

Applies to:

SAP NetWeaver Application Server Java, SAP NetWeaver Developer Studio

Summary

OpenID is a new technology framework to provide a decentralized single sign-on system on the Internet. This article takes a deep-dive into the OpenID technology and its integration with SAP NetWeaver in particular. You'll also have the opportunity to get your hands dirty implementing a JAAS login module to support OpenID-based authentication in the SAP NetWeaver Application Server Java. Along with this article, the complete source code of the solution is provided for download (1).

Author: Martin Raeppe

Company: SAP AG

Created on: June 2nd 2008

Author Bio



As a Standards Architect with SAP's Industry Standards team, Martin works in the area of standardization and interoperability testing of new Web Services technologies, focusing on security and identity management. Martin is a frequent speaker at SAP TechEd and the author of the SAP PRESS book "The Developer's Guide to SAP NetWeaver Security" ([german](#) | [english](#) edition).

Table of Contents

OpenID – A short primer	3
What is OpenID?	3
Who supports OpenID?	3
How does it work?	4
Step 1: User authenticates with the Relying Party web site	4
Step 2: Relying Party discovers OpenID Provider metadata	4
Step 3: The Relying Party establishes an association with the OpenID Provider	5
Step 4: Relying Party send authentication request to OpenID Provider	5
Step 5: Sending the Authentication Response to the Relying Party	6
Integrating OpenID with SAP NetWeaver	6
Functional Requirements	6
System Architecture	7
Implementation	8
Adding an OpenID login form field to the Logon Screen	9
Adding a custom attribute for the OpenID identifier to the User Profile	13
Implementing the OpenID Login Module	13
Deploying the OpenID Login Module	16
Configuring the OpenID Login Module	17
Testing the solution	18
Conclusion	21
Further Information	21
Related Content	22
Copyright	23

OpenID – A short primer

Very briefly, OpenID (2) is a way for Internet users to sign on to many different web sites with a single digital identity. This eliminates the need to keep a long list of user names and passwords and people can ‘single sign-on’ (SSO) across multiple OpenID-enabled web sites once they have been authenticated by an OpenID Provider of their choice, a service that manages their OpenID account. You might be wondering what’s really new about this approach – similar services like Microsoft Windows Live ID (formerly known as the Microsoft Passport system) have been around for quite a long time now. So what’s different about it and why should you invest some time reading this article and getting more familiar with OpenID?

What is OpenID?

From an end user’s perspective, OpenID seems to be just another SSO proposal for the Internet. However, there are some key characteristics that make OpenID different from existing digital identity technologies:

- OpenID is decentralized: As with any other SSO system, users must have previously registered for an account at a trusted authority, also known as an Identity Provider. With OpenID, this provider allows users to manage their central online identity in one place. However, compared to other SSO services on the Internet, there is not just *one* provider with *one* server infrastructure. With OpenID, users are in control of who they choose as their preferred identity provider. They can even decide to run their own provider. Thus, OpenID introduces a new concept of decentralization for SSO on the Internet both at the technical and at the organizational level.
- OpenID is truly open: OpenID is not a program owned by single company. It has started as a lightweight technology framework initiated by an Open Source community that does not want anyone to own it. To support further adoption of the OpenID technology by the industry, the OpenID Foundation (OIDF) was formed in June 2007. OIDF ensures that all specifications developed by the OpenID community are freely implementable. All contributors granted a copyright license to the Foundation to publish the specifications and signed a so-called ‘patent non-assertion agreement’ that states that the initial contributors will not sue anyone for implementing OpenID specifications.

Who supports OpenID?

There is more than one answer to this question. Since its initial launch in 2005, OpenID has been adopted by many providers, web sites, vendors and the Open Source community is pretty impressive:

- OpenID Providers: When you start with OpenID, you first have to decide who should be your OpenID Identity Provider. For registered users of large internet portals such as Yahoo or AOL, this decision can be fairly simple because their existing accounts are already pre-configured and can be easily enabled for OpenID with just a few clicks. There are also a number of well-known public OpenID Providers (3) such as *myOpenID.com* or *meinguter.name* (a German OpenID Provider) that issue OpenID identities and run the infrastructure to verify them.
- OpenID-enabled Web sites: As of June 2007 (3) there were approximately 4.500 Web sites that offered OpenID-based SSO to their users. With Yahoo’s recent announcement (4) to support OpenID, the number of Internet users with OpenIDs increased to a total of approximately one third of a billion (5).
- OpenID libraries and servers: To lower the entry barrier for web site owners and developers, a wealth of libraries (6) for many programming languages including Java, PHP and C/C++ are available to provide support for OpenID under an Open Source licensing model. The libraries basically hide all of the protocol details and aim to accelerate the integration of OpenID into a web site, either as a consumer or as a provider for OpenIDs. People who want to run their own OpenID Provider can also choose among a series of standalone OpenID server implementations (7).



Figure 1 OpenID Logo

How does it work?

Now it's time to take a closer look at the OpenID technology and what happens behind the scenes in a typical SSO scenario. In general, the OpenID protocol only uses standard HTTP(S) requests and responses that can be processed by any Web Browser or other client software. Assuming that the user has previously registered with an OpenID Provider (8) of his choice, the login process basically consists of five main steps as shown in the overview in Figure 2 Sequence Diagram of a typical login process with OpenID Figure 2.

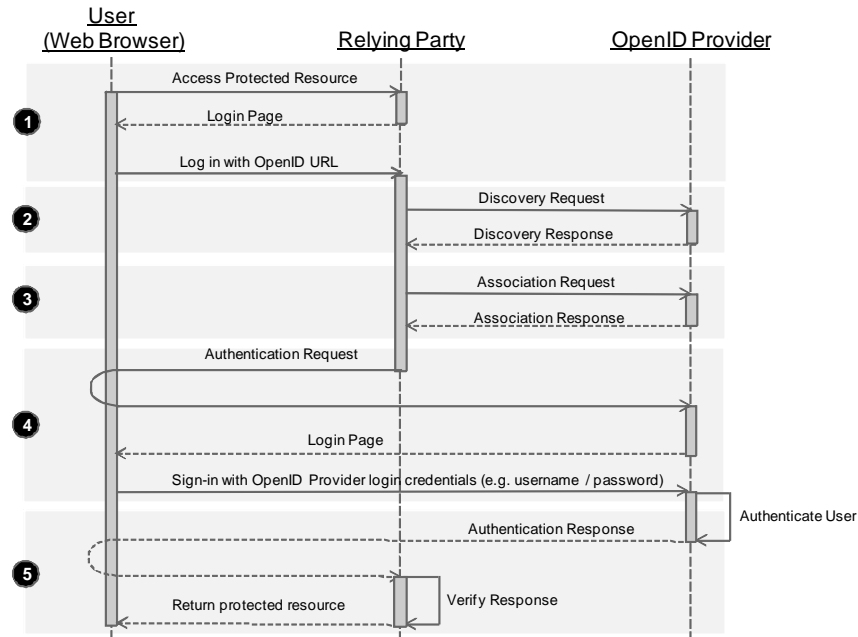


Figure 2 Sequence Diagram of a typical login process with OpenID

The following sections describe these steps in more detail.

Step 1: User authenticates with the Relying Party web site

The user visits the OpenID-enabled web site, also called the Relying Party in an SSO scenario. Instead of typing in a user name and password, the user only enters the *identifier* of his OpenID identity into a login form field that typically has a small OpenID logo (see Figure 3). An OpenID identifier is simply a URL that points to the user's OpenID Provider (e.g. <http://raepple.myopenid.com>).



Figure 3 OpenID Login Form Field

Step 2: Relying Party discovers OpenID Provider metadata

Next, the Relying Party must verify that the identifier belongs to the user who wants to get access to the protected resource. To do so, the Relying Party contacts the authentication service of the user's OpenID Provider. The process of looking up the necessary information for initiating this communication is also called *discovery*. The relying party sends an HTTP request to the OpenID identifier URL entered by the user in the login form field and parses the response. The OpenID Provider returns the user's OpenID Identity Page which contains an HTML link tag with the required URL (e.g. `http://www.myopenid.com/server`) in the href attribute (see Listing 1).

```
...
<link rel="openid.server" href="http://www.myopenid.com/server" />
...
```

Listing 1 OpenID Provider Metadata in the user's OpenID Identity Page hosted by the OpenID Provider

Note: With the latest release 2.0 of the OpenID specification, the Relying Party can also discover the authentication service location of the user's OpenID Provider by requesting an *eXtensible Resource Descriptor Sequence* (XRDS) document. XRDS (9) is a standardized XML format for discovery of metadata about a resource.

