IdPAuthUserPassLoginPage

Customizing the Username/Password Login Page

Finding, Updating, and Deploying the Login Page

The login page is a JavaServer Pages (JSP) and before attempting to edit the page you should be familiar with at least to basics of JSPs. Go out to the web and read a few tutorials (e.g. one, two, three).

To make changes to the login page and deploy them:

1. Edit *src/main/webapp/login.jsp* within your IdP distribution package
2. Run the IdP install script
3. Restart your Servlet container

Required Parts

While nearly everything on the login page can be customized a few pieces **must** always be there:

- The form element with an action element value of `<%=request.getAttribute("actionUrl")%>`
- The `j_username` input field must be used for the username.
- The `j_password` input field must be used for the user's password.

These elements can be moved around in the page, but the input fields must be located within the form.

Taglib support

From IdP version 2.3.0 a taglib is available to make certain commonly used fields easily available to the jsp pages. Many of these tags are aimed at making it simple to extract the information described in `<mdui:UIInfo>` extensions, as described here.

Declare this library by putting this statement

```
<%@ taglib uri="urn:mace:shibboleth:2.0:idp:ui" prefix="idpui" %>
```

at the top of the jsp file. This makes the tags described below available.

Depending on the precise metadata, some tags may not be able to find appropriate results. In such a situation the body of the tag (if present) is used.

Service Provider Name

Tag Format: `<idpui:serviceName/>`

Typical Outputs: "`sp.example.org`" *or* "`The Example SP`" *or* "`urn:foo:bar:1:24`"

This provides a user-friendly name for the service provider. No HTML markup is applied. Specifically, for services identified by metadata, it will:

- Look for a `<mdui:DisplayName>` of the appropriate language in a `<mdui:UIInfo>` extension attached to the SP's `<md:SPSSODescriptor>`.
- If that doesn't exist look for a `<md:ServiceName>` of the appropriate language in an `<md:AttributeConsumingService>` for the SP.
- In the absence of both of these, if the `EntityID` is a URL, then the hostname part of the URL is used.

- Otherwise the entire `EntityID` is used.

The resulting string is then inserted into the page with no markup.

In the case that the SP is unidentified by metadata (typically meaning support for "anonymous" relying parties has been enabled) the tag will output a default string. In V2.4.0 and later, the default can be overridden with a tag attribute called `defaultValue`.

Service Provider Description

Tag Format: `<idpui:serviceDescription>Default Value</idpui:serviceDescription>`

Typical Outputs: "`An example SP, used for something exciting`" *or* "`Default Value`"

This provides a user-friendly description the service provider. No HTML markup is applied. Specifically it will:

- Look for a `<mdui:Description>` of the appropriate language in a `<mdui:UIInfo/>` extension attached to the SP's `<md:SPSSODescriptor>`.
- If that doesn't exist look for a `<md:ServiceDescription>` of the appropriate language in an `<md:AttributeConsumingService>` for the SP.
- In the absence of both of these the body of the tag is used.

The resulting string is then inserted into the page with no markup.

Service Provider Organization Name (V2.4.0+)

Tag Format: `<idpui:organizationName>Default Value</idpui:organizationName>`

Typical Outputs: "`Widgets R Us`" *or* "`Default Value`"

This outputs a language-aware version of the `<md:OrganizationName>` element from the SP's `<md:SPSSODescriptor>` or surrounding `<md:EntityDescriptor>`, or the tag body in the event no such value exists.

The resulting string is then inserted into the page with no markup.

Note that many metadata sources may not populate this information in a manner useful for display in such a context. This will vary by federation.

Service Provider Organization Display Name (V2.4.0+)

Tag Format: `<idpui:organizationDisplayName>Default Value</idpui:organizationDisplayName>`

Typical Outputs: "`Widgets R Us`" *or* "`Default Value`"

This outputs a language-aware version of the `<md:OrganizationDisplayName>` element from the SP's `<md:SPSSODescriptor>` or surrounding `<md:EntityDescriptor>`, or the tag body in the event no such value exists.

The resulting string is then inserted into the page with no markup.

Note that many metadata sources may not populate this information in a manner useful for display in such a context. This will vary by federation.

