Space Details

Key:	PentahoDoc
Name:	BI Server Documentation - Latest
Description:	Latest version of the Pentaho BI Server
Creator (Creation Date):	admin (Nov 15, 2006)
Last Modifier (Mod. Date):	admin (Nov 27, 2006)

Available Pages

- Security
  - 00. What's New in 1.6
  - 01. Introduction to Security Concepts
  - 02. Security Implementation Options
  - 03. Platform Security Implementation
    - 01. Security Data Access Objects
      - LDAP Configuration Migration
      - LDAP Search Filter Syntax
    - 02. Authentication
      - Single Sign-On
        - 01. Central Authentication Service
          - Enabling SSL in Tomcat
        - 02. JBoss Portal
        - 03. Pentaho BI Platform
        - 04. Troubleshooting
    - 03. Web Resource Authorization
    - 04. Domain Object Authorization
  - 04. Security HOWTOs
    - Changing the Admin Role
    - Changing to the JDBC Security DAO
    - Changing to the LDAP Security DAO
    - Customizing ACL Decisions
    - Customizing the Login Page
    - Implementing a New Security DAO
    - Re-Applying Default ACL
    - Refreshing JBoss Portal Database
    - Removing Security
    - Using Security from Action Sequences

Security

This page last changed on Mar 16, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

00. What's New in 1.6

This page last changed on Mar 16, 2007 by mlowery.

> **Useful Information**
>
> Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

This page lists the changes to the security system within Pentaho BI Platform version 1.6.

Libraries

- Upgraded from JBoss Portal 2.2.1-SP3 to JBoss Portal 2.4.1-SP1.
- Upgraded from Acegi Security 1.0.0-RC2 to Acegi Security 1.0.2.
- Upgraded from Spring 2.0-m3 to Spring 2.0.

Pentaho Code

General

- Moved classes within `org.acegisecurity` package within Pentaho code to `com.pentaho.security`. The `org.acegisecurity` package originally existed to subclass protected Acegi Security classes. The elimination of this package was made possible by delegating to Acegi Security classes rather than extending them. The `UserRoleListService` implementations all delegate to Acegi Security.
  - Also moved testcases out of the `org.acegisecurity` package.
  - `org.acegisecurity.context`, `org.acegisecurity.providers.portlet`, and `org.acegisecurity.ui.portlet` packages (and their classes) were not moved but instead eliminated as they were unused.
- Replaced `userName` with `username` throughout code to be consistent with Acegi Security.

Security Data Access Objects

- Refactored `UserRoleListService` implementations. These changes are detailed in security data access objects. Essentially the goal was to switch from extension of Acegi Security classes to delegation.
  - Refactoring of in-memory and database implementations of `UserRoleListService` was relatively straightforward; LDAP implementation is very different because it includes numerous improvements.
- LDAP implementation of `UserRoleListService`
- Split up configuration files to allow for easier switching between backend security datastores.

- Eliminated `DirMgrBindAuthenticator`. This class was originally created as a workaround to a bug in a pre-1.0 release of Apache Directory Server. That bug has since been fixed, obviating this class.

Login & Logout

- Improved messaging in the login page. (See screenshots.)
  - Added reason for login failure.
  - Added detection of logged in user.
  - Added detection of HTTP session re-use.
- Eliminated logoff.jsp which contained logout logic. This JSP was defined in `web.xml` as `/Logout`. Logout logic now resides in `ProPentahoLogoutHandler` which is defined in `applicationContext-acegi-security.xml`.

Web Resource (URL) Authorization

- Replaced `ForceLoginFilter` with `FilterSecurityInterceptor`. This class, while more verbose, allows for declarative URL authorization rules based on users and/or roles. As a consequence of using `FilterSecurityInterceptor`, if a user requests a page and he or she is not logged in, their original request will be stored, they will be redirected to the login page, and upon entering valid credentials, they will be redirected to the original URL.
- Eliminated `ProPentahoSystem.isAdministrator()` calls from Admin pages. This protection is now provided by `FilterSecurityInterceptor`. If the user requests a page for which they do not have access, status code 403 (Forbidden) is returned. A custom error page can be displayed.

Samples

- Added SQL script for relational database sample.
- Added LDIF script for directory server sample.

01. Introduction to Security Concepts

This page last changed on Mar 21, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

Security

Security, as it relates to software, can be defined as the protection of information or functionality from access by individuals who have not explicitly been granted access. Below are some security-related terms that will be used throughout the remainder of this document.

Authentication

Authentication is the process of confirming that the user requesting access is the user that they claim to be. This is often done by presenting a user identifier (e.g. a username) paired with a secret known only to that user (e.g. a password), but can sometimes involve certificates or other means of establishing identity. In this documentation, authentication is synonymous with login.

Authorization

Authorization is the process of deciding if the user (who has been authenticated) is allowed to access the information or functionality for which he or she is making the request. A software system can protect itself at multiple levels. In the Pentaho BI Platform, pages in the web-based user interface can be protected. In addition, objects within the solution repository, such as folders and action sequences, can be protected using access control lists.

Web Resource (URL) Authorization

In a web-based application, developers can protect specific URLs which unique identify web pages. Protecting URLs is done by specifying what user or role is required to view a page. Note that read is the only permission applicable to web resources. Even though accessing a page might delete records in a backend datastore, a developer can only specify that a page is viewable or not.

Domain Object Authorization

An application developer can protect particular instances of objects. A typical example with which most users are familiar is a filesystem. Each file or directory in a filesystem has an access control list (ACL) associated with it. In the case of a filesystem, one can specify that user `suzy` can `read` the file named `readme.txt`. In this example, the object is the file `readme.txt`; the recipient is `suzy`; and the permission is `read`.

Access Control List

An access control list (ACL) is associated with an object and contains entries that specify who (i.e. the recipient) can do what (i.e. the permission) with the associated object.

Authority, Role, & Group

In the Pentaho BI Platform, the terms authority, role, and group are synonymous. They are used during authorization to decide whether to grant or deny access. An example of a role is `ROLE_ADMIN`.

02. Security Implementation Options

This page last changed on Mar 16, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

Implementation Options

Design Goals

The security system of the Pentaho BI Platform strives to meet the following requirements:

- Transparency: As security is a cross-cutting concern, security should be as transparent as possible.
- Extensibility: The security system must provide the option to extend key classes to fit customers' needs.
- Flexibility: The security system should be easily customized, preferably declaratively.
- Portability: To the greatest extent possible, the security system should not use container-specific libraries.

Java Enterprise Container Security

While the J2EE 1.4 specification provides for the declarative protection of URLs, it leaves the implementation of authentication and authorization up to each vendor. Communicating to the container what security implementation to use often requires container-specific configuration files. This option limits portability between vendor containers.

Acegi Security System for Spring

The Acegi Security System for Spring provides portability by implementing security completely within the web application, inside the container, but using only Java Enterprise APIs. Acegi Security provides a host of other benefits including:

- Based on the Spring Framework.
- Domain object security.
- Transparent URL security using filters.
- Multiple security back-ends.

03. Platform Security Implementation

This page last changed on Mar 16, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

Implementation

Technologies Used

Acegi Security

In the Pentaho BI Platform, security is based on the infrastructure provided by the Acegi Security System for Spring. Because the platform builds on top of Acegi Security, it is highly recommended that readers consult the Acegi Security documentation, specifically the *Tutorial Sample*.

Warning

The online documentation for Acegi Security is for the **latest** version of Acegi Security. At the time of this writing, the latest version was 1.0.3 while the Pentaho BI Platform is based on 1.0.2. These versions have significant differences and it is not recommended that you use the online documentation. Instead, go to the Acegi Security downloads page and download `acegi-security-1.0.2.zip` which when expanded creates a `doc` directory. Opening `index.html` with your browser then allows for local browsing of the correct version of Acegi Security.

Spring Framework

Acegi Security is written to take advantage of the Spring Framework so, following that example, the platform leverages the Spring Framework's "dependency injection" capabilities to declaratively configure security. Spring is a huge framework, covering many aspects--including data access objects, MVC, and dependency injection. The platform only directly takes advantage of dependency injection (a.k.a. inversion of control). Essentially, this allows an application deployer to "inject" objects on which the system depends. Because the platform builds on top of Spring, it is highly recommended that readers consult the Spring documentation, specifically the Spring beans XML DTD.

Warning

While the Pentaho BI Platform uses Spring 2.0, it does not take advantage of any Spring

2.0-features. Specifically, the Spring bean XML files adhere to the Spring 1.x DTD. This is noted to steer new-to-Spring readers away from XML Schema-based configuration which the platform does not use.

Security Breakdown

In order to deliver the documentation on security within the platform in manageable chunks, security has been broken down into the areas listed in the table below.

Useful Information

Note that the areas listed below are not necessarily how security is partitioned in terms of configuration files.

Area	Description
Security data access objects	Security data includes usernames, passwords, granted authorities, web resource (URL) protection data, and ACLs for domain objects.
Authentication	This area is concerned with processing interactive login information (e.g. username and password) and comparing it with data retrieved from the security datastore.
Web resource (URL) authorization	Protecting URLs is a matter of answering for each user, whether or not they can access each URL (web page). Note that access here is `Yes` or `No`--there is no `Read` or `Write` granularity. Given an authenticated user, it is the responsibility of web resource authorization to decide whether to allow the page to be accessed.
Domain object authorization	Currently, the only domain objects protected by the platform are solution repository objects (e.g. action sequences). Given an authenticated user, it is the responsibility of domain object authorization to decide whether to allow the requested operation.

Core Security Types

The word "types," as used here, means Java interfaces or classes. Below, the core security types are described. These types are used throughout the platform including the areas of web resource authorization and domain object authorization.

`Authentication`

`org.acegisecurity.Authentication` instances represent authentication requests. When passed to an `AuthenticationManager`, the request, in the form of an `Authentication` instance, will be authenticated. If the authentication is successful, a fully populated `Authentication` instance (including granted authorities) will be returned. If the authentication is unsuccessful, an `AuthenticationException` will be thrown. For a web-based application, the `Authentication` is stored between requests in the HTTP session (because the connection is stateless). While processing a request, the `Authentication` is stored in a `SecurityContextHolder`.

GrantedAuthority

Each `org.acegisecurity.GrantedAuthority` instance represents a permission that has been granted to an authenticated user. It can also refer to a role of which the user is a member. That role can then be associated with permissions such as those specified by an ACL.

SecurityUtils

Many of the functions that the ACL voter implementations rely on are provided in the `com.pentaho.security.SecurityUtils` class. Specifically, the `getAuthentication()` method is extremely important because it's used to get the Acegi Security `Authentication` object that should be bound to the user's session.

01. Security Data Access Objects

This page last changed on Mar 16, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

Getting Security Data

There are two key interfaces that fetch security data: `UserDetailsService` and `UserRoleListService`.

UserDetailsService

The Acegi Security class `UserDetailsService` defines a single method: `UserDetails loadUserByUsername(String username)`. Given a username, it returns a `UserDetails` instance.

UserRoleListService

The Pentaho class `UserRoleListService` defines 4 methods:

- `GrantedAuthority[] getAllAuthorities()`
- `String[] getAllUsernames()`
- `String[] getUsernamesInRole(GrantedAuthority authority)`
- `GrantedAuthority[] getAuthoritiesForUser(String username)`

For the final method above, all `UserRoleListService` implementations should delegate to the associated `UserDetailsService`, if possible.

Choice of Security Back-end

An organization can keep their security data in one or more back-ends. Typical security back-ends include relational databases and directory servers accessed via LDAP. Using Acegi Security, the platform offers these choices of security back-ends:

Backend	`UserDetailsService` implementation	`UserRoleListService` implementation

In-Memory*	`InMemoryDaoImpl`	`InMemoryUserRoleListService`
DBMS	`JdbcDaoImpl`	`JdbcUserRoleListService`
Directory Server	`LdapUserDetailsService`	`DefaultLdapUserRoleListService`

* An in-memory security back-end is primarily provided for testing or proofs of concept. In other words, it's not intended to be used in production.

Configuration Files

The platform stores its security data access object (DAO) configuration files in the `WEB-INF` directory. The table below enumerates these files.

Filename	Description
`applicationContext-acegi-security-*.xml`	Defines a `UserDetailsService` based on *. Examples of * include `memory`, `jdbc`, and `ldap`. The `UserDetailsService` defined in this file is referenced in `applicationContext-acegi-security.xml` and `applicationContext-pentaho-security-*.xml`. Read more about the `UserDetailsService` implementations and how to configure them in Getting Password and Granted Authorities of a User.
`applicationContext-pentaho-security-*.xml`	Defines a `UserRoleListService` based on *. Examples of * include `memory`, `jdbc`, and `ldap`. Read more about the `UserRoleListService` implementations and how to configure them in Getting All Usernames and Roles.

One might ask: Why are there so many configuration files? The Acegi Security Contacts Sample Application doesn't have this many! The reason for all the files is (1) to partition the files to allow them to be swapped out in order to connect to different security backends and (2) to provide examples configurations for each of the three supported security backends: memory, dbms, and ldap.

Because of the partitioning, the `contextConfigLocation` `context-param` in `web.xml` has a consistent template which looks like the following:

Warning

Note that the end-of-line backslashes that occur in the excerpt below are present for formatting purposes only and should not be present in the actual file.

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext-acegi-security.xml \
    /WEB-INF/applicationContext-common-authorization.xml \
    /WEB-INF/applicationContext-acegi-security-*.xml \
    /WEB-INF/applicationContext-pentaho-security-*.xml</param-value>
</context-param>
```

Given this template, to switch to an LDAP-based security backend, you simply replace the * above with `ldap`. To switch to memory-based, use `memory`. And finally, to switch to db-based, use `jdbc`.

Getting Password and Granted Authorities of a User

The DAO that has the responsibility of fetching a password and granted authorities, given a username, is an instance of `UserDetailsService` and is defined in `applicationContext-acegi-security-*.xml`.

You'll notice a naming convention in the files listed in this section. Each `UserDetailsService` implementation has a bean name of `userDetailsService`. Why is that? It's because `applicationContext-acegi-security.xml` has a dependency on a bean named `userDetailsService`. For this reason, do not change the names of these beans.

Useful Information

The vast majority of the configuration contained in the `applicationContext-acegi-security-*.xml` files is a standard Acegi Security setup and is well-documented in the Acegi Security documentation. Where the configuration strays from the Acegi Security documentation, it is documented below.

Memory

This DAO reads usernames, passwords, and roles specified in a Spring XML file.

The `InMemoryDaoImpl` class uses an instance of `UserMap`. But a `UserMap` restricts how the information passed into its constructor can be accessed. For example, one cannot ask the question of a `UserMap`: What are all the users? Since the platform user management system needs to answer this exact question, the platform comes with a way to intercept the information passed into a `UserMap` constructor. The information passed into the `UserMap` constructor is intercepted as a `String` and later fed into a `UserMapFactoryBean` for use in the `InMemoryDaoImpl`.

But `InMemoryDaoImpl` doesn't take a `UserMapFactoryBean`! It takes a `UserMap`! The secret to this working lies in the Spring type called `FactoryBean`. When Spring detects a bean of this type, instead of returning the instance, it returns `instance.getObject()`.