Step 3: The Relying Party establishes an association with the OpenID Provider

The Relying Party starts the conversation with the OpenID Provider to authenticate the user. To protect all subsequent protocol messages against eavesdropping and modification, the OpenID specification (10) recommends establishing a so-called *association session* between the OpenID Provider and the Relying Party. An association session is basically a shared secret between the two parties that is identified by a unique identifier, the *association handle*. The shared secret is used as a key to digitally sign the message content. To securely transmit it over an insecure communication channel like the Internet, the Relying Party can either choose transport layer encryption (e.g. SSL/TLS) or the Diffie-Hellman key exchange (11) to establish the association. The message flow between both parties is illustrated in Figure 4:

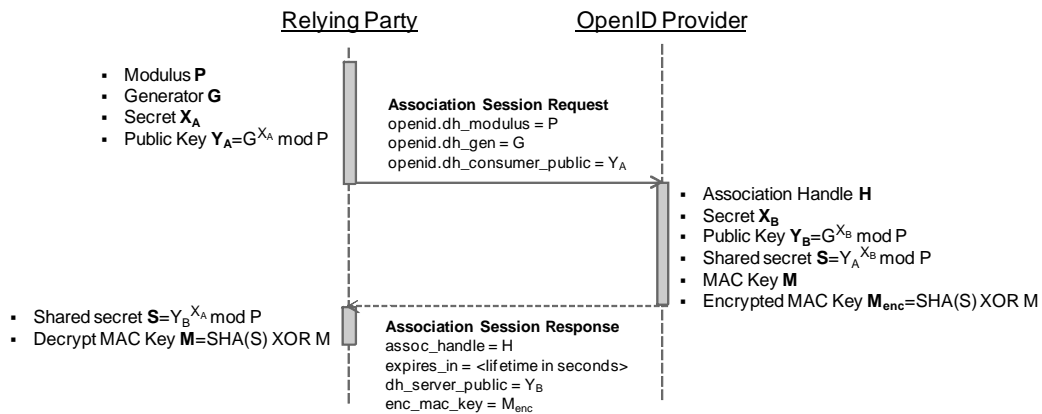


Figure 4 Sequence diagram of an OpenID message exchange to establish a Diffie-Hellman-based Association Session

The Relying Party starts the Diffie-Hellman key exchange by generating the modulus (prime) P and the generator G which it both sends in clear text to the OpenID Provider via the corresponding Association Session Request message parameters `openid.dh_modulus` and `openid.dh_gen`. Based on P , G and a secret integer X_A the Relying Party's public key (Y_A) is computed and also sent with the Request message. Upon reception of the message, the OpenID Provider generates the unique Association Handle H for the new association which both parties will use as a reference in subsequent messages. It also computes a public key Y_B based on its secret integer X_B . With the exchange of both public keys, both parties are now able to compute a shared secret S . The OpenID generates a Message Authentication Code (MAC, M) that is used later in the message exchange to sign the data. It encrypts M with the hash value (digest) of S and sends it along with its public key in the Association Session Response back to the Relying Party. The Relying Party is now also able to calculate S and decrypt M . The association's parameters are stored on each side and must only be used for the duration of its lifetime which is determined by the response parameter `expires_in`.

Note: It is also possible to establish an association session with a MAC sent in clear text with the association session response. Such *No-Encryption Association Sessions* do not incorporate a Diffie-Hellman key exchange but require message protection at the transport layer using protocols like SSL or TLS.

Step 4: Relying Party send authentication request to OpenID Provider

So far all messages have been sent directly between the Relying Party and the OpenID Provider, i.e. the user did not take any notice of the discovery and association establishment. Next, the Relying Party sends an authentication request to the OpenID Provider to authenticate the user based on his OpenID identifier. This time the request is not sent directly to the OpenID Provider; rather, the Relying Party redirects the

request via the user's web browser to the previously discovered OpenID Provider authentication service URL. This allows the OpenID Provider to

- present the user with a login screen if he has not yet authenticated at the central authentication service
- establishing an HTTP session once the user successfully signed-in e.g. via a cookie stored in the user's web browser for the length of the login session

The authentication request is simply an HTTP request (see Listing 2) sent to the OpenID Provider's authentication service that passes along with other data the user's OpenID identity (`openid.identity`) and the handle (`openid.assoc_handle`) of the previously established association as URL parameters. If the user has not yet authenticated to the OpenID Provider, he will have to supply the username and password for his centrally managed OpenID account via a login form. If the credentials are verified successfully, the user is logged in and a session is established.

```
http://www.myopenid.com/server?openid.assoc_handle=%7B HMAC -
SHA256%7D%7B483c970f%7D%7BqUr00g%3D%3D%7D&openid.identity=http%3A%2F%2Fraepple.myopen
id.com%2F&openid.return_to=http%3A%2F%2Fwww.plaxo.com%2Fopenid%3FactionType%3Dcomplet
e%26r%3D%252Fpulse%252F&...
```

Listing 2 Sample OpenID Authentication Request

The OpenID Provider will then ask whether the user trusts the Relying Party's web site URL (passed in the authentication request with the parameter `openid.return_to`) to receive the authentication response. Based on the user's decision to confirm or reject the Providers request to trust the Relying Party, the browser is redirected to the designated return URL. Most OpenID Providers allow their users to store their decisions about the trustworthiness of Relying Parties they frequently visit with their user account so that they are not asked a second time the same question.

Step 5: Sending the Authentication Response to the Relying Party

The authentication response sent by the OpenID Provider is also an HTTP request, passed through the User-Agent to the Relying Party. URL parameters with the response data are signed by the OpenID Provider using the shared secret (MAC) with a cryptographic hash function (HMAC) to protect the integrity of message. Based on the established association, the Relying Party can validate the signature received with the MAC previously stored and thus verify that the response really came from the OpenID Provider.

If this last step of the process has completed successfully, the Relying Party will most likely want to assign a local user account to the authenticated user based on the verified OpenID identifier. Therefore it will need a registry to look up local user account by their OpenID identifier. Please note that this mechanism is not in scope of the OpenID protocol and thus is also not part of the specification. Nevertheless, an implementation of an OpenID authentication module for a Relying Party such as the one presented in this article must deal with this problem. How this can be solved for SAP NetWeaver is described as part of the solution presented in the following chapter.

Integrating OpenID with SAP NetWeaver

The solution presented in this article proposes an integration approach to support OpenID-based authentication for users accessing applications deployed on the SAP NetWeaver Application Server Java, including the SAP NetWeaver Portal. In other words, the implementation in this chapter will solely focus on the functionality required by the *Relying Party* in the OpenID authentication process (see Figure 2). On the OpenID Provider side, we'll use an existing public provider to test the authentication process end-to-end with our solution.

Functional Requirements

Before we roll-up our sleeves and jump right into the coding, let's take a minute and think about the key functional requirements of an OpenID-based authentication enhancement for SAP NetWeaver:

- The solution should nicely fit into the overall authentication framework of the SAP NetWeaver Application Server Java, i.e. it should be pluggable and configurable in the same way as any other authentication mechanism such as certificate-based sign-on
- The solution should still allow users to authenticate with their conventional logon credentials such as username and password. OpenID-based authentication should be offered to the user on the logon screen as an alternative mechanism for SSO (just like certificate-based authentication)
- If users choose OpenID to sign-on for the first time on the Application Server, they must be able to assign the OpenID identifier to their existing account in the User Management Engine (UME). With this users are still required to initially register an account in the UME but do not need to remember their username and password once they successfully assigned their OpenID identifier to it. The process of linking both accounts is also called *account federation*.
- Users should be given the opportunity to unlink or *de-federate* their OpenID- and local UME-accounts

System Architecture

Following the above requirements, the main component of the OpenID solution is a login module based on the *Java Authentication and Authorization Service* (JAAS, see box below) (12). The block diagram in Figure 5 depicts the main components of the OpenID solution running on the SAP NetWeaver Application Server Java, illustrates the surrounding system components and the communication and data exchange between them.

