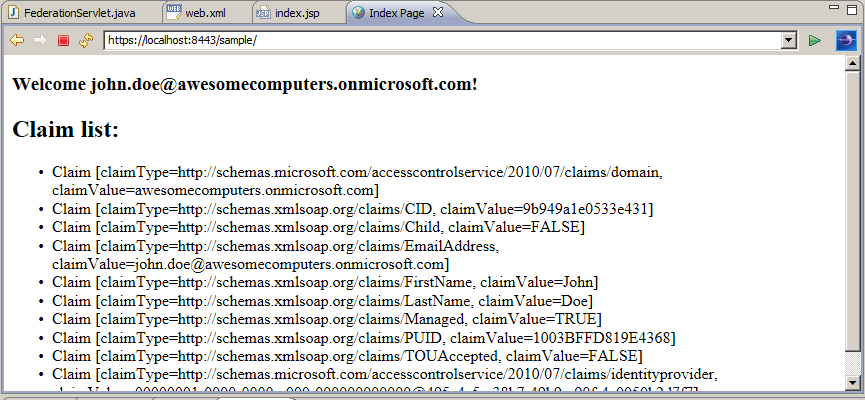
1. Open the **index.jsp** file and replace the existing code with the following:

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"* pageEncoding=*"ISO-8859-1"*%>  <%@ page import=*"com.microsoft.samples.federation.\*"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>  <title>Index Page</title>  </head>  <body>  <h3>Welcome <strong><%=FederatedLoginManager.fromRequest(request).getPrincipal().getName()%></strong>!</h3>    <h2>Claim list:</h2>  <ul>  <%  **for** (Claim claim : FederatedLoginManager.fromRequest(request).getClaims()) {  %>  <li><%= claim.toString()%></li>  <% } %>  </ul>  </body>  </html> |

1. Open the **web.xml** located in the **src/main/webapp/WEB-INF** folder and add this node under **web-app** to make the application run over SSL:

|  |
| --- |
| <security-constraint>  <web-resource-collection>  <web-resource-name>SSL Forwarding</web-resource-name>  <url-pattern>/\*</url-pattern>  <http-method>POST</http-method>  <http-method>GET</http-method>  </web-resource-collection>  <user-data-constraint>  <transport-guarantee>CONFIDENTIAL</transport-guarantee>  </user-data-constraint>  </security-constraint> |

|  |
| --- |
| ***Note****: Ensure that JBoss server is already configured to support SSL.* |

1. **Right-click** the **sample** project and select ***Run As -> Run on Server,*** click **Finish** and you should be able to see the login page with the “Awesome Computers” link.  
     
   
2. Once on the Office 365 identity provider page, you can log in using your **awesomecomputers.onmicrosoft.com** credentials (e.g. [john.doe@awesomecomputers.onmicrosoft.com](mailto:john.doe@awesomecomputers.onmicrosoft.com)).
3. Finally, if the login process is successful, you will be redirected to the secured page (**sample**/**index.jsp**) as an authenticated user.  
     
   

If for some reason your application is meant to work with a single Office365 tenant, for example if you are writing a LoB application, you can stop here. What you have seen so far enables you to offer to one arbitrary app the same single sign on experience you enjoy with Office365. If instead you are developing applications that need to be accessed by multiple tenants, the next section will help you to modify the code to accommodate your scenario.