```
<bean id="userMap" class="java.lang.String">
  <constructor-arg type="java.lang.String">
    <value>
    <![CDATA[
    joe=password,ROLE_ADMIN,ROLE_CEO,ROLE_AUTHENTICATED
    suzy=password,ROLE_CTO,ROLE_IS,ROLE_AUTHENTICATED
    pat=password,ROLE_DEV,ROLE_AUTHENTICATED
    tiffany=password,ROLE_DEV,ROLE_DEVMGR,ROLE_AUTHENTICATED
    admin=secret,ROLE_ADMIN,ROLE_AUTHENTICATED
    ]]>
    </value>
  </constructor-arg>
</bean>

<bean id="userMapFactoryBean"
```

```
    class="com.pentaho.security.memory.UserMapFactoryBean">
    <property name="userMap">
      <ref local="userMap" />
    </property>
  </bean>

  <bean id="userDetailsService"
    class="org.acegisecurity.userdetails.memory.InMemoryDaoImpl">
    <property name="userMap">
      <ref local="userMapFactoryBean" />
    </property>
  </bean>
```

LDAP

This DAO reads usernames, passwords, and roles specified in directory server.

Acegi Security doesn't provide an LDAP-based `UserDetailsService`. So the platform filled that gap with `LdapUserDetailsService`. `LdapUserDetailsService` uses an `LdapUserSearch`, an `LdapAuthoritiesPopulator`, and an optional `LdapUserDetailsMapper`. Note that `LdapUserSearch`, `LdapAuthoritiesPopulator`, and `LdapUserDetailsMapper` are highly specialized classes whose purpose is to find a user record, find the roles of that user, and map the attributes of the user record into a `UserDetails` instance respectively.

Handy Hint

For an overview of search filter syntax, see LDAP Search Filter Syntax.

```
  <bean id="initialDirContextFactory"
    class="org.acegisecurity.ldap.DefaultInitialDirContextFactory">
    <constructor-arg index="0"
      value="ldap://localhost:10389/ou=system" />
    <property name="managerDn" value="uid=admin,ou=system" />
    <property name="managerPassword" value="secret" />
  </bean>

  <bean id="ldapAuthProvider"
    class="org.acegisecurity.providers.ldap.LdapAuthenticationProvider">
    <constructor-arg>
      <bean
        class="org.acegisecurity.providers.ldap.authenticator.BindAuthenticator">
        <constructor-arg>
          <ref local="initialDirContextFactory" />
        </constructor-arg>
        <property name="userSearch">
          <ref local="userSearch" />
        </property>
      </bean>
    </constructor-arg>
    <constructor-arg>
      <ref local="populator" />
    </constructor-arg>
  </bean>

  <bean id="populator"
    class="org.acegisecurity.providers.ldap.populator.DefaultLdapAuthoritiesPopulator">
    <constructor-arg index="0">
      <ref local="initialDirContextFactory" />
    </constructor-arg>
    <constructor-arg index="1" value="ou=roles" />
    <property name="groupRoleAttribute" value="cn" />
    <property name="groupSearchFilter" value="roleOccupant={0}" />
  </bean>
```

```
<bean id="userSearch"
  class="org.acegisecurity.ldap.search.FilterBasedLdapUserSearch">
  <constructor-arg index="0" value="ou=users" />
  <constructor-arg index="1" value="cn={0}" />
  <constructor-arg index="2">
    <ref local="initialDirContextFactory" />
  </constructor-arg>
</bean>

<bean id="userDetailsService"
  class="com.pentaho.security.ldap.LdapUserDetailsService">
  <property name="userSearch">
    <ref local="userSearch" />
  </property>
  <property name="populator">
    <ref local="populator" />
  </property>
</bean>
```

FixedDefaultLdapAuthoritiesPopulator

Pentaho provides an alternative implementation of `LdapAuthoritiesPopulator`. It is called `FixedDefaultLdapAuthoritiesPopulator`. The default configuration for the platform uses this class instead of `DefaultLdapAuthoritiesPopulator` because `FixedDefaultLdapAuthoritiesPopulator` fixes an issue with search scope in `DefaultLdapAuthoritiesPopulator`. See Spring Security Issue SEC-450.

Getting All Usernames and Roles

The DAO that has the responsibility of fetching all usernames and authorities (plus a few other responsibilities) is an instance of `UserRoleListService` and is defined in `applicationContext-pentaho-security-*.xml`.

Warning

The `UserDetailsRoleListService` class introduced in this section wraps a `UserRoleListService` instance and must be defined in the Spring XML Beans document for proper functioning of the Pentaho BI Platform.

Memory

The in-memory implementation of `UserRoleListService` is `InMemoryUserRoleListService`. Notice the bean below named `userRoleListEnhancedUserMapFactoryBean`? It refers to the bean defined in `applicationContext-acegi-security-memory.xml`. It uses the same `FactoryBean` trick that is used by `UserMapFactoryBean`. In an indirect fashion, the information contained in a `UserMap` is passed into the `InMemoryUserRoleListService`.

There's one more property to mention: `allAuthorities`. This defines all authorities that are allowed to be granted to users. Why can't the platform get this information from the `UserRoleListEnhancedUserMap`? That's because the `UserRoleListEnhancedUserMap` might only contain a subset of all the available authorities.

```
<bean id="userRoleListEnhancedUserMapFactoryBean"
  class="com.pentaho.security.memory.UserRoleListEnhancedUserMapFactoryBean">
  <property name="userMap" ref="userMap" />
</bean>

<bean id="inMemoryUserRoleListService"
  class="com.pentaho.security.memory.InMemoryUserRoleListService">
  <property name="userRoleListEnhancedUserMap">
    <ref local="userRoleListEnhancedUserMapFactoryBean" />
  </property>
  <property name="userDetailsService" ref="userDetailsService" />
  <property name="allAuthorities">
    <list>
      <bean class="org.acegisecurity.GrantedAuthorityImpl">
        <constructor-arg value="ROLE_AUTHENTICATED" />
      </bean>
      <!-- some authorities omitted -->
      <bean class="org.acegisecurity.GrantedAuthorityImpl">
        <constructor-arg value="ROLE_ANONYMOUS" />
      </bean>
    </list>
  </property>
</bean>

<bean id="pentahoUserRoleListService"
  class="com.pentaho.security.UserDetailsRoleListService">
  <property name="userRoleListService">
    <ref local="inMemoryUserRoleListService" />
  </property>
</bean>
```

DBMS

The DBMS implementation of `UserRoleListService` is `JdbcUserRoleListService`. It is analogous to `JdbcDaoImpl`. It simply has three more properties defining the SQL queries that can return the information defined by the `UserRoleListService` interface.

Be Careful

Be sure to add `as username` and `as authorities` where appropriate in your queries.

```
<bean id="jdbcUserRoleListService"
  class="com.pentaho.security.jdbc.JdbcUserRoleListService">
  <property name="allAuthoritiesQuery">
    <value>
      <![CDATA[SELECT distinct(authority) as authority FROM authorities]]>
    </value>
  </property>
  <property name="allUsernamesInRoleQuery">
    <value>
      <![CDATA[SELECT distinct(username) as username FROM granted_authorities where authority =
?]]>
    </value>
  </property>
  <property name="allUsernamesQuery">
    <value>
      <![CDATA[SELECT distinct(username) as username FROM users]]>
    </value>
  </property>
  <property name="dataSource" ref="dataSource" />
</bean>

<bean id="pentahoUserRoleListService"
  class="com.pentaho.security.UserDetailsRoleListService">
  <property name="userRoleListService">
    <ref local="jdbcUserRoleListService" />
  </property>
```

```
</bean>
```

LDAP

The LDAP implementation of `UserRoleListService` is `DefaultLdapUserRoleListService`.

> **Useful Information**
>
> Note that this version is a significant refactoring of the previous LDAP implementation. See LDAP Configuration Migration for a side-by-side comparison of old and new.

In this release of the platform, some new types have been introduced. These new types are outlined below. Before the new types are introduced, it is important to be familiar with a key JNDI class: `DirContext`. This class is the interface into directory services. In particular, there is a method with the following signature:

```
public NamingEnumeration search(String name,
                                String filterExpr,
                                Object[] filterArgs,
                                SearchControls cons)
                         throws NamingException;
```

Notice the parameters to this method? In the platform, these parameters are encapsulated into `LdapSearchParams`.

LdapSearchParams

`LdapSearchParams` bundle together the parameters that are eventually passed into a `search` call on a `DirContext` instance. The best description of these parameters is in the javadoc for the `DirContext` class but a brief description of each is below.

Parameter Name	Description
`name`	The DN of the node at which to begin the search. This parameter is also referred to as a search base.
`filterExpr`	A query for objects in the directory. This query can contain placeholders. This is analogous to a parameterized SQL query. See LDAP Search Filter Syntax for a quick overview of the syntax. See LDAP Search Filter Syntax for a quick overview of the syntax.
`filterArgs`	The values to be substituted for the placeholders in `filterExpr`.
`cons`	Search controls.

`LdapSearchParams` can only be created by `LdapSearchParamsFactory` instances.

LdapSearchParamsFactory

`LdapSearchParamsFactory` defines a single method:

```
LdapSearchParams createParams(Object[] filterArgs);
```

The platform provides a single implementation of this interface called `LdapSearchParamsFactoryImpl`. The constructor of this class requires a `name`, `filterExpr`, and `searchControls`--all of which go into creating `LdapSearchParams` instances.

LdapSearch

The `LdapSearch` interface is a generalization of `org.acegisecurity.ldap.LdapUserSearch`. In Acegi Security's `LdapUserSearch`, the goal was to find a *user* object in the directory; in Pentaho's `LdapSearch`, the goal is to find *any* object in the directory. In the platform, there are two implementations of this interface: `GenericLdapSearch` and `UnionizingLdapSearch`.

GenericLdapSearch

`GenericLdapSearch` instances are created using the four parameters shown below. The diagram below shows how `GenericLdapSearch` uses these parameters to execute a search.

Parameter Name	Description
`initialDirContextFactory`	An instance of `InitialDirContextFactory`.
`paramsFactory`	An instance of `LdapSearchParamsFactory`.
`resultsTransformer`	Transforms LDAP search results into custom objects. The type of this parameter is `Transformer`.
`filterArgsTransformer`	Transforms filter arguments before passing to the `paramsFactory`. The type of this parameter is `Transformer`.

The `Transformer` interface is a simple but powerful type. Transformers can be chained together, each doing a small part of a large task. There are three transformers provided with the platform.

SearchResultToAttrValueList

`SearchResultToAttrValueList` extracts the value of the token `tokenName` from the attribute `attributeName` which is taken from the `SearchResult` input.

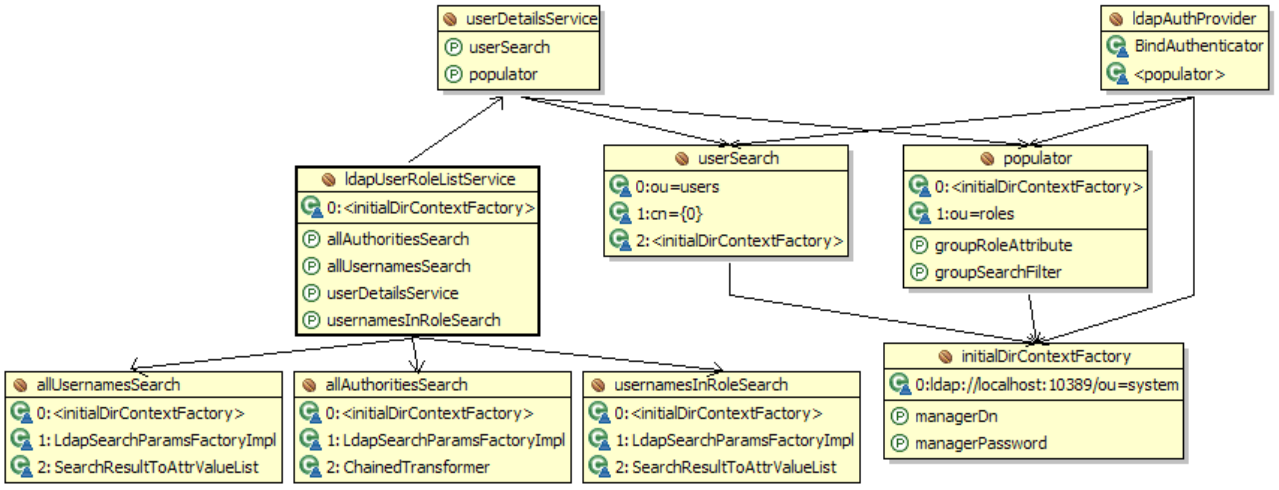
StringToGrantedAuthority

Authorities in a directory server are simple strings. In order to each of those strings into a `GrantedAuthority`, this transformer is used. It provides the option of adding a role prefix and converting the string's case.

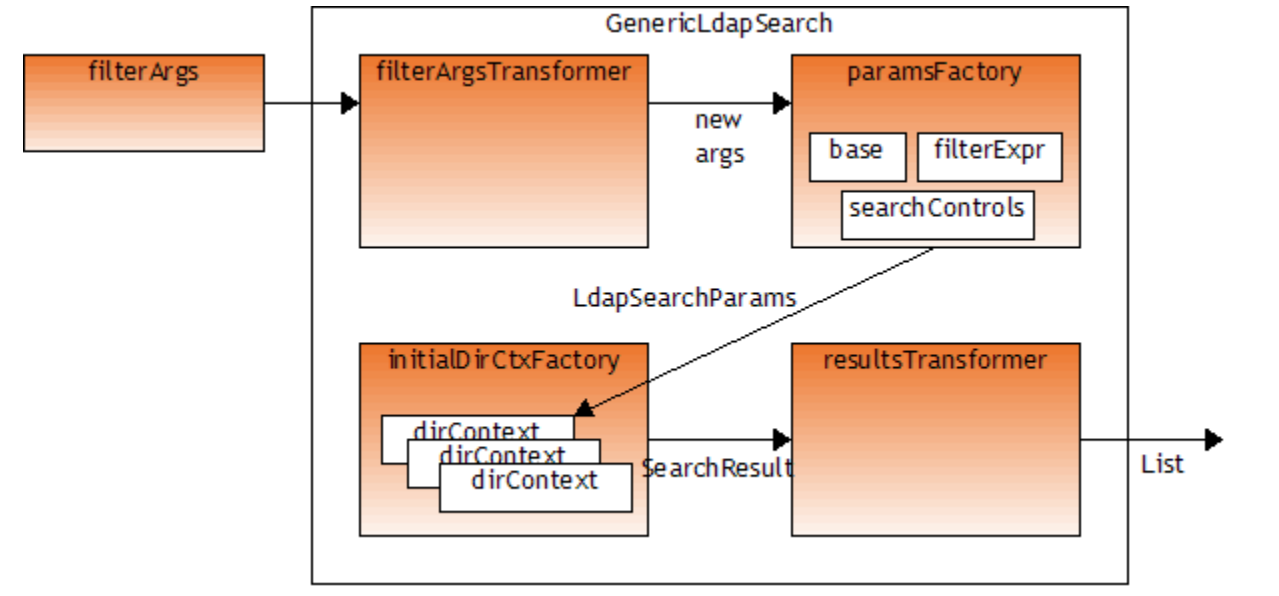
GrantedAuthorityToString

This class does the reverse of `StringToGrantedAuthority` except that it cannot control string case. This could be used in the query for what users are in particular role. The role is passed in with a role prefix, this transformer removes it, and the new role string is passed in as a filter argument.

Example LDAP Object Graph



GenericLdapSearch Internals



UnionizingLdapSearch

The purpose of this class is to merge the results returned by two or more `LdapSearch` instances.

DefaultLdapUserRoleListService

Remember that `DefaultLdapUserRoleListService` is the LDAP implementation of

. As a consequence, it must implement the methods defined in `UserRoleListService`. Recall that those methods are: `getAllAuthorities()`, `getAllUsernames()`, `getUsernamesInRole()`, and `getAuthoritiesForUser()`. `DefaultLdapUserRoleListService` implements the first three methods by delegating to three separate `LdapSearch` instances--stored in its `allAuthoritiesSearch`, `allUsernamesSearch`, and `usernamesInRoleSearch` properties. (Remember `GenericLdapSearch` is an implementation of `LdapSearch`.) `DefaultLdapUserRoleListService` implements the last method by delegating to an `LdapUserDetailsService` instance--stored in its `userDetailsService` property. The `userDetailsService` property is not detailed below since it is covered in the section on LdapUserDetailsService.

And since `DefaultLdapUserRoleListService` is configured via Spring, the task is reduced to defining three `GenericLdapSearch` instances (plus an `LdapUserDetailsService`) in Spring! But before the Spring config is presented, the equivalent configuration via Java code is presented. This route is chosen since Spring configuration can be very verbose and most are more familiar with the more ubiquitous Java syntax.