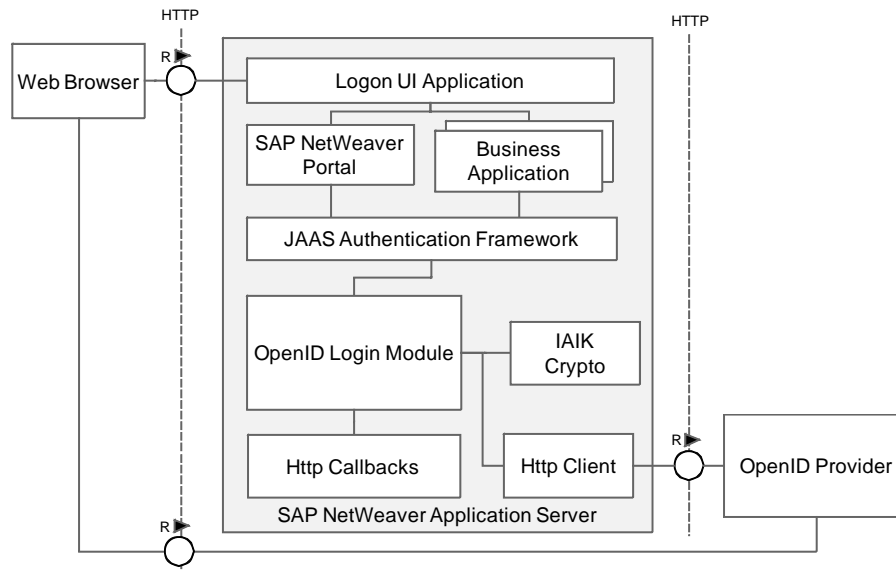


Figure 5 Block Diagram of the OpenID solution architecture

The roles of the components in the diagram are as follows:

- **Web Browser:** The user accesses the Portal or business application running on the SAP NetWeaver Application Server through a regular web browser interface and protocols (HTTP/S)
- **Logon UI Application:** The SAP NetWeaver Application Server Java ships with a standard set of logon screens which are bundled into the logon user interface (UI) application *com.sap~tc~sec~ume~logon~ui*. The application includes for example, the logon screen and the password help screen.
- **SAP NetWeaver Portal & Business Application:** The user either accesses the SAP NetWeaver Portal or a JEE-based Business Application deployed on the Application Server. Both require the user to authenticate at the server by entering his username and password or – as the outcome of this exercise – with his OpenID identifier.

- **JAAS Authentication Framework:** The JAAS Authentication Framework is responsible to control the login procedure on the Application Server and invoke the OpenID Login Module (see below) as part of this process.
- **OpenID Login Module:** The OpenID Login Module is the core of the solution. It implements the JAAS API and is added to the default login module stack ('ticket') used by the SAP NetWeaver Portal and other Business Applications to authenticate the user. It orchestrates the message flow of the OpenID login process (see Figure 2) between the Web Browser, the Application Server (Relying Party) and the OpenID Provider chosen by the user.
- **HTTP Callbacks:** These SAP-specific HTTP Callback classes are used to control the communication via HTTP between the user's Web Browser and the OpenID Login Module. The *HttpGetterCallback* is used by the login module to obtain information from HTTP requests, e.g. from the Authentication Response received from the OpenID Provider. In addition, the *HttpSetterCallback* is used in general to attach information gained in the login module to the response. The OpenID Login Module uses it to redirect the user's Web Browser to the OpenID Provider as described in steps 4 and 5 of Figure 2.
- **IAIK Crypto:** The security library from the Institute for Applied Information Processing and Communication (IAIK) is shipped with the SAP NetWeaver Application Server Java. It is used by the OpenID Login Module to process the cryptographic functions when establishing the association session with the OpenID Provider and verifying the signature of the incoming authentication response. In particular, IAIK provides an implementation of the Diffie-Hellman key exchange algorithm and hash function as defined by the OpenID specification (10) for the protocol.
- **Http Client:** The Http Client allows applications deployed on the SAP NetWeaver Application Server to issue server-side HTTP requests and retrieving data from external sites on the Internet. The OpenID Module uses this SAP-specific component to support protocol messages that are exchanged directly between the Relying Party and the OpenID Provider, e.g. during the discovery and association establishment phases. The Http Client offers a broad range of features, including the use of a HTTP proxy, e.g. if the Relying Party (SAP NetWeaver Application Server) is behind a firewall. This is one of the reasons why none of the existing Java libraries for OpenID were used because they do not support this configuration option.
- **OpenID Provider:** This is the OpenID Provider chosen by the user. It must support the final OpenID 2.0 specification (see also *Current Limitations* on page 21).

JAAS: As of J2SE Version 1.4 and J2EE Version 1.3, the *Java Authentication and Authorization Service* (JAAS) is an integral part of the APIs in both platforms. From a developer's point of view, the main advantage of JAAS is the possibility to control the authentication process in the application server, which means the login process can be customized to suit specific requirements. Login modules are the actual authentication components in JAAS. They implement a uniform interface, the JAAS service provider interface (SPI). Because the interface is formulated independently of the technology, the modules can be exchanged easily. For example, if an application verifies a user's identity based on his user name and password, the authentication method can easily be changed to certificate-based authentication by replacing the corresponding login module with another one. This does not require any major changes in the program code. In addition to the many module-specific configuration options provided for controlling the internal functions of the modules, the modules can also be grouped into login module stacks to implement more flexible and at the same time more comprehensive login procedures. For more information on JAAS and how to use it in the SAP NetWeaver platform, please see (12).

Implementation

Now it's time to get our hands dirty. This section will lead you through the implementation of the solution following these four steps:

- Step 1: Adding an OpenID login form field to the Logon Screen
- Step 2: Adding a custom attribute for the OpenID identifier to the UME user profile
- Step 3: Implementing the OpenID Login Module
- Step 4: Deploying the OpenID Login Module
- Step 5: Configuring the OpenID Login Module
- Step 6: Testing the solution

Adding an OpenID login form field to the Logon Screen

To support OpenID-based SSO on the default logon screen of the SAP NetWeaver Application Server Java, the user must obviously be able to enter his OpenID identifier in a new field as shown in Figure 3. Most OpenID-enabled web sites on the Internet add a link with a text similar to “Sign-in with OpenID” to their existing logon page which point to a new page for OpenID logon only. To keep things simple here, we’ll just enhance the existing logon screen by an additional entry field for the user’s OpenID identifier. In addition this screen will also be used to allow the user to link (federate) his local and remote account if he signs-on with OpenID for the first time. Please follow these steps to customize the logon screen accordingly:

1. Start your SAP NetWeaver Developer Studio (NWDS) and choose **File • Import**
2. Select **Web • WAR** in the Import dialog and click on **Next**
3. Browse for the Web Archive file *sap.com~tc~sec~ume~logon~ui.war* of the Logon UI Application from a local copy or on your Application Server’s installation directory (e.g. `\usr\sap\<SID>\JC<XX>`) under the path `\j2ee\cluster\apps\sap.com\com.sap.security.core.logon\servlet_jsp\logon_ui_resources` and accept the default settings by clicking on **Finish** (see Figure 6).

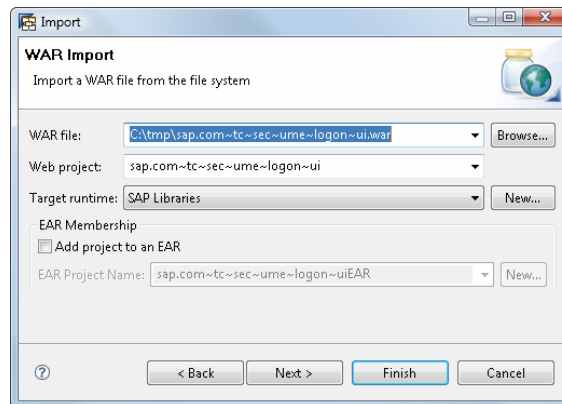


Figure 6 Import of the Logon UI Application WAR File into the Workspace

4. Confirm the dialog box and switch to the **Java EE perspective**
5. Copy the image file *openid.gif* from the code archive root directory to the project directory `/WebContent/layout`
6. Open the file *logonPage.jsp* in the project directory `/WebContent` with the JSP editor
7. Replace the existing page with the new logon page from the code archive (1) which can be found in the directory `/Code/sap.com~tc~sec~ume~logon~ui/WebContent/logonPage.jsp`.