Service Provider Organization URL (V2.4.0+)

Tag Format: `<idpui:organizationURL linkText="text" cssClass="class" cssId="id" cssStyle="style">default text</idpui:organizationURL>`

Typical Outputs: "`<a href="http://widgets.com" id="id">text</a>`" *or* "`default text`"

This searches the `<md:Organization>` element attached to the SP's `<md:SPSSODescriptor>` for an `<md:OrganizationURL/>` in the browser's language. If no such entry is found then the body of the tag is inserted. Otherwise the found URL is placed inside an `<a href="URL" class="cssClass" id="cssId" style="cssStyle"/>text</a>` element where

- **URL** is as taken from the metadata.
- **cssClass** is taken from the `cssClass` attribute of the tag. If there is no such attribute then the resultant hyperlink will not have a `class` attribute.
- **cssId** is taken from the `cssId` attribute of the tag. If there is no such attribute then the resultant hyperlink will not have an `id` attribute.
- **cssStyle** is taken from the `cssStyle` attribute of the tag. If there is no such attribute then the resultant hyperlink will not have an `style` attribute.
- **text** is taken from the tag's (mandatory) attribute "linkText".

Note that many metadata sources may not populate this information in a manner useful for display in such a context. This will vary by federation.

Service Provider Contact Name

Tag Format: `<idpui:serviceContact name="text" cssClass="class" cssId="id" cssStyle="style" contactType="type">default</idpui:serviceContact>`

Typical Output: "`<a href="mailto:contact@sp.example.edu" class="ContactClass">Technical Support</a>`"

This provides the appropriate contact information for the service provider. Specifically, the `contactType` attribute (default value "support" is used to locate a `<md:ContactPerson>` entry for the SP.

The display text is taken from the contents of the name attribute. If the attribute is not present then the name is extracted appropriately from the sub elements of the `<md:ContactPerson>`.

If no name is found then the body of the tag will be inserted.

If the `<md:EmailAddress>` is not present in the `<md:ContactPerson>` then the name is inserted with no additional markup.

Otherwise the name is rendered a hyperlink with the contents of the `<EmailAddress>` being the reference, the name being the body and the "cssClass", "cssId", "cssStyle" attributes from the tag being used (if present) to set the "class", "id", "style" attributes respectively in the `<a>` element.

For example given the following metadata segment

```
<ContactPerson contactType="support">
        <GivenName>John</GivenName>
        <SurName>Doe</SurName>
</ContactPerson>
<ContactPerson contactType="technical">
        <GivenName>Jane</GivenName>
        <SurName>Smith</SurName>
        <EmailAddress>mailto:tech@example.edu</EmailAddress>
</ContactPerson>
<ContactPerson contactType="billing">
        <EmailAddress>mailto:billing@example.edu</EmailAddress>
</ContactPerson>
```

Then

- The tag `<idpui:serviceContact cssId="cp"/>` would return `John Doe`.
- The tag `<idpui:serviceContact cssStyle="font-size:20px" contactType="technical"/>` would return `<a href="mailto:tech@example.edu" style="font-size:20px">Jane Smith</a>`
- The tag `<idpui:serviceContact cssClass="cpClass" name="Technical contact" contactType="technical"/>` would return `<a href="mailto:tech@example.edu" class="cpClass">Technical contact</a>`
- The tag `<idpui:serviceContact name="contact" contactType="billing"/>` would return `<a href="mailto:billing@example.edu">contact</a>`
- The tag `<idpui:serviceContact contactType="administrative">Your SP's administrators</idpui:serviceContact>` would return `Your SP's administrators`

Service Provider Privacy Statement.