```
// create params factory with the following settings:
// search base="ou=users",
// filterExpr="(objectClass=person)",
LdapSearchParamsFactory paramsFactory = new LdapSearchParamsFactoryImpl(
  "ou=users", "(objectClass=Person)");

// create a resultsTransformer that extracts the uid attribute
Transformer transformer = new SearchResultToAttrValueList("uid");

// create a GenericLdapSearch with objects created above;
// (don't worry about getInitialCtxFactory()--just know that
// it returns an InitialDirContextFactory)
LdapSearch allUsernamesSearch = new GenericLdapSearch(
  getInitialCtxFactory(), paramsFactory, transformer);
```

Now the equivalent Spring config is presented.

```
<bean id="allUsernamesSearch"
  class="com.pentaho.security.ldap.search.GenericLdapSearch">
  <constructor-arg index="0" ref="initialDirContextFactory" />
  <constructor-arg index="1">
    <bean
      class="com.pentaho.security.ldap.search.LdapSearchParamsFactoryImpl">
      <constructor-arg index="0" value="ou=users" />
      <constructor-arg index="1" value="objectClass=Person" />
    </bean>
  </constructor-arg>
  <constructor-arg index="2">
    <bean
      class="com.pentaho.security.ldap.transform.SearchResultToAttrValueList">
      <constructor-arg index="0" value="uid" />
    </bean>
  </constructor-arg>
</bean>
```

Useful Information

Why are `constructor-arg` elements used? Why not call property setters instead? The reason for this is that the only way to set some of the properties in the example below is to pass those properties in during object creation. This enforces the policy that these properties should be set once and never changed.

Handy Hint

For an overview of search filter syntax, see LDAP Search Filter Syntax.

The steps to create a `GenericLdapSearch` have been introduced. Follow these steps to create the three searches required by `DefaultLdapUserRoleListService`. The only remaining property to set on `DefaultLdapUserRoleListService` is `userDetailsService`. Since that was introduced in the section on LdapUserDetailsService, it will not be covered again here.

The contents of `application-pentaho-security-ldap.xml` is below. Notice that some of the bean references refer to beens defined in `applicationContext-acegi-security-ldap.xml`.

```
<bean id="allUsernamesSearch"
  class="com.pentaho.security.ldap.search.GenericLdapSearch">
  <constructor-arg index="0" ref="initialDirContextFactory" />
  <constructor-arg index="1">
    <bean
      class="com.pentaho.security.ldap.search.LdapSearchParamsFactoryImpl">
      <constructor-arg index="0" value="ou=users" />
      <constructor-arg index="1" value="objectClass=Person" />
    </bean>
  </constructor-arg>
  <constructor-arg index="2">
    <bean
      class="com.pentaho.security.ldap.transform.SearchResultToAttrValueList">
      <constructor-arg index="0" value="uid" />
    </bean>
  </constructor-arg>
</bean>

<bean id="allAuthoritiesSearch"
  class="com.pentaho.security.ldap.search.GenericLdapSearch">
  <constructor-arg index="0" ref="initialDirContextFactory" />
  <constructor-arg index="1">
    <bean
      class="com.pentaho.security.ldap.search.LdapSearchParamsFactoryImpl">
      <constructor-arg index="0" value="ou=roles" />
      <constructor-arg index="1"
        value="objectClass=organizationalRole" />
    </bean>
  </constructor-arg>
  <constructor-arg index="2">
    <bean
      class="org.apache.commons.collections.functors.ChainedTransformer">
      <constructor-arg index="0">
        <list>
          <bean
            class="com.pentaho.security.ldap.transform.SearchResultToAttrValueList">
            <constructor-arg index="0" value="cn" />
          </bean>
          <bean
            class="com.pentaho.security.ldap.transform.StringToGrantedAuthority" />
        </list>
      </constructor-arg>
    </bean>
  </constructor-arg>
</bean>

<bean id="usernamesInRoleSearch"
  class="com.pentaho.security.ldap.search.GenericLdapSearch">
  <constructor-arg index="0" ref="initialDirContextFactory" />
  <constructor-arg index="1">
    <bean
      class="com.pentaho.security.ldap.search.LdapSearchParamsFactoryImpl">
      <constructor-arg index="0" value="ou=roles" />
      <constructor-arg index="1">
        <value>
          <![CDATA[(&(objectClass=organizationalRole)(cn={0}))]]>
        </value>
      </constructor-arg>
```

```
      </bean>
    </constructor-arg>
    <constructor-arg index="2">
      <bean
        class="com.pentaho.security.ldap.transform.SearchResultToAttrValueList">
        <constructor-arg index="0" value="roleOccupant" />
        <constructor-arg index="1" value="uid" />
      </bean>
    </constructor-arg>
  </bean>

  <bean id="ldapUserRoleListService"
    class="com.pentaho.security.ldap.DefaultLdapUserRoleListService">
    <constructor-arg index="0" ref="initialDirContextFactory" />
    <property name="allAuthoritiesSearch">
      <ref local="allAuthoritiesSearch" />
    </property>
    <property name="allUsernamesSearch">
      <ref local="allUsernamesSearch" />
    </property>
    <property name="usernamesInRoleSearch">
      <ref local="usernamesInRoleSearch" />
    </property>
    <property name="userDetailsService">
      <ref bean="userDetailsService" />
    </property>
  </bean>

  <bean id="pentahoUserRoleListService"
    class="com.pentaho.security.UserDetailsRoleListService">
    <property name="userRoleListService">
      <ref local="ldapUserRoleListService" />
    </property>
  </bean>
```

LDAP Configuration Migration

This page last changed on Mar 16, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

Changes in Search Parameters

One confusing aspect of the previous version of LDAP configuration was that related search parameter properties of `DefaultLdapRoleListAuthoritiesProvider` were not grouped together. For example, the properties named `userAttribute`, `userSearchBase`, and `allUserNamesQuery` were related. But when looking at `DefaultLdapRoleListAuthoritiesProvider`, this was not obvious. The latest version of LDAP configuration seeks to change this by grouping related properties together in `LdapSearchParams` instances.

Side-By-Side Comparison

Below is a side-by-side comparison of the previous version of LDAP configuration (left) vs. the latest version of LDAP configuration (right).

Handy Hint

For an overview of search filter syntax, see LDAP Search Filter Syntax.

Warning

Note that the end-of-line backslashes that occur in the excerpt below are present for formatting purposes only and should not be present in the actual file.

```
<bean id="initialDirContextFactory"
  class="org.acegisecurity.providers.ldap.
\
  DefaultInitialDirContextFactory">
  <constructor-arg>
    <value>ldap://localhost:10389/ou=system</
  </constructor-arg>
  <property name="managerDn">
    <value>uid=admin,ou=system</value>
  </property>
  <property name="managerPassword">
    <value>secret</value>
  </property>
```

```
<bean id="initialDirContextFactory"
  class="org.acegisecurity.ldap.DefaultInitialDirContextF
  <constructor-arg index="0"
    value="ldap://localhost:10389/ou=system"
/>
  <property name="managerDn"
value="uid=admin,ou=system" />
  <property name="managerPassword"
value="secret" />
</bean>

<bean id="allUsernamesSearch"
  class="com.pentaho.security.ldap.search.GenericLdapSear
```

```
  </bean>

  <bean id="ldapAuthoritiesPopulator"
    class="org.acegisecurity.providers.ldap.
  \
  populator.DefaultLdapRoleListAuthoritiesProvi
    <constructor-arg>
      <ref local="initialDirContextFactory"/>
    </constructor-arg>
    <constructor-arg>
      <value>ou=groups</value>
    </constructor-arg>
    <property name="groupRoleAttribute">
      <value>cn</value>
    </property>
    <property name="searchSubtree">
      <value>false</value>
    </property>
    <property name="rolePrefix">
      <value>ROLE_</value>
    </property>
    <property name="convertToUpperCase">
      <value>true</value>
    </property>
    <property name="userSearchBase">
      <value>ou=users</value>
    </property>
    <property name="userAttribute">
      <value>uid</value>
    </property>
    <property name="allUserNamesQuery">
      <value>(objectClass=inetOrgPerson)</value
    </property>
    <property name="allGroupsQuery">
      <value>(objectClass=groupOfUniqueNames)</v
    </property>
    <property name="allRolesQuery">
      <value>(objectClass=organizationalRole)</v
    </property>
    <property name="rolesSearchBase">
      <value>ou=roles</value>
    </property>
    <property
  name="includeGroupsInRolesList">
      <value>false</value>
    </property>
    <property name="userNamesInRoleQuery">
      <value>(&amp;(objectClass=organizationalR
    </property>
    <property name="rolesSearchModeByUser">
      <value>false</value>
    </property>
    <property name="groupSearchFilter">
      <value>(uniqueMember={0})</value>
    </property>
    <property name="userRoleAttributes">
      <list>
        <value>uniqueMember</value>
      </list>
    </property>
    <property name="userDnPatterns">
      <list>
        <value>uid={0},ou=users</value>
      </list>
    </property>
  </bean>

  <bean id="ldapAuthProvider"
    class="org.acegisecurity.providers.ldap.
  \
    LdapAuthenticationProvider">
    <constructor-arg>
      <bean
  class="org.acegisecurity.providers.ldap.authe
  \
      DirMgrBindAuthenticator">
        <constructor-arg>
          <constructor-arg index="0"
  ref="initialDirContextFactory" />
          <constructor-arg index="1">
            <bean
  class="com.pentaho.security.ldap.search. \
              LdapSearchParamsFactoryImpl">
              <constructor-arg index="0"
  value="ou=users" />
              <constructor-arg index="1"
  value="objectClass=Person" />
            </bean>
          </constructor-arg>
          <constructor-arg index="2">
            <bean
  class="com.pentaho.security.ldap.transform.
  \
              SearchResultToAttrValueList">
              <constructor-arg index="0"
  value="uid" />
            </bean>
          </constructor-arg>
  </bean>

  <bean id="allAuthoritiesSearch"
    class="com.pentaho.security.ldap.search.GenericLdapSear
    <constructor-arg index="0"
  ref="initialDirContextFactory" />
    <constructor-arg index="1">
      <bean
  class="com.pentaho.security.ldap.search. \
        LdapSearchParamsFactoryImpl">
        <constructor-arg index="0"
  value="ou=roles" />
        <constructor-arg index="1"
          value="objectClass=organizationalRole"
  />
      </bean>
    </constructor-arg>
    <constructor-arg index="2">
      <bean
  class="org.apache.commons.collections.functors.
  \
        ChainedTransformer">
        <constructor-arg index="0">
          <list>
            <bean
  class="com.pentaho.security.ldap.transform.
  \
              SearchResultToAttrValueList">
              <constructor-arg index="0"
  value="cn" />
            </bean>
            <bean
  class="com.pentaho.security.ldap.transform.
  \
              StringToGrantedAuthority" />
          </list>
        </constructor-arg>
      </bean>
    </constructor-arg>
  </bean>

  <bean id="usernamesInRoleSearch"
    class="com.pentaho.security.ldap.search.GenericLdapSear
    <constructor-arg index="0"
  ref="initialDirContextFactory" />
    <constructor-arg index="1">
      <bean
  class="com.pentaho.security.ldap.search. \
        LdapSearchParamsFactoryImpl">
        <constructor-arg index="0"
  value="ou=roles" />
        <constructor-arg index="1">
          <value>
            <![CDATA[(&(objectClass=organizationalRole)(cn=
          </value>
        </constructor-arg>
```

```
        <ref
local="initialDirContextFactory"/>
      </constructor-arg>
      <property name="userDnPatterns">
        <list><value>uid={0},ou=users</value>
      </property>
    </bean>
  </constructor-arg>
  <constructor-arg>
    <ref local="ldapAuthoritiesPopulator"
/>
  </constructor-arg>
</bean>
```

```
    </bean>
  </constructor-arg>
  <constructor-arg index="2">
    <bean
class="com.pentaho.security.ldap.transform.
\
      SearchResultToAttrValueList">
      <constructor-arg index="0"
value="roleOccupant" />
      <constructor-arg index="1"
value="uid" />
    </bean>
  </constructor-arg>
</bean>

<bean id="ldapUserRoleListService"
  class="com.pentaho.security.ldap.DefaultLdapUserRoleLis
  <constructor-arg index="0"
ref="initialDirContextFactory" />
  <property name="allAuthoritiesSearch">
    <ref local="allAuthoritiesSearch" />
  </property>
  <property name="allUsernamesSearch">
    <ref local="allUsernamesSearch" />
  </property>
  <property name="userDetailsService">
    <ref bean="userDetailsService" />
  </property>
  <property name="usernamesInRoleSearch">
    <ref local="usernamesInRoleSearch" />
  </property>
</bean>