Note: There are only a few changes applied to the default logon page which are marked by the comments “ADDED FOR OPENID LOGIN – BEGIN” and “...– END” in the new file. Besides a new input field for entering OpenID identifier, the additional Java code in the JSP primarily enables the user to link a previously verified OpenID identifier with his local account. In this case, the OpenID Login Module redirects the user back to the logon page again with two additional URL parameters: `openid.verifiedOpenIdUrl`, which contains the approved identifier by the OpenID provider, and `openid.federate`. If the latter is set to “true” by the OpenID Login Module, an additional submit button with the label “Federate” and a short help text is displayed on the screen.

8. Create a new EAR Application Project which will be used to deploy the new Logon UI Application
 - a. Start the EAR Application Project wizard with **File • New • Project...** and select **Enterprise Application Project** from the **J2EE** folder.
 - b. Enter a project name (e.g. “OpenIDLogonUI”) and click on **Next**
 - c. Keep the default settings in the **Project Facets** dialog and click on **Next**
 - d. Select the **sap.com~tc~sec~ume~logon~ui** project from the list of modules and click on **Finish**

- In the new EAR project, open the SAP-specific Deployment Descriptor file *application-j2ee-engine.xml* from the */EarContent/META-INF/* directory and add the `<reference>` and `<provider-name>` elements (see Listing 3):

```
<application-j2ee-engine ...>
  <reference reference-type="hard">
    <reference-target provider-name="sap.com" target-type="application">
      com.sap.security.core.logon
    </reference-target>
  </reference>
  <provider-name>openid.com</provider-name>
</application-j2ee-engine>
```

Listing 3 SAP EAR Deployment Descriptor for the customized Logon UI Application

The provider name chosen here ("openid.com") defines the namespace where the applications will reside on the runtime after deployment. In this case, the new Logon UI Application deploys to the path `\j2ee\cluster\apps\openid.com\<project_name>`. The reference points to the existing application that implements the business logic of the Logon UI.

- Add the standard JEE Deployment Descriptor *application.xml* by selecting **New • File** in the context menu of the */EarContent/META-INF/* directory. Change the content of the file as shown in Listing 4.

```
<?xml version="1.0" encoding="UTF-8"?>
<application id="Application_ID" version="5"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/application_5.xsd">
  <description/>
  <display-name>OpenId logon</display-name>
  <icon/>
  <module
    id="WebModule_1183039485065">
    <web>
      <web-uri>sap.com~tc~sec~ume~logon~ui.war</web-uri>
      <context-root>openid_logon</context-root>
    </web>
  </module>
</application>
```

Listing 4 JEE EAR Deployment Descriptor for the customized Logon UI Application

Note that the alias "openid_logon" in the `<context-root>` element defines the new URL for the customized Logon UI application.

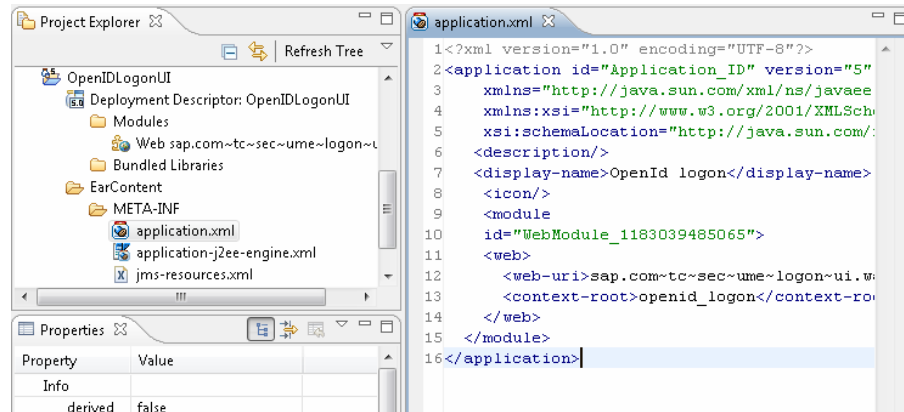


Figure 7 EAR project structure

- Export the Enterprise Application project to the local file system by selecting **Export • SAP EAR File** from the context menu of the OpenIDLogonUI project. Accepts the default settings and click on **Finish**.

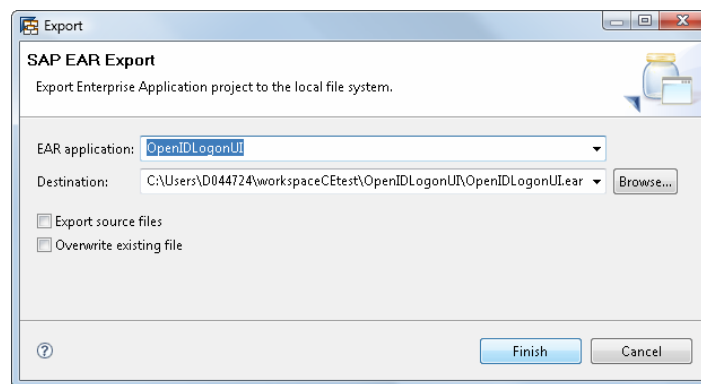


Figure 8 Export SAP EAR File

- Prepare the deployment the new Logon UI Application EAR file by switching to the Deploy perspective (**Window • Open Perspective • Deploy**). In the **Deploy View**, select **External Deployable Archives** in the list and click on the **Add element** button (see Figure 9). Browse to your local SAP NetWeaver Developer Studio Workspace directory and select the *OpenIDLogonUI.ear* file from the *OpenIDLogonUI* project directory that you created in the previous step.

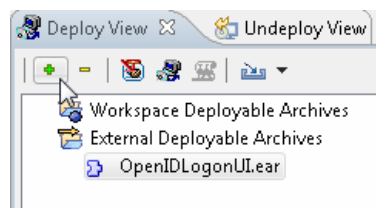



Figure 9 Deploying the Logon UI EAR File

- Click on the **Deploy** () button to start the deployment. The deployment log in the **Deploy View Console** must include a status message that states successful completion as shown in Figure 10.

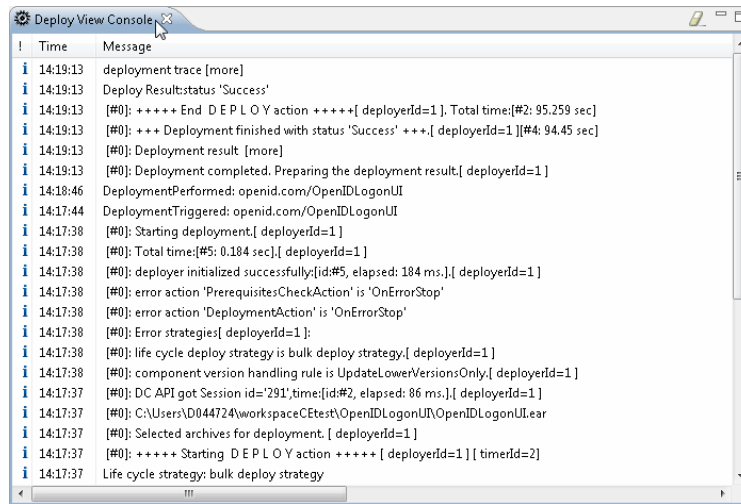



Figure 10 Deployment trace messages after successful deployment of the new Logon UI Application EAR file

14. Use the **Config Tool** to configure the new URL alias of the customized Logon UI Application in the UME properties:
 - a. Stop the Application Server with the SAP Management Console
 - b. Start the Config Tool with the batch file *configtool.bat* from your Application Server's subdirectory `\j2ee\configtool\`
 - c. Switch to the configuration edit mode ()
 - d. In the **Display Configuration** tab, go to `cluster_config • system • custom_global • cfg • services • com.sap.security.core.ume.service • PropertySheet properties`

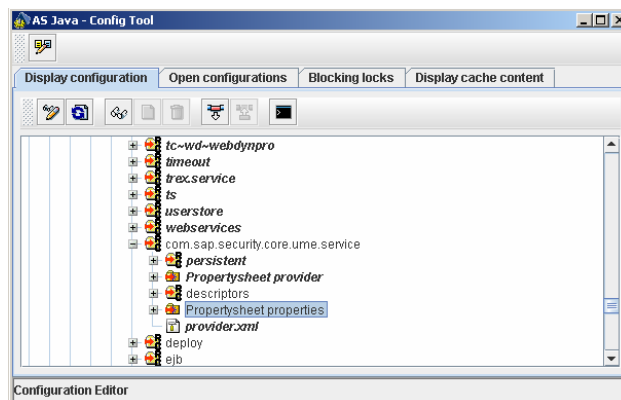


Figure 11 UME Properties in the Config Tool

- e. Switch to the **Edit mode** by clicking on the pencil symbol
- f. Open the *PropertySheet properties* and change the default value of the property entry `ume.logon.application.ui_resource_alias` to `/openid_logon` (see Figure 12).