Tag Format: `<idpui:servicePrivacyURL linkText="text" cssClass="class" cssId="id" cssStyle="style">default text</idpui:servicePrivacyURL>`

Typical Outputs: "`<a href="http://sp.example.edu/PrivacyStatement.html" id="privacyStatement">Privacy Statement</a>`" *or* "`default text`"

This searches the `<mdui:UIInfo>` extension attached to the SP's `<md:SPSSODescriptor>` for a `<mdui:PrivacyStatementURL/>` for the browser's language. If no such entry is found then the body of the tag is inserted. Otherwise the found URL is placed inside an `<a href="URL" class="cssClass" id="cssId" style="cssStyle"/>text</a>` element where

- **URL** is as taken from the metadata.
- **cssClass** is taken from the `cssClass` attribute of the tag. If there is no such attribute then the resultant hyperlink will not have a `class` attribute.
- **cssId** is taken from the `cssId` attribute of the tag. If there is no such attribute then the resultant hyperlink will not have an `id` attribute.
- **cssStyle** is taken from the `cssStyle` attribute of the tag. If there is no such attribute then the resultant hyperlink will not have an `style` attribute.
- **text** is taken from the tag's (mandatory) attribute "linkText".

Service Provider Information

Tag Format: `<idpui:serviceInformationURL linkText="text" cssClass="class" cssId="id" cssStyle="style">default</idpui:serviceInformationURL>`

Typical Outputs: "`<a href="http://sp.example.edu/Information.html" class="info">Further Information</a>`" *or* "`default`"

This searches the `<mdui:UIInfo>` extension attached to the SP's `<md:SPSSODescriptor>` for a `<mdui:InformationURL/>` for the browser's language. If no such entry is found then the body of the tag is inserted. Otherwise the found URL is placed inside an `<a href="URL" class="cssClass" id="cssId" style="cssStyle"/>text</a>` element where

- **URL** is as taken from the metadata.
- **cssClass** is taken from the `cssClass` attribute of the tag. If there is no such attribute then the resultant hyperlink will not have a `class` attribute.
- **cssId** is taken from the `cssId` attribute of the tag. If there is no such attribute then the resultant hyperlink will not have an `id` attribute.
- **cssStyle** is taken from the `cssStyle` attribute of the tag. If there is no such attribute then the resultant hyperlink will not have an `style` attribute.

- **text** is taken from the tag's (mandatory) attribute "linkText".

Service Provider Logos

Tag Format: `<idpui:serviceLogo cssId="Id" cssClass="class" cssStyle="style" minWidth="integer" maxWidth="integer" minHeight="integer" maxHeight="integer">default</idpui:serviceLogo>`

Typical Output: `<img src="https://sp.example.edu/Information.html" class="logo" />`

Attributes

Name	Default Value	Controls
alt	same as `<idpui:serviceName/>`	The text to associate with the alt attribute
minHeight	0	The minimum height of any returned logo
maxHeight	MaxInt	The maximum height of any returned logo
minWidth	0	The minimum width of any returned logo
maxWidth	MaxInt	The maximum width of any returned logo
cssId	<none>	If present, the value is included as the id for the image (`id="value"`)
cssClass	<none>	If present, the value is included as the class for the image (`class="value"`)
cssStyle	<none>	If present, the value is included as the style for the image (`style="value"`)

This tag searches the content of the `<mdui:UIInfo>` element attached to the SP's `<md:SPSSODescriptor>` for a `<mdui:Logo>` element in the browser's language (or one with no language associated), suitably constrained by the attributes of the `<idpui:serviceLogo>` element. If such an `<mdui:Logo>` element is found, then result is

```
<img src="URL" [cssId="suppliedIdIfPresent"] [class="suppliedCssClassIfPresent"] alt="suppliedText" />
```

Otherwise the result is the content of the `<idpui:serviceLogo>` element (if present).

Recommendations for metadata extensions

Federations that rely on the Shibboleth IdP are encouraged to make recommendations to service providers about their metadata extensions.

Other Available APIs

As of IdP release 2.1.3 information regarding the current login process is more easily available. This information can be gotten through the use of the HttpServletHelper.

Almost all of the `HttpServletHelper` methods require a `ServletContext` as a parameter. This is available as the standard JSP variable `application`.

Login Context

The most useful object available via this Helper is the LoginContext and its SAML protocol specific subclasses ShibbolethSSOLoginContext (if it's a SAML 1 request) or a SAML2LoginContext (if it's a SAML 2 request).

The login context will give you access to information such as the entity ID of the relying party (the service provider), the requested authentication methods, the authentication method the IdP is attempting to perform, whether the authentication is a forced and/or passive authentication, etc.