<bean id="ldapAuthProvider"
  class="org.acegisecurity.providers.ldap.
\
  LdapAuthenticationProvider">
  <constructor-arg>
    <bean
class="org.acegisecurity.providers.ldap.authenticator.
\
    BindAuthenticator">
      <constructor-arg>
        <ref
local="initialDirContextFactory" />
      </constructor-arg>
      <property name="userSearch">
        <ref local="userSearch" />
      </property>
    </bean>
  </constructor-arg>
  <constructor-arg>
    <ref local="populator" />
  </constructor-arg>
</bean>

<bean id="populator"
class="org.acegisecurity.providers.ldap.populator.
\
  DefaultLdapAuthoritiesPopulator">
  <constructor-arg index="0">
    <ref local="initialDirContextFactory"
/>
  </constructor-arg>
  <constructor-arg index="1"
value="ou=roles" />
  <property name="groupRoleAttribute"
value="cn" />
  <property name="groupSearchFilter"
value="roleOccupant={0}" />
</bean>

<bean id="userSearch"
  class="org.acegisecurity.ldap.search.FilterBasedLdapUse
  <constructor-arg index="0"
value="ou=users" />
  <constructor-arg index="1" value="cn={0}"
```

```
  />
    <constructor-arg index="2">
      <ref local="initialDirContextFactory"
  />
    </constructor-arg>
  </bean>

  <bean id="userDetailsService"
    class="com.pentaho.security.ldap.LdapUserDetailsService
    <property name="userSearch">
      <ref local="userSearch" />
    </property>
    <property name="populator">
      <ref local="populator" />
    </property>
  </bean>
```

LDAP Search Filter Syntax

This page last changed on Mar 16, 2007 by mlowery.

> **Useful Information**
>
> Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

RFC 2254 defines the query syntax for a directory service. The full syntax can be found in that specification. However, an overview of the most frequently used forms is given below.

Basic Form

A search filter contains one or more filter components where each component has one of four forms. In the table below, `attribute` is a property of an object within the directory and `value` is a string to match.

Form	Description
`(attribute=value)`	Returns the objects where `attribute` **EQUAL TO** `value`.
`(&(attribute1=value1)(attribute2=value2))`	Returns the objects where `attribute1` equal to `value1` **AND** `attribute2` equal to `value2`.
`(|(attribute1=value1)(attribute2=value2))`	Returns the objects where `attribute1` equal to `value1` **OR** `attribute2` equal to `value2`.
`(!(attribute=value))`	Returns the objects where `attribute` is **NOT EQUAL TO** `value`.

Note that the specification defines three other operators in addition to equals: ~=, >=, and <=. These operators are not covered here.

> **Useful Information**
>
> You might see search filters defined in the platform that contain `{n}` where n is some integer. Note that this syntax is not part of the search filter specification. Instead, it is a placeholder using `MessageFormat` syntax. The platform substitutes the values of the filter arguments into this filter expression and the resulting search filter is compliant with the search filter specification. That is the string that is sent to the directory service.

Wildcards

Values can contain asterisks. An asterisk represents zero or more characters. When it appears as the only

character on the right hand side, it can be used as a test for the presence of an attribute.

Examples

Example	Description
`(cn=joe)`	Return all objects where attribute `cn` has the value `joe`.
`(&(objectClass=person)(cn=joe))`	Return all objects where attribute `objectClass` has the value `person` and attribute `cn` has the value `joe`.
`(|(objectClass=person)(objectClass=organizationalRole))`	Return all objects where attribute `objectClass` has the value `person` or the value `organizationalRole`.
`(!(cn=joe))`	Return all objects where attribute `cn` does not have the value `joe`.
`(cn=*)`	Return all objects where attribute `cn` is present.
`(cn=*Smith)`	Return all objects where attribute `cn` ends with "Smith".

02. Authentication

This page last changed on Apr 09, 2007 by gmoran.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

The act of processing a submitted username and password is called authentication. Note that authentication is a prerequisite to authorization. The Pentaho BI Platform uses Acegi Security to process authentication requests. Out-of-the-box authentication mechanisms provided by the platform are form, basic, and request parameter.

The vast majority of the configuration contained in the `applicationContext-acegi-security.xml` is a standard Acegi Security setup and is well-documented in the Acegi Security documentation. Where the configuration strays from the Acegi Security documentation, it is documented below.

Form-Based Authentication

Form-based authentication lets developers customize the authentication user interface. While the J2EE specifications provide a standard way to specify the login page URL as well as URL authorization rules, there is still container-specific configuration to specify how to read usernames and passwords from a security datastore. This is one reason that the platform uses Acegi Security. The Acegi Security class that processes form posts is `AuthenticationProcessingFilter`.

Login Handling

SecurityStartupFilter

`SecurityStartupFilter` allows the Pentaho BI Platform to obtain a user's credentials (`java.security.Principal`) and inject it into the Pentaho user session. This requires a new bean definition:

```
<bean id="pentahoSecurityStartupFilter"
  class="com.pentaho.security.SecurityStartupFilter" />
```

This bean is then added to the `filterChainProxy` bean (shown later).

HttpSessionReuseDetectionFilter

`HttpSessionReuseDetectionFilter` detects when an HTTP session which contains a authenticated user is attempting to authenticate again without logging out. Upon detecting this condition, the session is invalidated, the security context is cleared, and the user is redirected to `sessionReuseDetectedUrl`. This prevents reuse of an HTTP session which contains potentially sensitive, user-specific data. The `filterProcessesUrl` value should match the value of the same property in `AuthenticationProcessingFilter`.

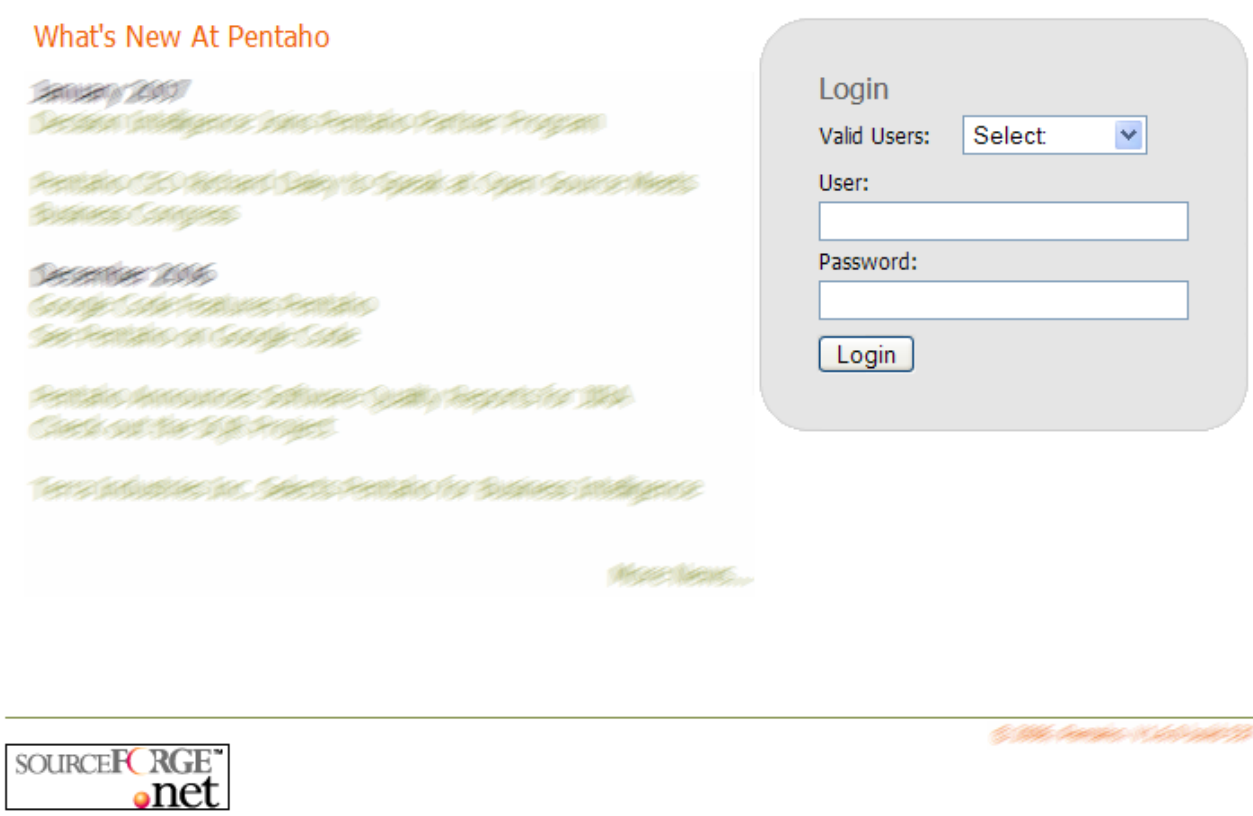
Notice the `login_error=2` parameter on the `filterProcessesUrl`? The login page should test for `login_error=2` and print the appropriate message describing what just happened.

```
<bean id="httpSessionReuseDetectionFilter"
class="com.pentaho.security.HttpSessionReuseDetectionFilter">
  <property name="filterProcessesUrl" value="/j_acegi_security_check" />
  <property name="sessionReuseDetectedUrl" value="/Login?login_error=2" />
</bean>
```

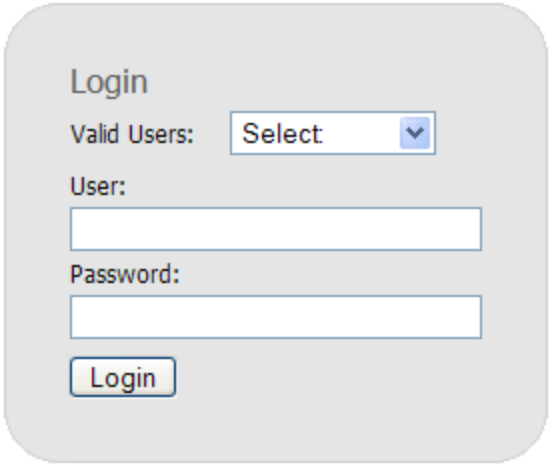
Login Page

Below are some screenshots of the login page in different states. To customize this page, including changing strings, see Customizing the Login Page.

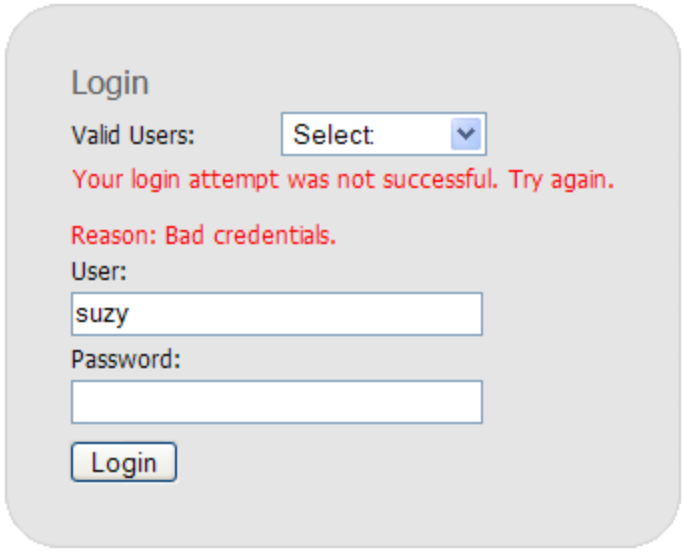
Login Page



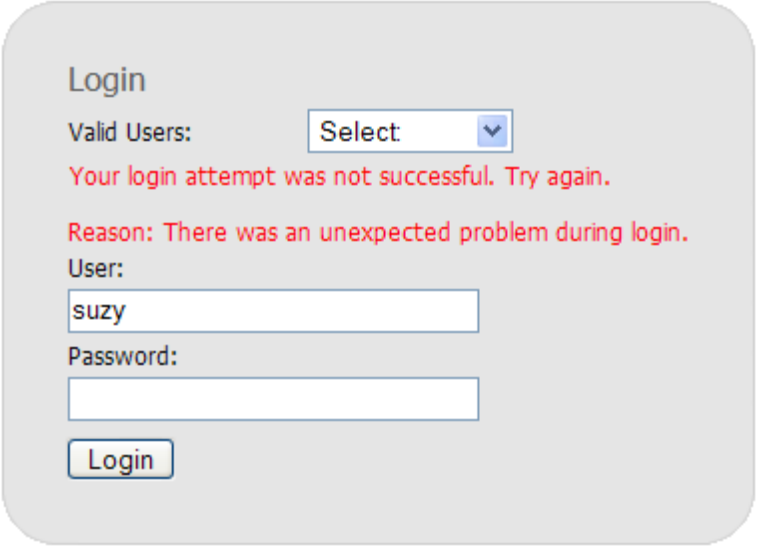
Blank Login Form



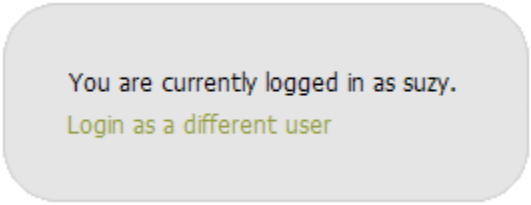
Login Form After Bad Credentials Submitted



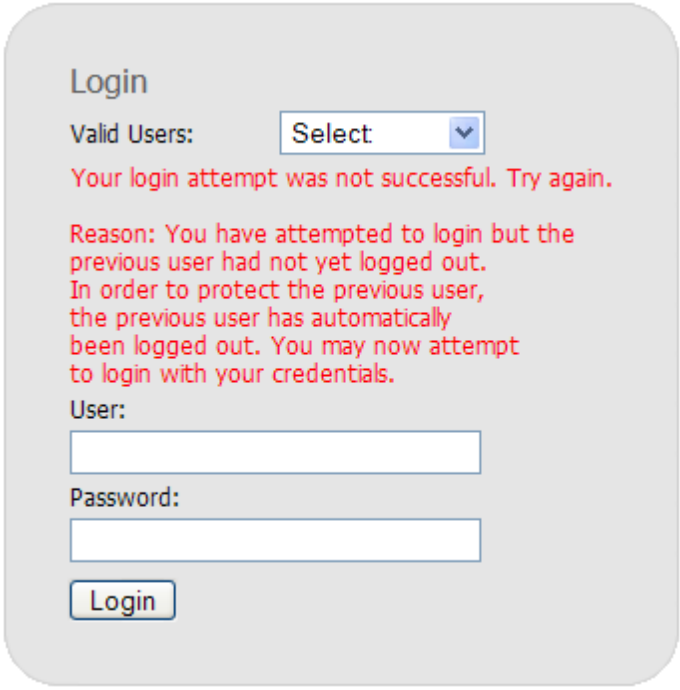
Login Form After Generic Security Error



Login Form While Logged In



Login Form After Session Re-Use Detected



`HttpSessionReuseDetectionFilter`

Logout Handling

ProPentahoLogoutHandler

`ProPentahoLogoutHandler` executes various cleanup tasks when the user logs out.