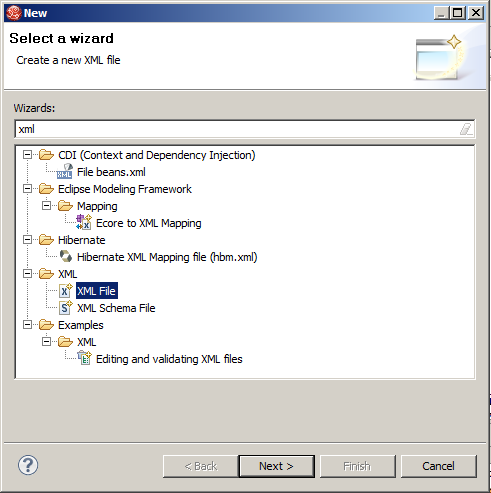
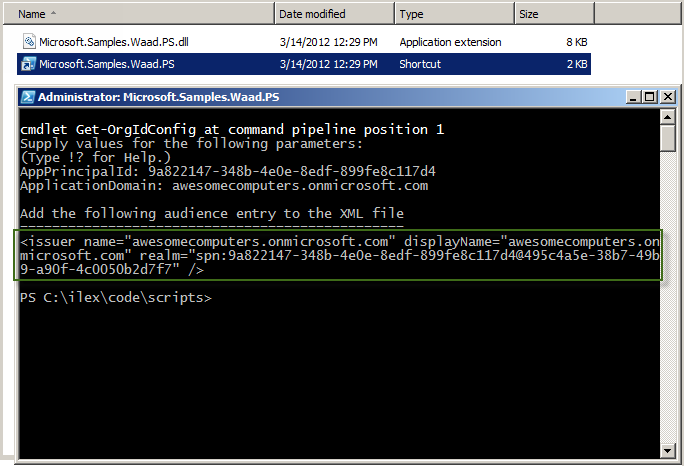
## ISV - Configuring the Application to Handle Sign On with Multiple Tenants

Let’s say that Fabrikam wants to sell access to its application to multiple customers. The technique demonstrated so far works with only one trusted provider: Fabrikam’s developers will need to make some changes in the application to accommodate sign on from both Awesome Computers and its future new customers. The main new features needed are:

* Support for multiple identity providers in the login page
* Maintain a list of the trusted providers and the audienceURI they’ll send to the application; use that list to determine how to validate incoming tokens

For the sake of this example we will add another fictitious customer, Trey research. Trey Research Inc. will register the application in its tenant as shown for Awesome Computers in the Customers task earlier in the walkthrough.

Here there’s the list of tweaks that need to happen in the app to make it enable multi-tenant sign-on, intertwined with the provisioning of Trey Research.

1. From JBoss Developer Studio, **right-click** the **src/main/resources** folder in the **sample** project, select **New -> Xml File** and provide *“trusted.issuers.xml”* as the file name. This file will contain a list of the trusted issuers for the application (in this case with Awesome Computers and Trey Research Inc.) which will be used by the dynamic audience Uri validator.  
     
   
2. Go to the scripts folder and open the **Microsoft.Samples.Waad.Federation.PS** link to generate the trusted issuers’ nodes to add to the XML repository. It will ask you for the **AppPrincipalId** and the **AppDomain** name to generate the issuer node as depicted below:  
     
   

|  |
| --- |
| ***Note****: The script retrieves the federation metadata directly from Windows Azure Active Directory to get the issuer identifier for generating the realm’s SPN value.* |

1. Open the XML file, create an **issuers** root node and include the output node:

|  |
| --- |
| <issuers>  <issuer name=*"awesomecomputers.onmicrosoft.com"* displayName=*"awesomecomputers.onmicrosoft.com "* realm=*"spn:9a822147-348b-4e0e-8edf-899fe8c117d4@495c4a5e-38b7-49b9-a90f-4c0050b2d7f7"* />  </issuers> |

1. Repeat Step 2 to generate Trey Research Inc. node. Notice that you can change the display name to show a user-friendly name.

|  |
| --- |
| <issuers>  <issuer name=*"awesomecomputers.onmicrosoft.com"* displayName=*"Awesome Computers"* realm=*"spn:9a822147-348b-4e0e-8edf-899fe8c117d4@495c4a5e-38b7-49b9-a90f-4c0050b2d7f7"* />  <issuer name=*"treyresearchinc.onmicrosoft.com"* displayName=*"Trey Research Inc."* realm=*"spn:9a822147-348b-4e0e-8edf-899fe8c117d4@13292593-4861-4847-8441-6da6751cfb86"* />  </issuers> |

1. Open the **login.jsp** file and replace the import declaration ***com.microsoft.samples.federation*** with ***com.microsoft.samples.waad.federation***.