So, to get the `LoginContext` you would do the following:

```
<%@ page import="edu.internet2.middleware.shibboleth.idp.util.HttpServletHelper" %>
<%@ page import="org.opensaml.util.storage.StorageService" %>
<%@ page import="edu.internet2.middleware.shibboleth.idp.authn.LoginContext" %>


<%
   StorageService storageService = HttpServletHelper.getStorageService(application);
   LoginContext loginContext = HttpServletHelper.getLoginContext(storageService,application, request);
%>
```

You could then use the `LoginContext` like this:

```
Shibboleth Identity Provider Login to Service Provider <%= loginContext.getRelyingPartyId() %>
```

Relying Party Metadata

Another useful set of information is the EntityDescriptor metadata for the relying party. This can include information such as human readable names and descriptions, informational URLs, etc. To get to the metadata for the relying party you would do the following:

```
<%@ page import="edu.internet2.middleware.shibboleth.idp.util.HttpServletHelper" %>
<%@ page import="org.opensaml.util.storage.StorageService" %>
<%@ page import="edu.internet2.middleware.shibboleth.idp.authn.LoginContext" %>
<%@ page import="edu.internet2.middleware.shibboleth.common.relyingparty.RelyingPartyConfigurationManager" %>
<%@ page import="org.opensaml.saml2.metadata.EntityDescriptor" %>

<%
   StorageService storageService = HttpServletHelper.getStorageService(application);
   LoginContext loginContext = HttpServletHelper.getLoginContext(storageService,application, request);
   RelyingPartyConfigurationManager rpConfigMngr = HttpServletHelper.getRelyingPartyConfigurationManager
(application);

   EntityDescriptor metadata = HttpServletHelper.getRelyingPartyMetadata(loginContext.getRelyingPartyId(),
rpConfigMngr);
%>
```

Handling Login Errors

Authentication Failures

One significant drawback to the use of JAAS as the authentication provider within the IdP is that it does not bubble up very useful authentication errors. It simply indicates whether the authentication failed or succeeded. You can determine if an authentication attempt failed by checking that the request attribute `LoginHandler.AUTHENTICATION_EXCEPTION_KEY` is not null.

For example:

```
<%@ page import="edu.internet2.middleware.shibboleth.idp.authn.LoginHandler" %>

<% if (request.getAttribute(LoginHandler.AUTHENTICATION_EXCEPTION_KEY) != null) { %>
   <p><font color="red">Authentication Failed</font></p>
<% } %>
```

Creating a more detailed response when using ActiveDirectory

If you are using Microsoft ActiveDirectory as your authentication realm, you can make use of the sub error codes that MSAD sends along with the LDAP failure code of 49.. details here.

Direct Login Page Access

Another common error comes from the misuse of the IdP. The login page can not be accessed directly, it can only be accessed after the IdP has done some initial processing of a valid authentication request. However, some users will mistakenly access the login page because they bookmarked it, found it in their browser's history, or by means of the back button. The best way to detect this is to look for the presence of the `LoginContext` and, if not available, display an appropriate error message.

For example:

```
<%@ page import="edu.internet2.middleware.shibboleth.idp.util.HttpServletHelper" %>
<%@ page import="org.opensaml.util.storage.StorageService" %>
<%@ page import="edu.internet2.middleware.shibboleth.idp.authn.LoginContext" %>


<%
   StorageService storageService = HttpServletHelper.getStorageService(application);
   LoginContext loginContext = HttpServletHelper.getLoginContext(storageService,application, request);
%>

<% if (loginContext == null) {%>
  <p><font color="red">Error:</font> Direct access to this page is not supported.
  Please ensure that you did not bookmark this page or reach it by using the back
  button or selecting it from your browser history.  To log in to a particular
  service, please visit that service first.</p>
<% } else { %>
   <!-- normal login page display goes here -->
<% } %>
```

Preventing Display Of Login Page in a Frame

Display login pages within a frame is a very common means of performing a "click-jacking" attack. The aforementioned OWASP page provides a "frame buster" javascript that can be used to attempt to prevent this kind of attack. It also identifies the `X-FRAME-OPTIONS` header. This header can be set as follows:

```
<% response.addHeader("X-FRAME-OPTIONS", "DENY"); %>
```