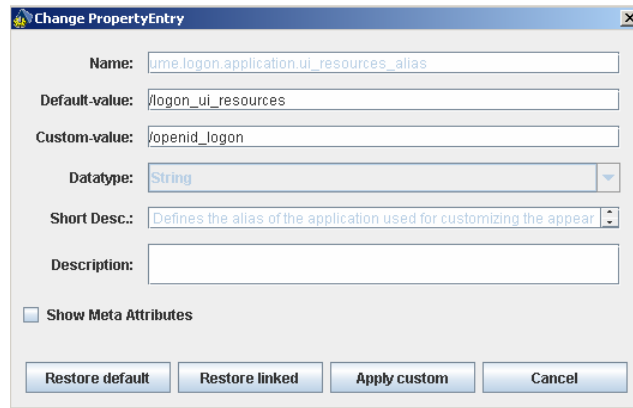


Figure 12 Changing the default Logon UI Application URL alias

- g. Click on the **Apply custom** button to store your settings and then on **Ok**
- h. Close the Config Tool
15. Restart the Application Server with the SAP Management Console

Adding a custom attribute for the OpenID identifier to the User Profile

This step prepares the User Management Engine to support the requirement for linking OpenID accounts with local UME account. Instead of deploying a separate database table to persist this relationship, the approach here is simply to create a custom attribute in the UME user profile and map it to the OpenID identifier of the user.

1. Start the Identity Management application (*http://<hostname>:<port>/useradmin*) and log in with the Administrator user
2. Click on the **User Management Configuration** button (see Figure 13)

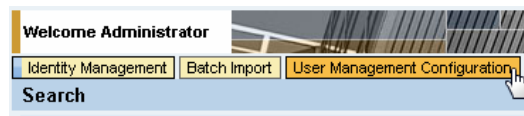


Figure 13 Start User Management Configuration

3. Click on the **Modify Configuration** button
4. Select the **User Admin UI** tab
5. Go to the section **Custom attributes of the user profile**. Follow the general syntax *<namespace>:<attribute>* for custom attributes and enter the value like "openidns:openidurl" in the entry field with the label **User-Managed Custom Attributes** (see Figure 14)

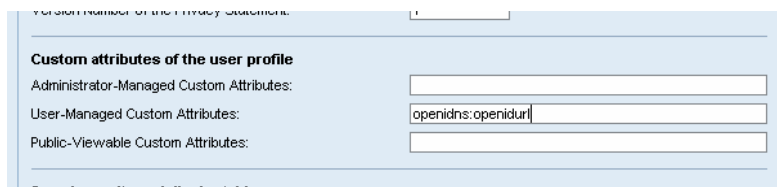


Figure 14 Custom Attribute for the OpenID Identifier

6. Click on **Save all changes** and log off.
7. Restart the Application Server to take the changes effect.

Implementing the OpenID Login Module

Following the successful customization of the Logon UI Application and the UME user profile, in this step we'll develop and deploy a new login module for OpenID authentication.

1. Create a new project for the login module based on the content from the code archive (1):
 - a. In SAP NetWeaver Developer Studio, select **File • New • Project ...**

- b. Select **Java Project** from the list
- c. Enter the project name "OpenIDLoginModule" in the wizard and click on **Finish**
- d. Right-click on the new project and select **Import ...** from the context menu
- e. Choose **General • File System** from the list sources
- f. Select the `/Code/OpenIDLoginModule` directory from the code archive (1) and include all of its content for the import as shown in Figure 15 Import initial project content from the code archiveFigure 15.

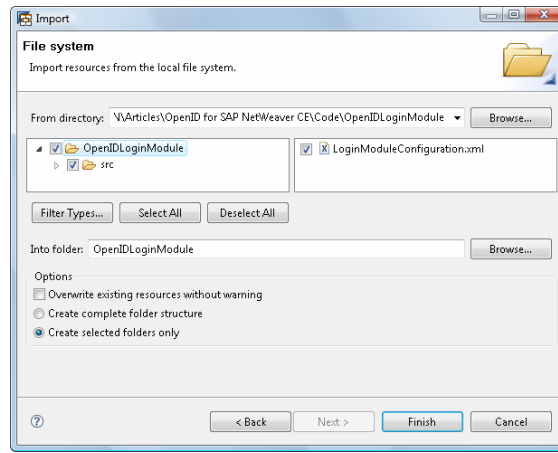


Figure 15 Import initial project content from the code archive

2. To implement and compile the login module we have to set additional libraries in the build path of the project:
 - a. Select the **OpenIDLoginModule** project and choose **Build Path • Add External Archives** from the context menu
 - b. Select the JAR file `sap.com~tc~bl~jkernel_util~impl.jar` from your Application Server installation directory following the path `\usr\sap<SID>\JC<XX>\j2ee\cluster\bin\system`
 - c. Repeat these two steps for the following libraries:

Library (JAR file)	Application Server Directory
<code>sap.com~tc~je~security_api~impl.jar</code>	<code>\usr\sap<SID>\JC<XX>\j2ee\cluster\bin\interfaces\security_api</code>
<code>sap.com~tc~sec~ume~api~impl.jar</code>	<code>\usr\sap<SID>\JC<XX>\j2ee\cluster\bin\ext\com.sap.security.api.sda</code>
<code>sap.com~httpclient.jar</code>	<code>\usr\sap<SID>\JC<XX>\j2ee\cluster\bin\ext\httpclient</code>

- d. Now select **Build Path • Add Libraries** and choose **SAP Java EE Libraries Server Runtime** from the list. Click on **Next** and select **SAP Libraries** before closing the wizard with **Finish**.

The project should now compile without any errors. Before we continue with the deployment of the login module we'll take a close look at the implementation.

`OpenIDLoginModuleClass` is the central class of the login module. It implements the JAAS interface `javax.security.auth.spi.LoginModule` which includes the `login()` method where the process flow of the OpenID authentication at the Relying Party is controlled by the login module. As mentioned earlier, the OpenID login module can be configured to use an HTTP proxy for the communication with the OpenID Provider by two configuration options: `ProxyHost` and `ProxyPort`. Both can be set by the security administrator in the SAP NetWeaver Administrator (NWA, see page 16) and are read by the module during initialization (see method `initialize()`).