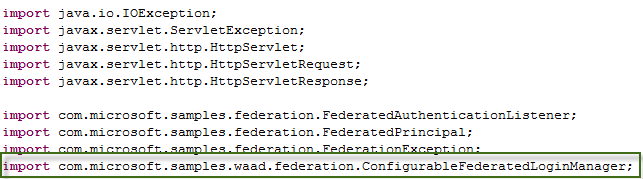
|  |
| --- |
| <%@ page import=*"com.microsoft.samples.waad.federation.\*"*%> |

1. Replace the old link (created for the first trusted issuer) with the following snippet to list all the trusted issuers from the XML repository:

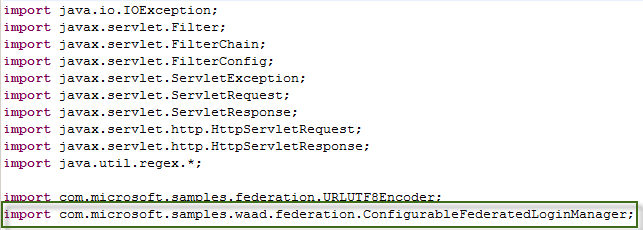
|  |
| --- |
| <ul>  <%  TrustedIssuersRepository repository = **new** TrustedIssuersRepository();  **for** (TrustedIssuer trustedIssuer : repository.getTrustedIdentityProviderUrls()) { %>  <li><a  href=*"*<%=trustedIssuer.getLoginURL(request.getParameter("returnUrl"))%>*"*><%=trustedIssuer.getDisplayName()%></a>  </li>  <% } %>  </ul> |

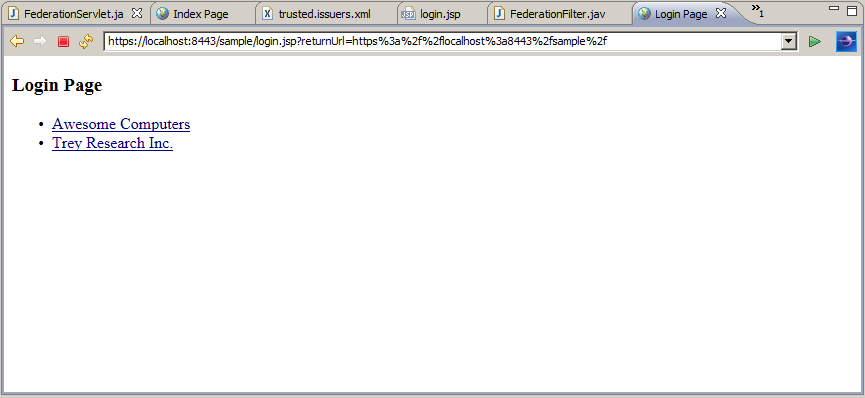
1. Open the **FederationServlet.java** file and replace the **FederatedLoginManager** class with **ConfigurableFederatedLoginManager**.

|  |
| --- |
| ConfigurableFederatedLoginManager loginManager = ConfigurableFederatedLoginManager.*fromRequest*(request, **new** SampleAuthenticationListener()); |

1. Also in the **FederationServlet.java** file, replace the **import** declaration ***com.microsoft.samples.federation.FederatedLoginManager*** with ***com.microsoft.samples.waad.federation.ConfigurableFederatedLoginManager***.  
     
   
2. Open the **FederationFilter.java** file and replace the **FederatedLoginManager** class with **ConfigurableFederatedLoginManager**.

|  |
| --- |
| ConfigurableFederatedLoginManager loginManager = ConfigurableFederatedLoginManager.fromRequest(httpRequest); |

1. Also in the **FederationFilter.java** file, replace the **import** declaration ***com.microsoft.samples.federation.FederatedLoginManager*** with ***com.microsoft.samples.waad.federation.ConfigurableFederatedLoginManager***.  
     
   
2. **Right-click** the **sample** project and select ***Run As -> Run on Server,*** and you should see a list with the links for each trusted identity provider retrieved from the *“trusted.issuers.xml”* repository.



|  |
| --- |
| ***Note****: The home realm discovery strategy of presenting an explicit list of trusted providers is not always feasible in practice. Here it is used for the sake of simplicity* |

Once you see the list in your browser, you can navigate to either providers: the authentication flow will unfold in the same way described in the former section. The application will validate the incoming token accordingly. You can try to delete entries in trusted.issuers.xml, as it would happen for example once a subscription expire, and verify that the application will reject authentication attempts from the corresponding provider.