```
<bean id="logoutFilter"
  class="org.acegisecurity.ui.logout.LogoutFilter">
  <constructor-arg value="/index.jsp" />
  <constructor-arg>
    <list>
      <bean
        class="com.pentaho.security.ProPentahoLogoutHandler" />
        <ref bean="rememberMeServices" />
      <bean
        class="org.acegisecurity.ui.logout.SecurityContextLogoutHandler" />
    </list>
  </constructor-arg>
  <property name="filterProcessesUrl" value="/Logout" />
</bean>
```

Logout Page

There is no logout page. The page to which a user is redirected after a logout is specified in the first constructor argument in the `logoutFilter` bean above.

Basic Authentication

Basic authentication is part of the HTTP specification. It is simple but relatively inflexible. Acegi Security implements Basic authentication using `BasicProcessingFilter` and `BasicProcessingFilterEntryPoint`.

Request Parameter Authentication

`RequestParameterAuthenticationFilter` provides security services for Pentaho Spreadsheet Services (PSS). It allows the user requesting access to provide his or her username and password on the query string of the URL. The credentials are unencrypted.

The parameters to pass on the query string are:

- userid=value - the userid to authenticate
- password=value - the user's password (clear-text)

RequestParameterAuthenticationFilter

`RequestParameterAuthenticationFilter` provides security services for Pentaho Spreadsheet Services (PSS). If you are using PSS, add this filter, along with the associated `RequestParameterFilterEntryPoint` bean to your Spring config.

```
<bean id="requestParameterProcessingFilter"
  class="com.pentaho.security.RequestParameterAuthenticationFilter">
  <property name="authenticationManager">
    <ref local="authenticationManager" />
  </property>
  <property name="authenticationEntryPoint">
    <ref local="requestParameterProcessingFilterEntryPoint" />
  </property>
</bean>

<bean id="requestParameterProcessingFilterEntryPoint"
  class="com.pentaho.security.RequestParameterFilterEntryPoint" />
```

FilterChainProxy

The `FilterChainProxy` with the Pentaho BI Platform filters is shown below.

Warning

Note that the end-of-line backslashes that occur in the excerpt below are present for formatting

purposes only and should not be present in the actual file.

Warning

Note that the `pentahoSecurityStartupFilter` needs to be preceded by the `httpSessionContextIntegrationFilter`. Otherwise, when the Pentaho startup filter is triggered, the `java.security.Principal` will not be in the session and will fail.

```
<bean id="filterChainProxy"
  class="org.acegisecurity.util.FilterChainProxy">
  <property name="filterInvocationDefinitionSource">
    <value>
      <![CDATA[CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
      PATTERN_TYPE_APACHE_ANT
      /**=securityContextHolderAwareRequestFilter,httpSessionContextIntegrationFilter, \
      httpSessionReuseDetectionFilter,logoutFilter,authenticationProcessingFilter, \
      basicProcessingFilter,requestParameterProcessingFilter,rememberMeProcessingFilter, \
      anonymousProcessingFilter,pentahoSecurityStartupFilter,switchUserProcessingFilter, \
      exceptionTranslationFilter,filterInvocationInterceptor]]>
    </value>
  </property>
</bean>
```

Single Sign-On

This page last changed on Apr 22, 2007 by mlowery.

This document describes how single sign-on (SSO) between (1) the servlet-based user interface and (2) the portlet-based user interface of the Pentaho BI Platform was implemented. The implementation introduced a third web application, the Central Authentication Service. In the pages that follow, the configuration of each of the three web applications below will discussed.

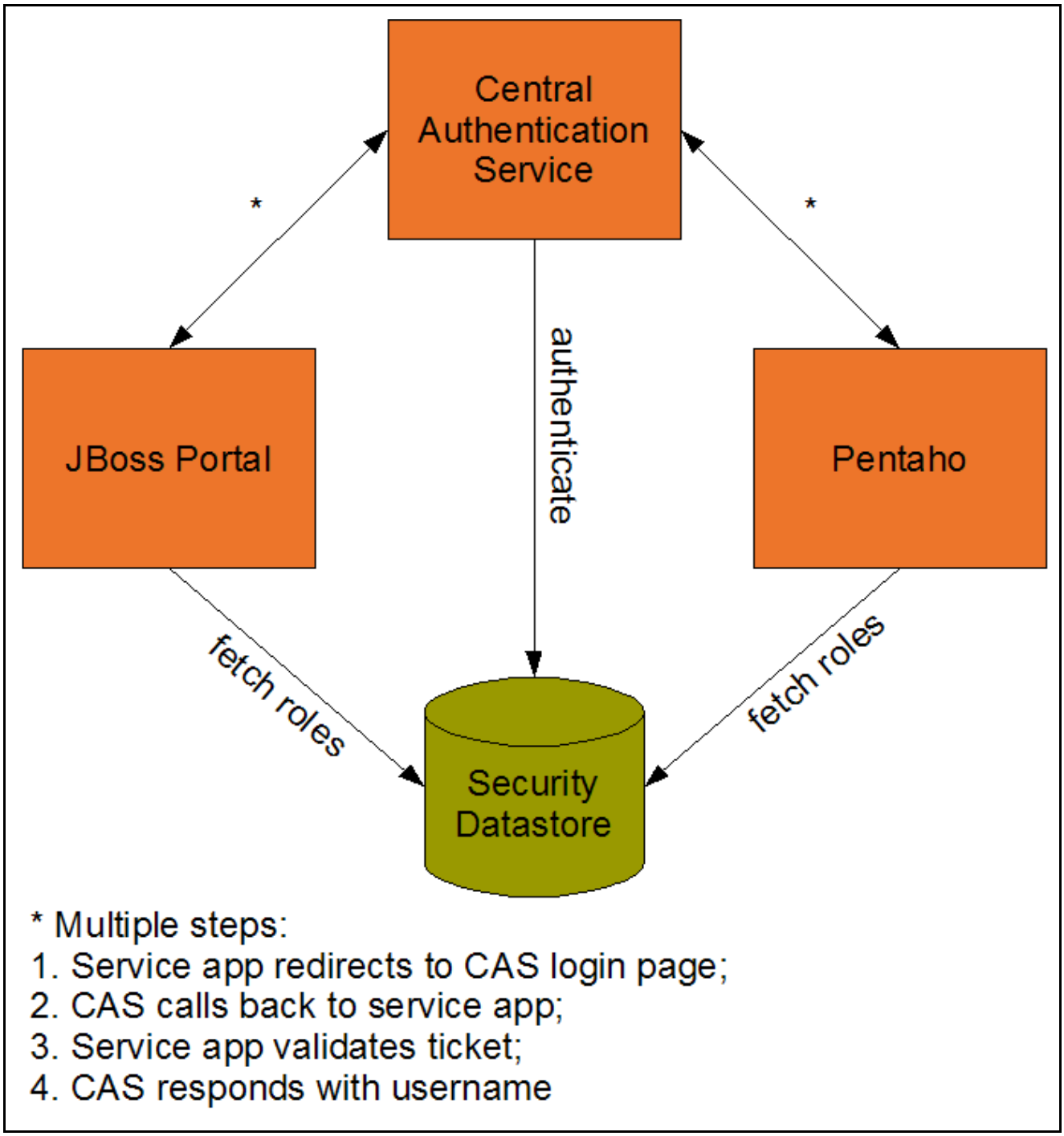
1. Central Authentication Service (CAS): the single sign-on server
2. JBoss Portal: the container in which the portlet-based user interface runs
3. Pentaho BI Platform: the servlet-based user interface

Before going into detail on configuring each of the three web applications to participate in the single sign-on system, let's first introduce two enabling technologies: CAS and Acegi Security.

Central Authentication Service

CAS is a single sign-on system. When users explicitly attempt to login (also known as *authenticate*) or when users request a resource which requires authentication, they are redirected to the CAS application. It alone handles the username and password submitted by the user. Upon successful login, CAS returns the user to the resource originally requested. It is up to the application containing the requested resource to grant or deny access based on authorization rules inside that application. Note that CAS provides only the name of the authenticated user to each application; it is up to each application to fetch the roles belonging to the authenticated user. Once it has fetched the roles belonging to an authenticated user, it can make authorization decisions based on those roles.

CAS at a Glance



In the above diagram, a "service app" refers to a "client" of the Central Authentication Service; it relies on CAS to authenticate users for it. Also note that the backing database used by CAS to check usernames and passwords is not necessarily the same backing database used by client applications to fetch roles, although in the above diagram they are the same.

Acegi Security

Acegi Security is a security framework written to take advantage of the Spring Framework's dependency injection. Among other benefits, Acegi Security provides connectors to various backing databases allowing for the fetching of usernames, passwords, and roles. While all three of the web applications

above use Acegi Security, the degree to which Acegi Security is used varies. For now, it is sufficient to know that Acegi Security was chosen as it allows for consistent, declarative security configuration among the applications.

01. Central Authentication Service

This page last changed on May 15, 2007 by mlowery.

Central Authentication Service

In order to implement single sign-on, a Central Authentication Service (CAS) instance was employed. The version currently used by Pentaho is 3.0.5. The high-level steps below describe how to enable services required by CAS (i.e. SSL), how to prepare the CAS server for building, and finally how to build and deploy the CAS server.

Setup SSL

CAS requires SSL. See Enabling SSL in Tomcat.

Follow the *Optional CAS Server Setup* instructions in the Acegi Security reference manual.

In Pentaho's setup of CAS, CAS uses Acegi Security classes to connect to back-end databases for security information. The complete instructions for this are outlined in the CAS section of the Acegi Security reference.

Copy Pentaho's Acegi Security updates

Pentaho has provided a modified version of the Acegi Security class named `DefaultLdapAuthoritiesPopulator`. The modified class is called (not very descriptively) `com.pentaho.security.ldap.FixedDefaultLdapAuthoritiesPopulator`. This modification is detailed in Acegi Security's issue tracking system as SEC-450. Finally, Pentaho has provided an LDAP-based implementation of `UserDetailsService` called `com.pentaho.security.ldap.LdapUserDetailsService`.

Modify CAS logging

In order not to conflict with JBoss' logging mechanism, the default `log4j.properties` in `cas-server-??/webapp/WEB-INF/classes` needs to be removed and replaced with an equivalent `log4j.xml`. Finally, modify the `log4jConfigLocation context-param` in CAS' `web.xml` to match the new `log4j.xml`.

Warning

Be very careful when modifying the log settings in `cas.war/WEB-INF/classes/log4j.xml`. If you set `org.jasig` classes to `DEBUG` level, it is possible that user passwords will be logged "in the clear."

Enable Pentaho theme

Because a CAS server is responsible for prompting for a username and password, it has its own user interface. That interface is skinnable. The instructions for customizing the look of CAS are documented in Customizing Views.

Build

CAS comes with an Ant script for building. Run `cas-server-??/localPlugins/build.xml`.

Other Resources

SSL HOWTO from Acegi Security.

Enabling SSL in Tomcat

This page last changed on Apr 22, 2007 by mlowery.

SSL Overview

For the purpose of this document, SSL, and its successor TLS, are identical. SSL provides security in several ways, including:

- Verification of server identity.
- Encryption of data between client and server.

A frequent source of trouble in setting up SSL is the configuration of the server certificate. Below is an breakdown of the parties involved to get SSL setup.

- certificate issuer, also known as certificate authority: A certificate issuer is a company that vouches for the identity of the server. The company vouches for the server by signing the server's certificate with its private key. Because everyone in the world has access to the certificate issuer's public key, they can verify that the certificate was in fact signed by the certificate issuer.
- certificate recipient: A certificate recipient is the web server. It's the server to which a user's browser will connect via HTTPS.
- browser: The browser receives the certificate from the server and will only proceed without user interaction if the signer of the certificate (the certificate authority) is "trusted." You'll receive a browser warning dialog window if the signer of the certificate is not trusted. This can happen when the certificate is "self-signed." This means that the certificate recipient signed (using its private key) its own certificate. This is only appropriate for development and should not be used in production.

Tomcat Setup (Using a Self-Signed Certificate)

Enabling SSL in Tomcat follows the Tomcat documentation. Where the Pentaho setup deviates from the Tomcat documentation, it is documented below.

Creating a Self-Signed Certificate

When creating the certificate, you will be prompted with `What is your first and last name?` Enter just the word `localhost` at that prompt. Otherwise the `HostnameVerifier` will fail.

Trusting the Self-Signed Certificate

Why does one need to trust the certificate? Usually, only clients that are connecting to servers via https need to trust the certificate of the server. And while the client (the web browser in one case) must trust the certificate of the CAS server, there is another client that must trust the CAS server. That client is the web application using CAS services. In the case of Pentaho, there are two web applications--the servlet

interface (`pentaho.war`) and the portlet interface. Both of these web applications connect via https to the CAS server during the ticket validation process. See section 2.6. in the CAS Protocol documentation.

If you do not trust the certificate, you'll get a `sun.security.validator.ValidatorException: No trusted certificate found` error.

1. Execute the following in `%USER_HOME%`:

```
keytool -export -alias tomcat -file tomcat.cer -storepass changeit -keypass changeit -keystore .keystore
```

2. Execute the following in `%JAVA_HOME%/jre/lib/security`:

```
keytool -import -alias tomcat -file tomcat.cer -keystore cacerts -storepass changeit
```

3. Now confirm that the `tomcat` entry in `%USER_HOME%/.keystore` is the same entry that is in `%JAVA_HOME%/jre/lib/security/cacerts`. Do this by comparing the fingerprints of the two entries.
   a. Execute the following in `%USER_HOME%`:

```
keytool -list -keystore .keystore
```

Output of keytool -list -keystore .keystore

```
Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry

tomcat, Mar 1, 2007, keyEntry,
Certificate fingerprint (MD5): xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx
```

   b. Note the fingerprint of the `tomcat` entry. Execute the following in `%JAVA_HOME%/jre/lib/security`:

```
keytool -list -keystore cacerts
```

Output of keytool -list -keystore cacerts

```
Keystore type: jks
Keystore provider: SUN

Your keystore contains n entries

entries omitted
tomcat, Mar 1, 2007, trustedCertEntry,
```

```
Certificate fingerprint (MD5): xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx
entries omitted
```

c. Make sure that the fingerprint for the `tomcat` entry in `cacerts` is the same as the `tomcat` entry in `.keystore`.

Enabling SSL

Add the following to `server/default/deploy/jbossweb-tomcat55.sar/server.xml` (making sure that this connector does not already exist):