The login module is actually called at least two times by the JAAS authentication framework in the course of an OpenID authentication. The `login()` method gets invoked when

- a. the user logs on by entering his OpenID identifier (instead of a username and a password) in the new Logon UI form field
- b. the OpenID Provider sends the authentication response upon an earlier authentication request from the login module

In both cases, the `login()` method first retrieves the URL parameters from the current HTTP request using the `HttpGetterCallback` callback class. If it finds an OpenID identifier in a parameter named `openid_url`, the login module starts to discover the OpenID Provider's metadata by calling the method `discoverOpenIDProvider()` of the `OpenIDLoginModuleClass` class which uses an instance of `HttpClient` to communicate directly with the OpenID Provider. Next, the login module establishes an association implemented by the `DHAssociationSession` class and sends the authentication request via the user's web browser to the OpenID Provider's service. Sending the redirect in the `sendAuthenticationRequest()` method is achieved using the SAP-provided `HttpSetterCallback` class which sets the correct location and HTTP response code (302) as shown in Listing 5.

```
HttpSetterCallback setRCodeCB = new HttpSetterCallback();
setRCodeCB.setType(HttpCallback.RESPONSE_CODE);
setRCodeCB.setName("Response Code");
setRCodeCB.setValue("302");

HttpSetterCallback setRedirCB = new HttpSetterCallback();
setRedirCB.setType(HttpCallback.HEADER);
setRedirCB.setName("Location");
setRedirCB.setValue(redirectUrl);

Callback[] cbSetter = new Callback[2];
cbSetter[0] = setRCodeCB;
cbSetter[1] = setRedirCB;
callbackHandler.handle(cbSetter);
```

Listing 5 Using the `HttpSetterCallback` class to send a redirect from a login module

If the authentication response from the OpenID Provider is received as an HTTP request, the `login()` method verifies the signature of the request parameters by calling `verifyAuthenticationResponse()`. To do so, it first extracts the signed URL parameters from the request and then checks for a replay attack by looking up the nonce value received with the response in its local nonce cache. If it is not found, the shared secret (MAC) from the association is used to calculate the signature over the signed request parameters and compare it with the OpenID provider's signature in the `openid.sig` parameter from the authentication response. If both values are equal, the message and the user's OpenID identifier can be considered valid.

To finalize the authentication process, the login module searches for a local account that is associated with the verified OpenID identifier in the `verifyIdentity()` method. As shown in Listing 6, the UME class `UserSearchFilter` can be used to search in the custom attribute "openidns:openidurl" of the user profile:

```
IUserFactory userFactory = UMFactory.getUserFactory();
IUserSearchFilter usf = userFactory.getUserSearchFilter();
usf.setSearchAttribute(Constants.UME_OPENID_ATTRIBUTE_NS,
    Constants.UME_OPENID_ATTRIBUTE_NAME, openidAuthData.getVerifiedOpenIDIdentity(),
    ISearchAttribute.EQUALS_OPERATOR, false);
ISearchResult searchResult = userFactory.searchUsers(usf);
```

Listing 6 Searching for a user account based on the OpenID Identifier in the custom attribute

If the search is successful, the unique id of the authenticated user account is returned and the subject is populated with an `OpenIDPrincipal` instance in the commit phase of the JAAS authentication process.

An empty search result set means that the verified OpenID identifier has not yet been linked to a local account. In this case, the user should be able to federate his accounts. Therefore, the OpenID login module redirects the user's web browser to the initially requested URL. It also adds the `openid.verifiedOpenIdUrl` and `openid.federate` parameters to the request so that the logon page will show the previously verified OpenID identifier (as a read-only input field) and a "Federate" button. By entering a valid username and password and clicking on the "Federate" button, the user links the OpenID identifier to his local account. This is a one-time action for all users and causes a third invocation of the `login()` method that will call the `federateOpenIDAccount()` method to store the OpenID identifier in the custom attribute of the user's account.

Deploying the OpenID Login Module

The SAP-specific deployment configuration of the OpenID Login Module is stored in the file `LoginModuleConfiguration.xml` (see Listing 7) which can be found in the root directory of the project.

```
<login-modules>
  <login-module>
    <display-name>OpenIDLoginModule</display-name>
    <class-name>com.sap.openid.OpenIDLoginModuleClass</class-name>
    <description>OpenID 2.0 Consumer Login Module</description>
    <options>
      <option>
        <name>ProxyHost</name>
        <value> </value>
      </option>
      <option>
        <name>ProxyPort</name>
        <value> </value>
      </option>
    </options>
  </login-module>
</login-modules>
```

Listing 7 OpenID Login Module Deployment Descriptor

The file is used to automatically register a deployed login module in the user store of the Application Server Java. It must contain a unique display name (`<display-name>`), the fully qualified class name (`<class-name>`) and zero to n options (`<option>`) of the login module. For the OpenID login module, these are the `ProxyHost` and `ProxyPort` options to configure the HTTP proxy at runtime.

Follow these steps to deploy the OpenID login module:

1. Create a new EAR Application Project
 - a. Start the EAR Application Project wizard with **File • New • Project...** and select **Enterprise Application Project** from the **J2EE** folder.
 - b. Enter a project name (e.g. "OpenIDLoginModuleEAR") and click on **Next**
 - c. Keep the default settings in the **Project Facets** dialog and click on **Next**
 - d. Select the **OpenIDLoginModule** project from the list (see Figure 16) of modules and click on **Finish**

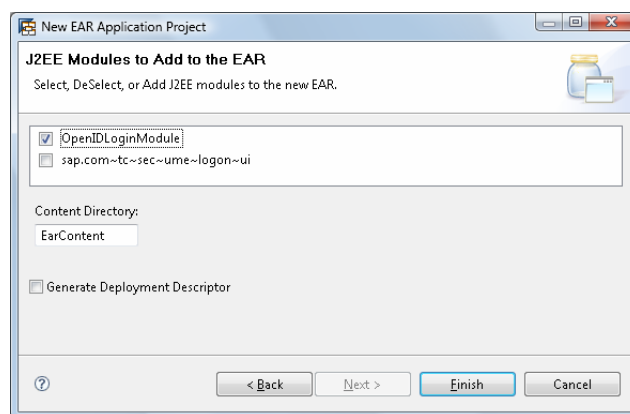


Figure 16 Adding the OpenIDLoginModule project to the EAR file

2. To provide the `HttpClient` library to the EAR project's classloader, select the **Bundled Libraries** folder, and choose **Add/Remove** in the context menu (see Figure 17)

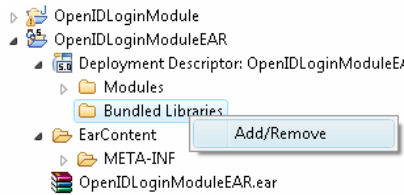


Figure 17 Bundling the HttpClient library with the EAR file

3. Click on the Add External JARs button to add the *sap.com~httpClient.jar* file (e.g. from the Application Server installation directory *usr\sap\<SID>\JC<XX>\j2ee\cluster\bin\ext\httpClient*)
4. Export the Enterprise Application project to the local file system by selecting **Export • SAP EAR File** from the context menu of the **OpenIDLoginModuleEAR** project. Accepts the default settings and click on **Finish**.
5. Open the new *OpenIDLoginModuleEAR.ear* file with an archiving program. Usually this works right out of the box by simply double-clicking the file in the Workbench.
6. Put the *LoginModuleConfiguration.xml* via Drag & Drop from the **OpenIDLoginModule** project in the EAR file (see Figure 18). The login module is now ready for deployment.

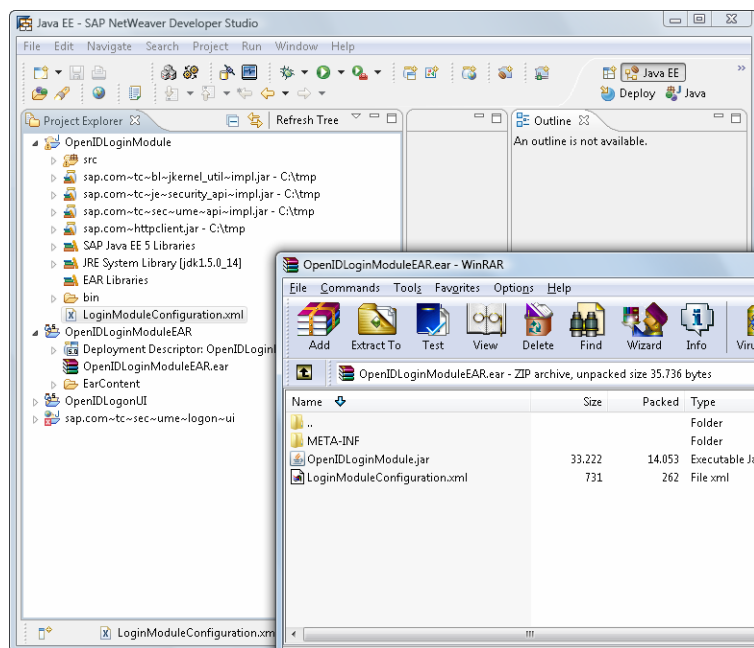



Figure 18 Adding the Configuration File to the EAR File

7. Switch to the **Deploy** perspective. In the **Deploy View**, select **External Deployable Archives** in the list and click on the **Add element** button (see Figure 9). Browse to your local SAP NetWeaver Developer Studio Workspace directory and select the *OpenIDLoginModuleEAR.ear* file from the *OpenIDLoginModuleEAR* project directory.
8. Click on the **Deploy** () button to start the deployment.