# Appendix: Deploying to JBOSS on Solaris

This sample has been tested on JBOSS 7.1 running on Solaris. Here there are some quick instructions if you want to try it.

1. Download the Virtualbox Solaris appliance from <http://www.oracle.com/technetwork/server-storage/solaris11/downloads/virtual-machines-1355605.html>  
     
   **Note:** you must have an Oracle account to download the bits and accept the license when downloading it.
2. Import the Solaris Virtualbox Appliance (README inside the **OracleSolaris11\_11-11\_VM.zip** file).   
    **Note: DO NOT INSTALL THE VM ADDITIONS** as that might cause serious performance issues
3. Once the machine is up and running, open a **Terminal** window.
4. Copy WAAD.WebSSO.Java.ZIP to the Downloads folder
5. Go to the Downloads folder  
     
   cd ~/Downloads
6. Unzip the contents of the drop:  
     
   unzip ./WAAD.WebSSO.Java.ZIP -d ./ilex
7. Download **JBoss** 7.1:  
     
   curl <http://download.jboss.org/jbossas/7.1/jboss-as-7.1.1.Final/jboss-as-7.1.1.Final.zip> -o ./jboss-as-7.1.1.Final.zip
8. Unzip the **JBoss** 7.1 contents   
     
   unzip ./jboss-as-7.1.1.Final.zip -d ./
9. Open the standalone configuration file with GEdit from the **Teminal**:  
     
   gedit ./jboss-as-7.1.1.Final/standalone/configuration/standalone.xml
10. Locate the node with the **urn:jboss:domain:web:1.1** namespace, and replace the node by this one. Replace the **user\_name** value with the current user (username used to login to Solaris VM)   
      
    <subsystem xmlns="**urn:jboss:domain:web:1.1**" native="false" default-virtual-server="default-host">  
        <configuration>  
            <jsp-configuration development="true"/>  
        </configuration>  
        <connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" redirect-port="8443"/>  
        <connector name="https" protocol="HTTP/1.1" scheme="https" socket-binding="https" enable-lookups="false" secure="true">  
            <ssl password="Passw0rd!" certificate-key-file="/home/**{user\_name}**/Downloads/ilex/java/assets/https.keystore" protocol="TLSv1" verify-client="false" certificate-file="/home/**{user\_name}**/Downloads/ilex/java/assets/https.keystore"/>  
        </connector>  
        <virtual-server name="default-host" enable-welcome-root="true">  
            <alias name="localhost"/>  
            <alias name="[example.com](http://example.com)"/>  
        </virtual-server>  
    </subsystem>
11. Save the file, close **GEdit** and return to the **Terminal** window
12. Go to the server folder.   
      
    cd ~/Downloads/jboss-as-7.1.1.Final/bin
13. Start the server   
      
    ./standalone.sh

**Deploying the WAR file**

1. Open a new **Terminal** window.
2. Copy in the Download folder the WAR created in the walkthrough
3. Go to the server folder.   
     
   cd ~/Downloads/jboss-as-7.1.1.Final/bin
4. Run the JBoss client to deploy the application.  
     
   ./jboss-cli.sh
5. Connect to the server.  
     
   connect
6. Deploy the application using the file downloaded in the first step.  
     
   deploy ~/Downloads/sample.war --force

Open a browser and navigate <https://localhost:8443/sample/>

1. Please note that in this walkthrough we don’t give detailed indications on how Fabrikam makes its application available to potential customers. We also don’t specify how the customers circle back information to Fabrikam after provisioning. In actual settings the flow would likely be driven by some kind of marketplace or would follow whatever licensing and provisioning system the ISV has in place. [↑](#footnote-ref-1)
2. See note 1. The data listed here are the same for all prospect customers, hence depending on the subscription model those might even be simply publicly available. [↑](#footnote-ref-2)