```
<Connector port="8443" minProcessors="5" maxProcessors="75"
enableLookups="true" disableUploadTimeout="true" acceptCount="100"
debug="0" scheme="https" secure="true" clientAuth="false" sslProtocol="TLS" />
```

Consideration for Mozilla Browsers

Some cryptographic algorithms used by Mozilla browsers during SSL connections are not provided by the JVM. The solution is to add other security providers.

If using Sun JRE 1.4.2:

1. Download the Legion of the Bouncy Castle security provider.
2. Copy the JAR into `jre/lib/ext` directory of the JRE running Tomcat.
3. Edit `jre/lib/security/java.security` and add `security.provider.n=org.bouncycastle.jce.provider.BouncyCastleProvider` where `n` is the highest unused integer available in the file.

If using Sun JRE 1.5.n:

1. Copy the Sun JCE provider from `{Program Files}/Java/j2re1.5.n_nn/lib/ext/sunjce_provider.jar` file into the `jre/lib/ext` of the JRE running Tomcat.

02. JBoss Portal

This page last changed on Apr 22, 2007 by mlowery.

JBoss Portal allows for declaration of authorization rules on portlet instances, portal instances, and pages. Because JBoss Portal can apply security at a finer granularity than simply a URL, the decision was made to use JBoss Portal's native mechanisms for security. This allows the portal container to do enforcement in a way that would be difficult to reproduce using Acegi Security to secure URLs.

Login Page

Since CAS processes username and password submittals, the login page of JBoss Portal was modified so that it immediately redirects to the CAS login page. Edit `pentaho-preconfiguredinstall/server/default/deploy/jboss-portal.sar/portal-server.war/login.jsp` to contain the redirect. (Use the code in `CasProcessingFilterEntryPoint` as a guide.

Login Processing

Remember that when CAS is present, "client" applications no longer process usernames and password submittals. Instead, client applications process ticket submittals. These tickets are one-time tokens that must be validated back on the CAS server. Essentially, when a ticket comes into a client application, the client application asks the CAS server if the ticket is valid. If it is, the client application authenticates the user.

Transforming the authentication

After a user successfully logs into the CAS server, the CAS server redirects the user back to the client application with a ticket. That ticket is then used as the `j_password` parameter in a Java EE form-based authentication. `CasContainerAdapterFilter` handles this transformation between ticket callback from CAS and Java EE form-based authentication. This filter is defined in `server/default/deploy/jboss-portal.sar/portal-server.war/web.xml`.

Validating the ticket

JBoss Portal uses the same authentication mechanism as JBoss Application Server--JAAS. Acegi Security provides a `LoginModule` implementation for JBoss Application Server called `JbossAcegiLoginModule`. Pentaho has created a slightly customized version of `JbossAcegiLoginModule` called `Jboss4AcegiLoginModule`. It simply follows the instructions in the JBoss wiki. To specify that JBoss Portal should use the Pentaho `Jboss4AcegiLoginModule`, edit `server/default/deploy/jboss-portal.sar/conf/login-config.xml`. See the Acegi Security documentation for the JBoss adapter.

Useful Information

Jboss4AcegiLoginModule overrides `JbossAcegiLoginModule` to add the `CallerPrincipal` group. When using a custom principal class (in this case `PrincipalAcegiUserToken`), JBoss requires a special group for the subject. From the JBoss wiki, "A custom principal must be installed under the

Subject using a java.security.acl.Group named CallerPrincipal with the sole group member being the custom principal instance."

Why doesn't Acegi Security include the CallerPrincipal in their JbossAcegiLoginModule? It might be because this issue only seems to arise when the j_username passed into j_security_check is not the final username to use (as is the case with CAS). In CAS, the username is initially _cas_stateful_ until the CAS ticket is validated and the CAS server sends back the true username.

Useful Information

Why isn't JbossIntegrationFilter used? In the portal side, there are no references to Acegi Security Authentication objects. Therefore, this filter is not needed.

Jboss4AcegiLoginModule only adapts JBoss Application Server's authentication mechanism to Acegi Security's authentication mechanism. It is still necessary to define the proper authentication handler--CasAuthenticationProvider. This class validates the ticket from the CAS server. Jboss4AcegiLoginModule is configured as documented in the Acegi Security documentation for the JBoss adapter. As is usually the case, the Acegi Security class CasAuthenticationProvider is configured via Spring and the Spring files that facilitate this are located in server/default/deploy/jboss-portal.sar/lib/pentaho-security-config.jar. The actual JAR containing Jboss4AcegiLoginModule is in server/default/lib/pentaho-jboss-login-module.jar.

For the login module to work, Acegi Security and Spring JARs should be present in server/default/lib.

Fetching Roles

Because CAS does not return roles, all client applications must define a CasAuthoritiesPopulator. This class delegates to a UserDetailsService to fetch roles belonging to the user which CAS has previously authenticated. Remember that the UserDetailsService implementations provided in the platform are InMemoryDaoImpl, JdbcDaoImpl, and LdapUserDetailsService. These are described in Security Data Access Objects. The Spring config files to configure the CasAuthoritiesPopulator are located in server/default/deploy/jboss-portal.sar/lib/pentaho-security-config.jar.

JBoss Portal defines two roles out-of-the-box:

1. User: Given to all users.
2. Admin: Required to view administrative portlets.

In addition to these roles, JBoss Portal uses a third role that should be granted to all authenticated users. That role is Authenticated. Be careful that you set rolePrefix and convertToUpperCase properties to "" and "false" respectively on UserDetailsService implementations (and RoleVoter.

Disabling Edit Profile

JBoss Portal provides the ability for authenticated users to edit their user records. The platform has

disabled this functionality as it requires that code be written to implement JBoss Portal-specific interfaces--namely `UserModule` and `RoleModule`. This disabling is accomplished by removing the edit profile link in `server/default/deploy/jboss-portal.sar/portal-core.war/WEB-INF/jsp/user/menu.jsp`.

Logout

In order to logout of CAS at the same time as a logout from JBoss Portal, a redirect to the CAS logout page should be specified when logout is clicked by the portal user. This is accomplished by modifying `server/default/deploy/jboss-portal.sar/portal-core.war/WEB-INF/jsp/user/menu.jsp`.

Required Libraries

The Acegi Security JAR will need to be available in `server/default/lib`. Additionally, the JARs containing CasContainerAdapter and Jboss4LoginModule will need to be available in `server/default/lib`. Finally, a JAR containing Pentaho Acegi Security updates will need to be available in `server/default/lib`.

03. Pentaho BI Platform

This page last changed on Apr 22, 2007 by mlowery.

The Pentaho BI Platform makes use Acegi Security's CAS integration. Enabling this is fairly straightforward according to the Acegi Security documentation.

Logout

In order to destroy the SSO cookie, you will need to make the `logoutSuccessUrl` equal to the CAS logout page. Note that the `logoutSuccessUrl` on the `LogoutFilter` and the `loginUrl` on the `CasProcessingFilterEntryPoint` should use the same `scheme://host:port`.

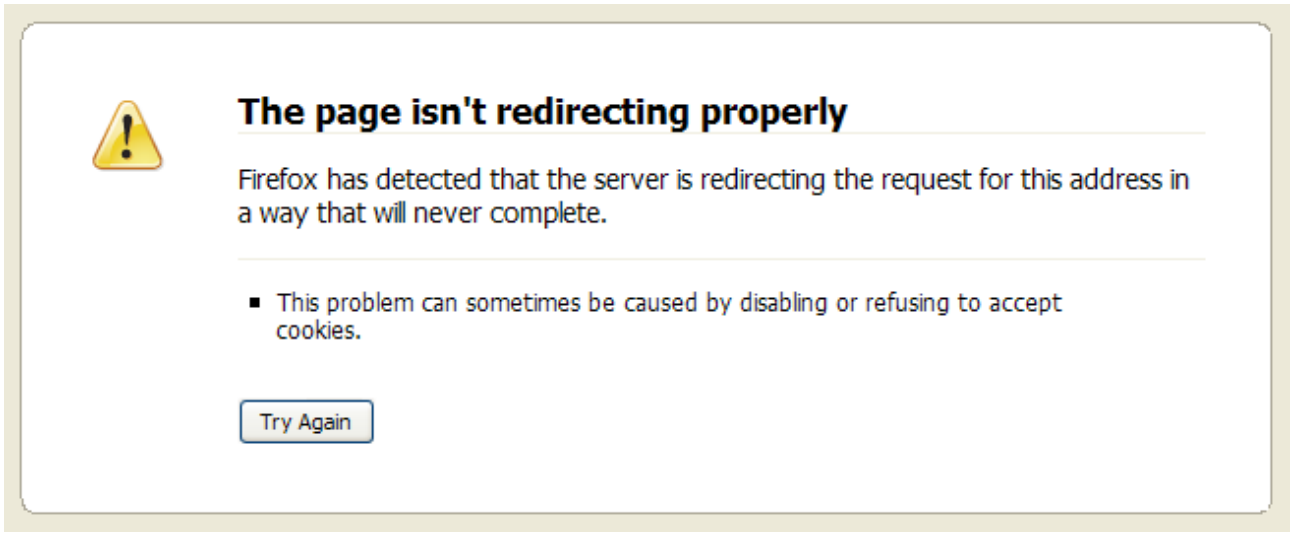
04. Troubleshooting

This page last changed on Mar 11, 2007 by mlowery.

Redirect Loop

The Firefox error below can occur when your SSL certificate is not setup correctly or when your Acegi Security config is incorrect. (Internet Explorer also fails but the error message is generic.)

Firefox Error Message



Essentially, Firefox has detected an infinite loop of redirects. Causes of this problem can be:

- This can happen with self-signed certificates and is due to the fact that the `tomcat` entry in the keystore used by Tomcat (as specified in `server.xml`) is not the same entry in `cacerts`. In other words, the `tomcat` entry is not trusted by the JVM. Follow the steps outlined in Trusting the Certificate to verify that the fingerprints match.
- This can also occur when you do not have a `CasAuthenticationProvider` defined in your config.

More than one entry in %USER_HOME%/.keystore

The Tomcat documentation states that the `keyAlias` attribute is required (on the `Connector` element in `server.xml`) if you have more than one entry in `%USER_HOME%/.keystore`.

> "Add this element [`keyAlias`] if your have more than one key in the KeyStore. If the element is not present the first key read in the KeyStore will be used."

03. Web Resource Authorization

This page last changed on Mar 16, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

Protecting URLs

If one attempted to differentiate between web resource authorization and domain object authorization, one could say that web resource authorization is more coarse-grained. It protects web resources, all of which are uniquely identified by a URL. URLs can point to static resources like images or they can point to dynamic resources such as the pages of a web application. Web resource authorization, as used in this document, deals with the latter. Web security is referred to as coarse-grained since web resource authorization doesn't enforce security on methods or even instances that are involved in dynamically creating a web page. That's not to say that one can't have finer grain control using domain object authorization--it's just that web resource authorization is the first security gate through which a user must pass.

Protecting URLs with Acegi Security

The Pentaho BI Platform comes out-of-the-box using a configuration setup very similar to the Contacts Sample Application. This sample comes with the Acegi Security download. The platform uses a standard Acegi Security setup that is well-documented in the Acegi Security documentation.

Below, a `FilterSecurityInterceptor` is defined along with an `AccessDecisionManager`. The two beans are associated through the `accessDecisionManager` property. The `objectDefinitionSource` property associates URL patterns with the role required to view pages that match the URL pattern. `RoleVoter` specifies that if any role on the right hand side of the equals is granted to the user, the user may view any page that matches that URL pattern.

```
<bean id="filterInvocationInterceptor"
  class="org.acegisecurity.intercept.web.FilterSecurityInterceptor">
  <property name="authenticationManager">
    <ref local="authenticationManager" />
  </property>
  <property name="accessDecisionManager">
    <ref local="httpRequestAccessDecisionManager" />
  </property>
  <property name="objectDefinitionSource">
    <value>
    <![CDATA[
    CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
    PATTERN_TYPE_APACHE_ANT
```

```
      /login*=ROLE_ANONYMOUS,ROLE_AUTHENTICATED
      /j_acegi_security_check*=ROLE_ANONYMOUS,ROLE_AUTHENTICATED
      /getmondrianmodel*=ROLE_ANONYMOUS,ROLE_AUTHENTICATED
      /getimage*=ROLE_ANONYMOUS,ROLE_AUTHENTICATED
      /admin*=ROLE_ADMIN
      /auditreport*=ROLE_ADMIN
      /auditreportlist*=ROLE_ADMIN
      /versioncontrol*=ROLE_ADMIN
      /propertieseditor*=ROLE_ADMIN
      /propertiespanel*=ROLE_ADMIN
      /subscriptionadmin*=ROLE_ADMIN
      /logout*=ROLE_ANONYMOUS
      /**=ROLE_AUTHENTICATED
      ]]>
      </value>
    </property>
  </bean>

  <bean id="httpRequestAccessDecisionManager"
    class="org.acegisecurity.vote.AffirmativeBased">
    <property name="allowIfAllAbstainDecisions" value="false" />
    <property name="decisionVoters">
      <list>
        <ref bean="roleVoter" />
      </list>
    </property>
  </bean>
```