Configuring the OpenID Login Module

Before testing the new login module it must be configured for the “ticket” login module stack on the Application Server Java:

1. Start the **SAP NetWeaver Administrator** with your web browser (*http://<hostname>:<port>/hwa*) and log in as the Administrator. Although the new OpenID login form field is already visible on the logon page, it has no effect because the Login Module is not yet activated
2. Switch to the tab **Configuration Management** and select **Authentication** from the **Security** page.

3. In the **List of Policy Configurations**, enter the filter term “ticket” in the entry field and hit return.
4. From the result list, select the entry with the name “ticket”. This will update the list of login modules assigned to the *ticket* stack in the **Policy Configuration Details** section
5. Switch to edit mode by clicking on the **Edit** button
6. To add the OpenID Login Module to the stack, click the **Add** button
7. From the drop-down list, select the **OpenIDLoginModule** entry and click on **Add**
8. Move the **OpenIDLoginModule** to the third position in the stack by clicking on the **Move up** button
9. Optionally, you can enter a value for an HTTP proxy hostname and port (e.g. “proxy” and “8080”) in the **Login Module Options** section
10. Save your changes to the login module stack “ticket” by clicking on the **Save** button. The result is shown in Figure 19.

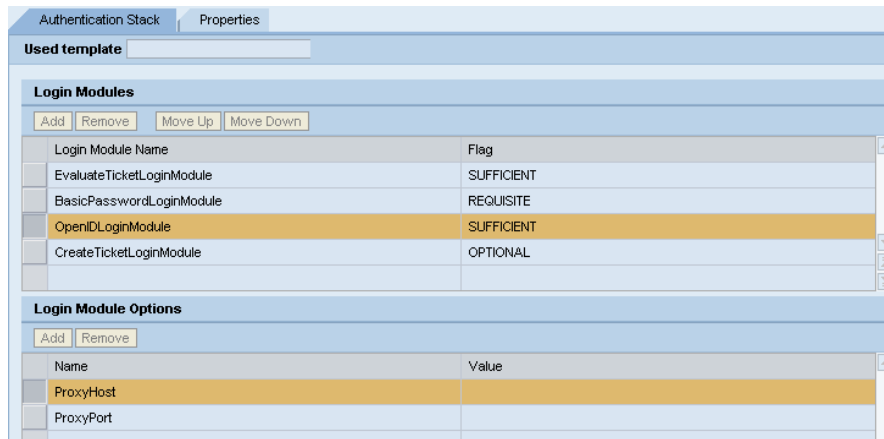
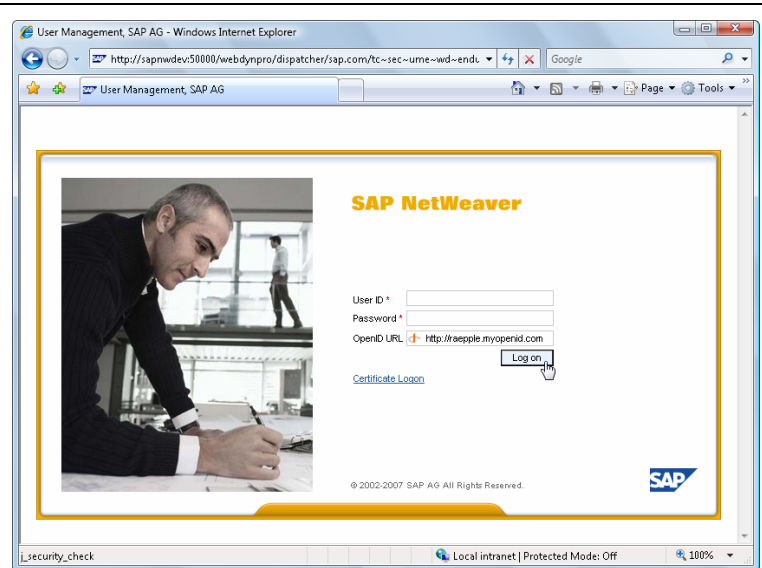


Figure 19 Customized ticket Authentication Stack for OpenID authentication

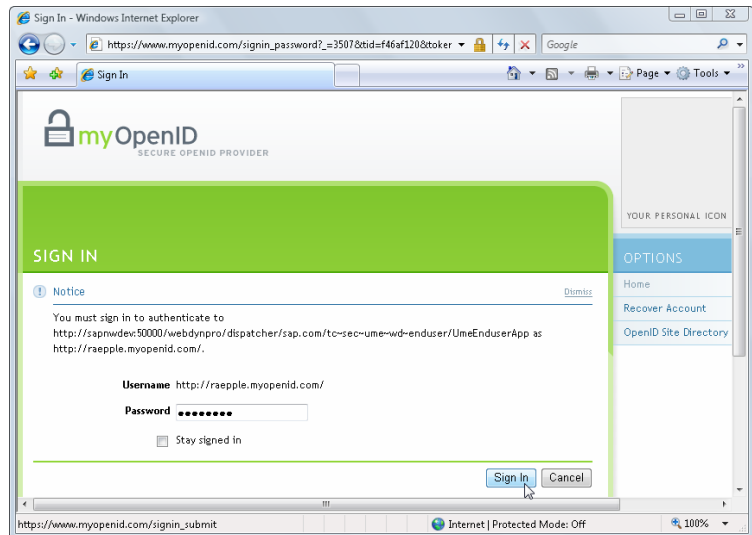
Testing the solution

A prerequisite for testing the OpenID login module is an OpenID account. If you don't have an account at one of the well-known public OpenID Providers, just pick one from the list at (8) and sign up for your own OpenID identifier. This process usually takes less than five minutes. The test case executed in this section uses the author's OpenID identifier <http://raapple.myopenid.com> and includes the following steps:

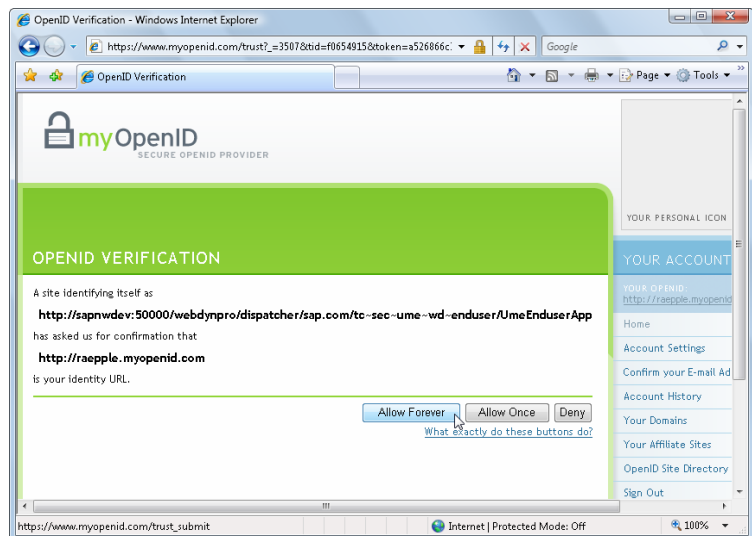
1. The user opens the URL `http://<hostname>:<port>/webdynpro/dispatcher/sap.com/tc~sec~ume~wd~enduser/UmeEnduserApp` of the UME user profile management application resource on the Application Server and is prompted to logon. Instead of using his username and password, the user enters his OpenID identifier in the new logon page form field.



2. The OpenID Login Module discovers the OpenID Provider's metadata, establishes the association and sends the authentication request. Since the user is not yet signed in at the OpenID provider, he is prompted with his provider's logon page.