```
  <bean id="roleVoter" class="org.acegisecurity.vote.RoleVoter" />
```

04. Domain Object Authorization

This page last changed on Mar 19, 2007 by sbarkdull.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

Protecting Action Sequences

In a software system, you can secure elements of that system at different levels, depending on your needs. In a web application, you can secure specific URLs. Deeper in the application you might want to secure specific service method calls. And finally, you might want to secure particular instances of objects. This page talks about the last type of security. Users of the Pentaho platform might wish to have a very precise level of control over objects in their solution repository. The Pentaho BI Platform provides this control.

Useful Information

This page describes key security classes in the Pentaho BI Platform. Unless otherwise noted, these classes can be found in `com.pentaho.security`. Class packages will be omitted in the discussion below.

Security in the platform is based in part on the Acegi Security System for Spring. Classes that are part of Acegi Security are marked with Acegi.

Access Control Lists

In the Pentaho BI Platform, objects in the solution repository (e.g. files and directories) can be secured using access control lists (ACLs). You can have any number of entries in an ACL--each specifying a different recipient.

ACL Entries

An entry in an access control list consists of a recipient, permissions, a reference to the object to which the ACL entry applies, and optionally the parent of the object to which the ACL entry applies. The default ACL entry type in Pentaho is `PentahoAclEntry`. This class extends `AbstractBasicAclEntry` Acegi.

Recipients

PentahoAclEntry stores a recipient as an Object. In practice, recipients can be of two types: a String containing a username or a GrantedAuthority containing a granted authority.

Permissions

PentahoAclEntry stores permissions using bit masks.

Objects and Parents

PentahoAclEntry stores an object (and its parent) as a AclObjectIdentity Acegi.

ACL Holders

An IAclHolder does exactly what its name implies--it holds or contains an access control list. An ACL is implemented in the platform using a java.util.List. Inside this list are implementations of AclEntry Acegi

Solution Repository Objects

Once you have a container for an ACL, how is it associated with objects in the solution repository? That is where the interface IAclSolutionFile comes in. This interface extends IAclHolder and is implemented by com.pentaho.repository.dbbased.solution.RepositoryFile. RepositoryFile also implements AclObjectIdentity. So not only does a RepositoryFile store an ACL (since it implements IAclHolder), it also is a securable object (since it implements AclObjectIdentity).

Persistence

The Pentaho BI Platform uses Hibernate for reading and writing to the db-based repository. The PRO_FILES table contains solution repository objects while the PRO_ACLS_LIST table contains ACL entries associated with those objects. Below are (incomplete) listings of the columns of each of these tables.

PRO_FILES Table

FILE_ID	PARENT	FILENAME	FULLPATH	DATA	DIRECTORY	LASTMODIFIED

FILE_ID is the primary key. PARENT is a reference (by file id) to the object's parent. DIRECTORY is a boolean that is true if this object is a directory and false if this object is a file.

PRO_ACLS_LIST Table

ACL_ID	ACL_MASK	RECIPIENT

Technically, rows in this table represent ACL entries, not ACLs. An ACL for an object can be created by

querying for all rows sharing the same `ACL_ID`. `ACL_ID` is a foreign key that references `PRO_FILES.FILE_ID`. `ACL_MASK` is the decimal representation of the bit mask that represents the permissions in this ACL entry. And `RECIPIENT` is the username or granted authority that is the recipient of this ACL entry.

Voters

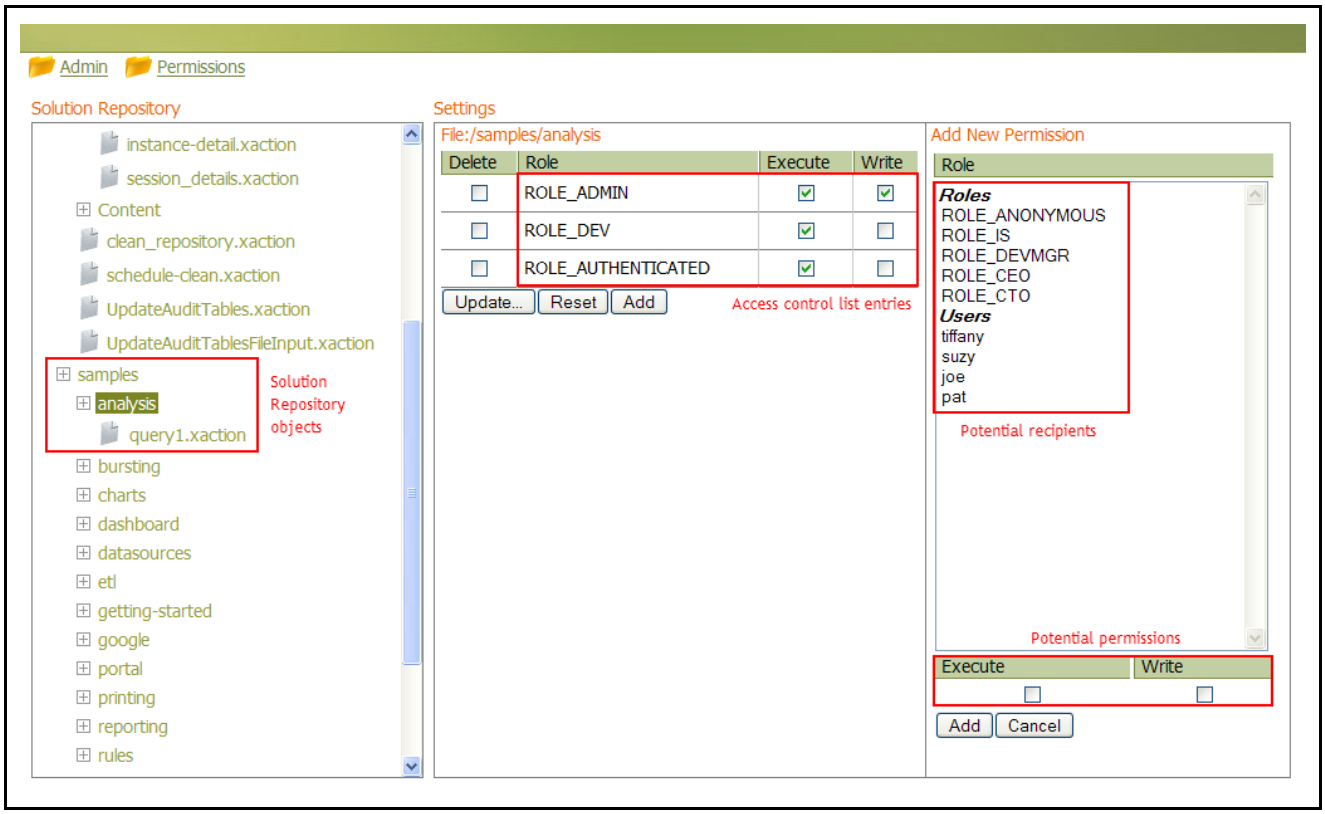
For every domain object, there is exactly one access control list. Add to that a user that wants to perform some operation on that object and that adds up to three inputs: a recipient, an operation, and an ACL. But what makes the "access granted" or "access denied" decision given these three pieces of information? The answer to that question is an `IAclVoter`. An instance of `IAclVoter` contains an all-important `hasAccess` method. It takes the three aforementioned inputs and returns a boolean result: `true` meaning access granted and `false` meaning access denied. An ACL voter is a singleton; there is only one instance per Java virtual machine. It is specified in `pentaho.xml`.

One might ask: How many ways can a voter arrive at a decision? Assume that user `sally` has the following granted authorities: `ROLE_DEV` and `ROLE_MGR`. Also assume that the ACL for a particular object contains the following entries: (`sally`, `read`), (`ROLE_DEV`, `readwrite`). Both ACL entries are applicable to `sally` since the first specifies `sally` (and she is `sally`) and the second specifies `ROLE_DEV` (and she has been granted the `ROLE_DEV` authority). Should the voter grant or deny a request to write to the object associated with this ACL? This is where extensibility of the voting system comes in. The Pentaho BI Platform provides multiple implementations of `IAclVoter` that each make different decisions in this situation! As the user of the platform, you decide how access decisions are made through your choice of `IAclVoter`. For more information about `IAclVoter` implementations, see 12. IAclVoter Node.

ACL Management

ACLs can be managed using a graphical interface that is accessed via the Admin menu. Once on the Admin menu, click Permissions to start the manager.

Permissions Editor



Note that in order to access the Permissions Editor, you must be logged in as an administrator to the platform. Also, ACLs are only available if you are using the RDBMS solution repository. This feature is not available for the file-based solution repository implementation.

In the screenshot above, the tree on the left represents all of the solution repository objects in your solution repository. You can set permissions on any level in the solution repository object tree. Setting permissions on lower level objects in the tree overrides permission settings higher in the tree. Conversely, if you set a permission on a solution repository object that has children, and the children do not have specific permissions set, they inherit the permissions settings from their parent. So, for example, if I set execute permissions for JoeUser on the analysis object, then the query1.xaction object inherits that execute permission. However, if I then set write and execute permission on the query1.xaction for JoeUser, these permissions are honored for that object, but other children of the analysis object would still only have their parent's (analysis) execute permission.

Today, there are only two permissions available, write and execute. Note that if you set write permission for an object, they automatically will get execute permission as well. Write permission allows the assigned user or role to edit the solution repository object, currently enforced through the platform client tools (Report Design Wizard, Report Designer and Cube Designer). Execute permission allows the user or role to execute the solution repository object, applicable to action sequences.

Each solution repository object can have any number of permission-role or permission-user combinations set. The
middle panel in the screenshot above lists the access control list entries defined for the solution repository object selected in the tree. You can modify the permissions for the roles or users that are defined in the existing access control list entries:

1. Check or uncheck the box for the permission you wish to remove or grant, next to the role or user that you wish this change to be applicable to.

2. Click the Update... button to submit the change.
3. Clicking the Reset button will reverse any changes that have NOT YET been submitted.

To add a new access control list entry, follow these steps:

1. Click the Add button under the access control list entry table. You will see a new list appear on the right, that lists all roles and users available to the system.
2. Select the roles and/or users that you wish to grant permissions to, and then select the permissions that you would like them to recieve.
3. Click the Add button at the bottom of the New Permission panel to add your newly defined access control list entries.

Handy Hint

If your organization has many users and/or you wish to only create ACL entries using roles, you can increase performance by adjusting the settings containing in the `access-ui node` in `pentaho.xml`.

ACL Publishing

The db-based solution repository is refreshed from the filesystem. In other words, solution repository objects are created as files on the filesystem and those objects are refreshed (published) in the db-based solution repository. In the filesystem, solution repository objects have no associated ACLs--at least as far as the platform is concerned. But once solution repository objects are published to the db-based repository, they do have associated ACLs. So how did the objects get their ACLs? The answer is an `IAclPublisher`. There is only one `IAclPublisher` instance per JVM and the type of that instance is specified in `pentaho.xml`. For more information about `IAclPublisher` implementations, see 13. IAclPublisher Node.

Note that an `IAclPublisher` is only responsible for the initial publishing of ACLs. After the filesystem is initially published, the Admin > Permissions interface should be used to tweak permissions.

04. Security HOWTOs

This page last changed on Mar 16, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

Changing the Admin Role

This page last changed on Jun 05, 2007 by mlowery.

By default, the platform defines an administrative role called `Admin`. Use the steps below to change this value. For the examples below, assume that the new administrative role is called "`NewAdmin`."

pentaho.xml

In `pentaho.xml`, update the `admin-role` element within the `acl-voter` element.

```
<pentaho-system>
  <acl-voter>
    <admin-role>NewAdmin</admin-role>
    ...
```

Additionally, replace any references to the old administrative role within the `default-acls` element within the `acl-publisher` element.

```
<pentaho-system>
  <acl-publisher>
    <default-acls>
      <acl-entry role="NewAdmin" acl="ADMIN_ALL" />
      ...
```

Warning

If you modify the `acl-publisher` element, you'll probably need to re-apply the default ACLs. Please see Re-Applying Default ACL. Be careful though as re-applying default ACLs will reset any ACLs created through the Admin Permissions interface.

applicationContext-acegi-security.xml

Using the Acegi Security documentation (section 21.3) as your guide, modify the `objectDefinitionSource` property of the `filterInvocationInterceptor` bean to match the new admin group.

Warning

While the example below only shows a single url to role mapping, multiple lines in `objectDefinitionSource` refer to the administrative role and therefore must be changed too.

```
<property name="objectDefinitionSource">
  <value>
    <![CDATA[
    CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
    PATTERN_TYPE_APACHE_ANT
    ...
    /admin*=NewAdmin
```

```
      ...
    ]]>
    </value>
  </property>
```

Changing to the JDBC Security DAO

This page last changed on Apr 25, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

By default, the Pentaho distribution comes with the "in-memory" security data access object (DAO) enabled. Because this is only recommended for testing, you'll want to switch over to either a relational database back-end or an LDAP back-end. This page shows you how to switch to the "JDBC" DAO. The instructions below describe a sample security database using HSQLDB.

1. Edit web.xml
   Change the Spring XML files to use the JDBC DAOs instead of the in-memory ones. Open `pentaho.war/WEB-INF/web.xml` and look for the following section:

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/applicationContext-acegi-security.xml
    /WEB-INF/applicationContext-common-authorization.xml
    /WEB-INF/applicationContext-acegi-security-memory.xml
    /WEB-INF/applicationContext-pentaho-security-memory.xml
  </param-value>
</context-param>
```

   Change that section to look like:

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/applicationContext-acegi-security.xml
    /WEB-INF/applicationContext-common-authorization.xml
    /WEB-INF/applicationContext-acegi-security-jdbc.xml
    /WEB-INF/applicationContext-pentaho-security-jdbc.xml
  </param-value>
</context-param>
```

2. Start the database
   The security database will need to be running before the first user logs in.

Command to start HSQLDB

```
java -cp lib\hsqldb.jar org.hsqldb.Server -database.0 userdb -dbname.0 userdb -port 9002
exit
```

3. Create the security tables
   The sample Spring XML files (i.e. `pentaho.war/WEB-INF/applicationContext-acegi-security-jdbc.xml` and

pentaho.war/WEB-INF/applicationContext-pentaho-security-jdbc.xml) assume the tables below. If you already have security tables setup, or you wish to alter the sample, you'll need to adjust your SQL queries in the aforementioned Spring XML files.

Sample SQL for HSQLDB to create security tables

```
CREATE SCHEMA PUBLIC AUTHORIZATION DBA
CREATE MEMORY TABLE USERS(USERNAME VARCHAR(50) NOT NULL PRIMARY KEY,PASSWORD
VARCHAR(50) NOT NULL,ENABLED BOOLEAN NOT NULL)
CREATE MEMORY TABLE AUTHORITIES(AUTHORITY VARCHAR(50) NOT NULL PRIMARY
KEY,DESCRIPTION VARCHAR(100))
CREATE MEMORY TABLE GRANTED_AUTHORITIES(USERNAME VARCHAR(50) NOT NULL,AUTHORITY
VARCHAR(50) NOT NULL,CONSTRAINT FK_GRANTED_AUTHORITIES_USERS FOREIGN
KEY(USERNAME) REFERENCES USERS(USERNAME),CONSTRAINT
FK_GRANTED_AUTHORITIES_AUTHORITIES FOREIGN KEY(AUTHORITY) REFERENCES
AUTHORITIES(AUTHORITY))
CREATE USER SA PASSWORD ""
GRANT DBA TO SA
SET WRITE_DELAY 10
SET SCHEMA PUBLIC
INSERT INTO USERS VALUES('admin','secret',TRUE)
INSERT INTO USERS VALUES('joe','password',TRUE)
INSERT INTO USERS VALUES('pat','password',TRUE)
INSERT INTO USERS VALUES('suzy','password',TRUE)
INSERT INTO USERS VALUES('tiffany','password',TRUE)
INSERT INTO AUTHORITIES VALUES('Admin','Super User')
INSERT INTO AUTHORITIES VALUES('ROLE_ANONYMOUS','User has not logged in')
INSERT INTO AUTHORITIES VALUES('Authenticated','User has logged in')
INSERT INTO AUTHORITIES VALUES('ceo','Chief Executive Officer')
INSERT INTO AUTHORITIES VALUES('cto','Chief Technology Officer')
INSERT INTO AUTHORITIES VALUES('dev','Developer')
INSERT INTO AUTHORITIES VALUES('devmgr','Development Manager')
INSERT INTO AUTHORITIES VALUES('is','Information Services')
INSERT INTO GRANTED_AUTHORITIES VALUES('joe','Admin')
INSERT INTO GRANTED_AUTHORITIES VALUES('joe','ceo')
INSERT INTO GRANTED_AUTHORITIES VALUES('joe','Authenticated')
INSERT INTO GRANTED_AUTHORITIES VALUES('suzy','cto')
INSERT INTO GRANTED_AUTHORITIES VALUES('suzy','is')
INSERT INTO GRANTED_AUTHORITIES VALUES('suzy','Authenticated')
INSERT INTO GRANTED_AUTHORITIES VALUES('pat','dev')
INSERT INTO GRANTED_AUTHORITIES VALUES('pat','Authenticated')
INSERT INTO GRANTED_AUTHORITIES VALUES('tiffany','dev')
INSERT INTO GRANTED_AUTHORITIES VALUES('tiffany','devmgr')
INSERT INTO GRANTED_AUTHORITIES VALUES('tiffany','Authenticated')
INSERT INTO GRANTED_AUTHORITIES VALUES('admin','Admin')
INSERT INTO GRANTED_AUTHORITIES VALUES('admin','Authenticated')
```

4. Start the application server
   Now that the database is running and the security tables have been created, start the application server.
5. Stop the database
   When you shutdown your application server, you'll want to shutdown the security database as well. The command to do that is below.

Command to stop HSQLDB