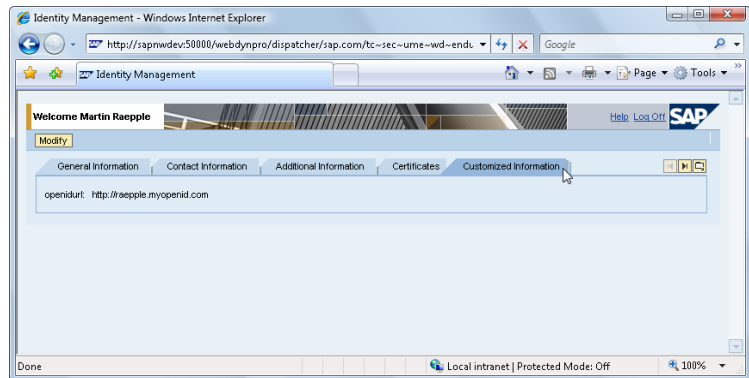
3. After successful log in at the OpenID Provider, the user must confirm that the OpenID Provider is allowed to send back the authentication response to the Relying Party. He can allow the OpenID provider to automatically confirm the user's OpenID identifier to the corresponding Relying Party in the future. By clicking on the **Allow Forever** button, the OpenID Provider will not ask the user the same question again.



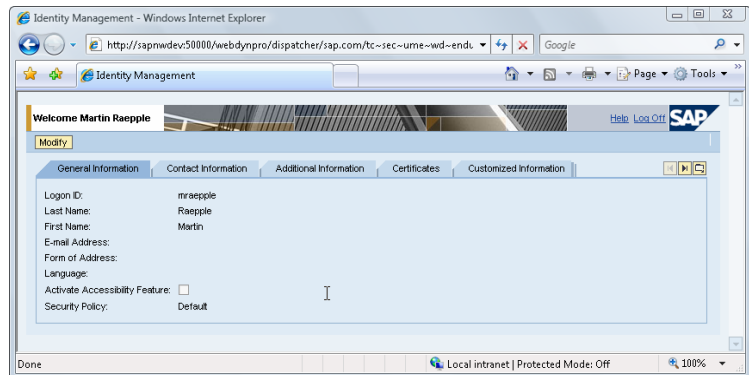
4. The successfully verified OpenID identifier is sent back to the Relying Party with the authentication response. Although the provider's signature has been verified successfully, the Relying Party cannot find an account which is linked to the entered and verified OpenID identifier. Therefore the user now has the choice to federate his accounts by providing the username and password of an existing local account and clicking on the **Federate** button.



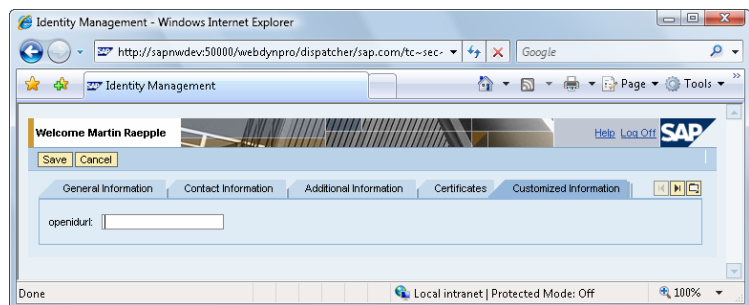
5. The user is now successfully authenticated and redirected to the user profile management application. By switching to the **Customized Information** tab, the user can view and modify the account linking information in the custom UME attribute *openidurl*.



6. Log off and log on again by entering only your OpenID identifier in the new logon page. This time the user is neither asked to confirm the authentication response at the OpenID provider, nor is there any longer the need to federate his identifier with the local account. As a result, the user is simply authenticated via SSO without being asked for a password again.



7. Let's try to de-federate the accounts: Switch to the **Customized Information** tab again and click on the **Modify** button. Delete the OpenID identifier stored by the OpenID Login Module in the *openidurl* entry field and click on **Save**.



8. Log off from the application and log on again with your OpenID identifier on the logon page. Since you are still logged in at the OpenID Provider, you are now only asked to federate your accounts again. Enter your local account credentials (username/password) and click on the **Federate** button to map your id once again.



Conclusion

OpenID seems to be a promising approach to solve the long-standing issue of Single Sign-On on the Internet. For Internet-facing portals based on the SAP NetWeaver Portal, the solution presented in this article can serve as a starting point to provide SSO capabilities based on the OpenID technology. However, the current status of the implementation is a first proof-of-concept and still leaves room for improvements. Here are some ideas for further enhancements:

- The extension made in the UME user profile only allows a user to assign one OpenID account to its UME account. Since users can have multiple OpenIDs, a many-to-one relationship would be required to support this requirement
- The login module only supports HTML-based discovery of the OpenID Provider's metadata. The OpenID 2.0 specification also recommends supporting two additional discovery mechanisms, namely XRI Resolution (9) and the YADIS protocol (13).
- The specification defines four simple rules for normalizing end user's input into a valid OpenID Identifier which are currently not implemented
- The login module only works with OpenID Providers implementing the latest version 2.0 of the OpenID specification. Older versions of the protocol are currently not supported
- Further testing with OpenID Providers other than the one used in the previous section (*myopenid.com*) still needs to be done



Further Information

More information and examples on JAAS and other security technologies related to SAP NetWeaver can be found in the SAP PRESS book "The Developer's Guide to SAP NetWeaver Security". As a practical guide for developers, system integrators, and software architects, the book covers all security technologies in conjunction with SAP NetWeaver Application Server up to and including Release 7.0. In addition to describing the basic principles of the different technologies (Web Services Security, Single Sign-On, Identity Management etc.), the book focuses on providing practical exercises and examples that help you gain a profound understanding of the technologies and standards used, enabling you to better assess their intended purpose and apply them in your projects. The book serves as an A-to-Z reference book that enables

you to use - and benefit from - open security standards that are based on an enterprise service-oriented architecture (Enterprise SOA).

Related Content

1. Source Code archive of the OpenID login module. [Online]
<http://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/304016d3-7f19-2b10-a986-cb7028f43285>.
2. OpenID Homepage. [Online] <http://www.openid.net>.
3. **Kveton, Scott.** The State of OpenID. [Online] 2007. <http://openid.net/pres/openid-solt-final.pdf>.
4. Yahoo! Announces Support for OpenID. [Online] January 17, 2008.
<http://yhoo.client.shareholder.com/press/releasedetail.cfm?ReleaseID=287698>.
5. **Dash, Anil.** OpenID: The Next Quarter-Billion Identities. [Online]
http://www.sixapart.com/blog/2008/01/openid_the_next.html.
6. Libraries. *OpenID Wiki*. [Online] <http://wiki.openid.net/Libraries>.
7. Run your own identity server. *OpenID Wiki*. [Online] http://wiki.openid.net/Run_your_own_identity_server.
8. Public OpenID Providers. [Online] <http://wiki.openid.net/OpenIDServers>.
9. **Committee, OASIS eXtensible Resource Identifier (XRI) Technical.** Extensible Resource Identifier (XRI) Resolution Version 2.0. [Online] <http://docs.oasis-open.org/xri/2.0/specs/xri-resolution-V2.0.html>.
10. OpenID Authentication 2.0 - Final. [Online] http://openid.net/specs/openid-authentication-2_0.html.
11. Diffie-Hellman key exchange. *Wikipedia*. [Online] <http://en.wikipedia.org/wiki/Diffie-Hellman>.
12. **Raepple, Martin.** *The Developer's Guide To SAP NetWeaver Security*. s.l. : Galileo Press, 2008. 978-1592291809.
13. Yadis Specification 1.0. *yadis.org*. [Online] <http://yadis.org/papers/yadis-v1.0.pdf>.
14. OpenID Contributor Non Assertion Agreements. [Online] <http://openid.net/ipr/Non-Assertion-Agreement/executed/>.
15. **TC, OASIS Extensible Resource Identifier (XRI).** Extensible Resource Identifier (XRI) Syntax V2.0. [Online] <http://www.oasis-open.org/committees/download.php/15376>.

Copyright

© 2008 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, System i, System i5, System p, System p5, System x, System z, System z9, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, Informix, i5/OS, POWER, POWER5, POWER5+, OpenPower and PowerPC are trademarks or registered trademarks of IBM Corporation.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

These materials are provided "as is" without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

SAP shall not be liable for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials.

SAP does not warrant the accuracy or completeness of the information, text, graphics, links or other items contained within these materials. SAP has no control over the information that you may access through the use of hot links contained in these materials and does not endorse your use of third party web pages nor provide any warranty whatsoever relating to third party web pages.

Any software coding and/or code lines/strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, except if such damages were caused by SAP intentionally or grossly negligent.