```
java -cp lib\hsqldb.jar org.hsqldb.util.ShutdownServer -url
"jdbc:hsqldb:hsql://localhost:9002/userdb" -user "sa" -password ""
exit
```

Changing to the LDAP Security DAO

This page last changed on Apr 25, 2007 by mlowery.

> **Useful Information**
>
> Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

By default, the Pentaho distribution comes with the "in-memory" security data access object (DAO) enabled. Because this is only recommended for testing, you'll want to switch over to either a relational database back-end or an LDAP back-end. This page shows you how to switch to the "LDAP" DAO. The instructions below describe a sample LDAP directory using ApacheDS and JXplorer.

1. Edit web.xml
   Change the Spring XML files to use the JDBC DAOs instead of the in-memory ones. Open `pentaho.war/WEB-INF/web.xml` and look for the following section:

```
<context-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>
   /WEB-INF/applicationContext-acegi-security.xml
   /WEB-INF/applicationContext-common-authorization.xml
   /WEB-INF/applicationContext-acegi-security-memory.xml
   /WEB-INF/applicationContext-pentaho-security-memory.xml
 </param-value>
</context-param>
```

   Change that section to look like:

```
<context-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>
   /WEB-INF/applicationContext-acegi-security.xml
   /WEB-INF/applicationContext-common-authorization.xml
   /WEB-INF/applicationContext-acegi-security-ldap.xml
   /WEB-INF/applicationContext-pentaho-security-ldap.xml
 </param-value>
</context-param>
```

2. Start the directory
   In Windows, just start the `Apacheds` service in the Services dialog.
3. Import the LDIF
   The sample Spring XML files (i.e. `pentaho.war/WEB-INF/applicationContext-acegi-security-ldap.xml` and `pentaho.war/WEB-INF/applicationContext-pentaho-security-ldap.xml`) assume the records below. If you already have a directory setup, or you wish to alter the sample, you'll need to adjust your LDAP queries in the aforementioned Spring XML files. Note: You may or may not have success importing the LDIF files below. If you run into errors, manually create the records. The first LDIF file below contains content records. Content records add records. The second LDIF file below contains change records. Change records alter existing records.

Sample LDIF (content records)

```
version: 1

dn: uid=joe,ou=users,ou=system
mail: joe.pentaho@pentaho.org
uniquemember: cn=ceo,ou=roles
uniquemember: cn=admin,ou=roles
objectclass: inetOrgPerson
objectclass: organizationalPerson
objectclass: person
objectclass: groupOfUniqueNames
objectclass: top
uid: joe
cn: joe
businesscategory: cn=ceo,ou=roles,ou=system
businesscategory: cn=admin,ou=roles,ou=system
businesscategory: cn=admin,ou=roles,ou=system
userpassword:: cGFzc3dvcmQ=
sn: Pentaho

dn: uid=suzy,ou=users,ou=system
mail: suzy.pentaho@pentaho.org
uniquemember: cn=cto, ou=roles
uniquemember: cn = is , ou = roles
objectclass: inetOrgPerson
objectclass: organizationalPerson
objectclass: person
objectclass: groupOfUniqueNames
objectclass: top
uid: suzy
cn: suzy
userpassword:: cGFzc3dvcmQ=
businesscategory: cn=cto,ou=roles,ou=system
businesscategory: cn=is,ou=roles,ou=system
sn: Pentaho

dn: uid=tiffany,ou=users,ou=system
mail: tiffany.pentaho@pentaho.org
uniquemember: cn=devmgr,ou=roles
uniquemember: cn=dev,ou=roles
objectclass: inetOrgPerson
objectclass: organizationalPerson
objectclass: person
objectclass: groupOfUniqueNames
objectclass: top
uid: tiffany
cn: tiffany
businesscategory: cn=devmgr,ou=roles,ou=system
businesscategory: cn=dev,ou=roles,ou=system
userpassword:: cGFzc3dvcmQ=
sn: Pentaho
```

```
dn: uid=pat,ou=users,ou=system
mail: pat.pentaho@pentaho.org
uniquemember: cn=dev,ou=roles
objectclass: inetOrgPerson
objectclass: organizationalPerson
objectclass: person
objectclass: groupOfUniqueNames
objectclass: top
uid: pat
cn: pat
businesscategory: cn=dev,ou=roles,ou=system
userpassword:: cGFzc3dvcmQ=
sn: Pentaho

dn: cn=Sales,ou=groups,ou=system
cn: Sales
objectclass: groupofuniquenames
objectclass: top
uniquemember: uid=joe,ou=users,ou=system

dn: cn=Marketing,ou=groups,ou=system
cn: Marketing
objectclass: groupofuniquenames
objectclass: top
uniquemember: uid=suzy,ou=users,ou=system

dn: cn=Development,ou=groups,ou=system
cn: Development
objectclass: groupOfUniqueNames
objectclass: top
uniquemember: uid=pat,ou=users,ou=system
uniquemember: uid=tiffany,ou=users,ou=system

dn: ou=roles,ou=system
ou: roles
objectclass: organizationalUnit
objectclass: top

dn: cn=devmgr,ou=roles,ou=system
l: Orlando
objectclass: organizationalRole
objectclass: top
description: Development Manager
roleoccupant: uid=tiffany,ou=users,ou=system
cn: devmgr

dn: cn=cto,ou=roles,ou=system
cn: cto
description: CTO Role
objectclass: organizationalRole
```

```
objectclass: top
roleoccupant: uid=suzy,ou=users,ou=system
l: Orlando
st: Florida

dn: cn=ceo,ou=roles,ou=system
cn: ceo
description: CEO Role
objectclass: organizationalRole
objectclass: top
roleoccupant: uid=joe,ou=users,ou=system
l: Orlando
st: Florida

dn: cn=is,ou=roles,ou=system
cn: is
description: Information Systems
objectclass: organizationalRole
objectclass: top
roleoccupant: uid=suzy,ou=users,ou=system
l: Orlando
st: Florida

dn: cn=dev,ou=roles,ou=system
objectclass: organizationalRole
objectclass: top
roleoccupant: uid=pat,ou=users,ou=system
roleoccupant: uid=tiffany,ou=users,ou=system
cn: dev

dn: cn=admin,ou=roles,ou=system
objectclass: organizationalRole
objectclass: top
roleoccupant: uid=joe,ou=users,ou=system
roleoccupant: uid=admin,ou=system
cn: admin

dn: cn=authenticated,ou=roles,ou=system
objectclass: organizationalRole
objectclass: top
roleoccupant: uid=joe,ou=users,ou=system
roleoccupant: uid=suzy,ou=users,ou=system
roleoccupant: uid=suzy,ou=users,ou=system
roleoccupant: uid=pat,ou=users,ou=system
roleoccupant: uid=pat,ou=users,ou=system
roleoccupant: uid=tiffany,ou=users,ou=system
roleoccupant: uid=tiffany,ou=users,ou=system
cn: authenticated

dn: cn=anonymous,ou=roles,ou=system
cn: anonymous
```

```
objectclass: organizationalRole
objectclass: top
```

Sample LDIF (change records)

```
version: 1
dn: cn=administrators,ou=groups,ou=system
changetype: modify
add: uniqueMember
uniqueMember: uid=joe,ou=users,ou=system
```

4. Start the application server
   Now that the directory is running and the LDIF has been imported, start the application server.
5. Stop the directory
   In Windows, just stop the `Apacheds` in the Services dialog.

Customizing ACL Decisions

This page last changed on Mar 16, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

Creating an IAclVoter

IAclVoter Hierarchy

Before proceeding, be sure that you have read about domain object authorization in the platform. The section on domain object authorization introduces access control lists (ACLs) and the `IAclVoter` interface. The Pentaho BI Platform provides an abstract class called `AbstractPentahoAclVoter` that implements `IAclVoter` as well as a concrete subclass called `PentahoBasicAclVoter`. If you want to provide your own implementation of the `IAclVoter` interface, it is recommended that you consider starting with the `PentahoBasicAclVoter`. Use this class as your superclass, and override behaviors as desired.

Voter Example

Assume that you want to build your own ACL voter. The requirements are as follows:

- Allow content to be accessed by anonymous users.
- However, if a user is specified in the access control list for an object, those access controls should override anything else in the access control list.

This is essentially merging the functionality of the `PentahoUserOverridesVoter` and `PentahoAllowAnonymousAclVoter`, both classes provided by the platform. So, where should one begin?

Consider the following implementation options:

1. Subclass `AbstractPentahoAclVoter`. This would require the most work because one would have to duplicate the logic in `PentahoBasicAclVoter`, `PentahoUserOverridesVoter` and `PentahoAllowAnonymousAclVoter`.
2. Subclass `PentahoBasicAclVoter`. This would be the second most difficult because this would require one to implement all the work currently being done by `PentahoUserOverridesVoter` and `PentahoAllowAnonymousAclVoter`.
3. Subclass either `PentahoAllowAnonymousAclVoter` or `PentahoUserOverridesVoter` and add the

functionality from the non-subclassed into the new one.

Option #3 seems like the best way to go. But should one subclass `PentahoUserOverridesVoter` or `PentahoAllowAnonymousAclVoter`? Tthe javadoc for `PentahoAllowAnonymousAclVoter` states that it simply overrides the `getAuthentication(IPentahoSession)` method to allow anonymous sessions. And, since getting the authentication from the `IPentahoSession` object is actually a `SecurityUtils` call, this is the easiest functionality to add to any voter. However, the logic in the `PentahoUserOverridesVoter` is a bit more involved, and not something that one would like to duplicate. Given this information, the decision is made to subclass `PentahoUserOverridesVoter` and simply add the `getAuthentication(IPentahoSession)` method that allows anonymous access. The final class ends up looking like:

```
package com.pentaho.security.acls.voter;

import org.acegisecurity.Authentication;
import org.pentaho.core.session.IPentahoSession;

import com.pentaho.security.SecurityUtils;

public class PentahoUserOverridesAllowAnonymousAclVoter
   extends PentahoUserOverridesVoter {
  // Allow anonymous users to have possible acls on an entry.
  public Authentication getAuthentication(IPentahoSession session) {
    return SecurityUtils.getAuthentication(session, true);
  }
}
```

This solution turns out to be the best of both worlds, and it meets the requirements without being too arduous to implement.

Your options are unlimited with respect to implementing your own ACL voters. For example, consider the case of a user (Sally) going on vacation, and delegating her responsibilities to someone else (Joe). Often this means a difficult change request going through the IT department that temporarily assigns Joe all of the roles that Sally is a member of. This would allow Joe access to all the solutions and action sequences that Sally has access to. Then, when Sally comes back, you'd have to process another change request through the IT department that would set Joe's access back the way it was before Sally went on vacation.

Or, you could create an ACL voter that, in addition to looking at Joe's roles, also looks in a relational database or an XML file for role overrides based on the date. In this way, instead of having to go through the IT department to make these changes, you could have time-based voter overrides.

Customizing the Login Page

This page last changed on Mar 16, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

In the shipping implementation of the Pentaho BI Platform there is an implementation of a login page. This page is defined in the form of a java server page located your applications server deployment directory in `pentaho.war/jsp/Login.jsp`. For instance, in the pre-configured install it would be located at `/pentaho-preconfiguredinstall/server/default/deploy/pentaho.war/jsp/Login.jsp`.

As currently shipped the login JSP is branded for Pentaho and contains a drop down menu with default users and passwords. These items (and others) can be changed by editing the `Login.jsp`.

Removing the "Valid Users" Drop Down

In `Login.jsp`, comment out or remove the following HTML and code. Note that formatting may be slightly different.

Warning

Note that the end-of-line backslashes that occur in the excerpt below are present for formatting purposes only and should not be present in the actual file.

```
<tr>
  <td>
    <%= ProMessages.getString("UI.USER_USERS_PROMPT") %>
  </td>
  <td>
    <select onchange="document.forms.login.elements.j_username.value= \
    this.options[this.selectedIndex].value; \
    document.forms.login.elements.j_password.value='password'">
      <option value="" selected><%= Messages.getString("UI.USER_SELECT") %></option>
      <option value="suzy">Suzy</option>
      <option value="joe">Joe (<%= Messages.getString("UI.USER_ADMIN") %>)</option>
      <option value="pat">Pat</option>
      <option value="tiffany">Tiffany</option>
    </select>
  </td>
</tr>
```

This removes the drop down menu and its title text in the login area.

Removing the Pentaho Branding

In `Login.jsp`, comment out or remove the following HTML and code. Note that formatting may be slightly different.

```
<td width="400" style="text-align:left">
  <span class="content_header" style="font-size:1.15em">
    <%= Messages.getString( "UI.USER_WHATS_NEW") %>
  </span>
  <p/>
  <%
    ArrayList messages = new ArrayList();
    HtmlComponent html = new HtmlComponent( HtmlComponent.TYPE_URL,
                                            "http://www.pentaho.org/demo/news1.htm",
                                            Messages.getString("UI.USER_OFFLINE"),
                                            null,
                                            messages);
  %>
  <%=  html.getContent( "text/html" ) %>
</td>
```

This will remove the Pentaho branding from the left side of the login page.

Localization

All strings in `Login.jsp` have been localized. Open `com.pentaho.locale.messages*.properties` to edit the strings. Strings used in the login page generally are named `UI.USER_LOGIN_*` or `UI.USER_*`.

Other Login Customizations

Implementors are free to change the `Login.jsp` as they see fit. The heart of the currently implemented `Login.jsp` is in the `<form>` tag. It is recommended that users examine this code in order to make their own replacement `Login.jsp` work correctly.

Implementing a New Security DAO

This page last changed on Apr 13, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

Implementing Your Own Security Data Access Object

In order to implement a new security data access object, one needs to implement `UserDetailsService`, an Acegi Security interface, and `UserRoleListService`, a Pentaho interface. To implement `UserDetailsService`, please consult the Acegi Security documentation. `UserDetailsService` implementations know how to get a user's record along with roles from some backend, given a username. The `UserRoleListService` interface is required for the Pentaho BI Platform administrative user interface for assigning access to action sequences. It's also a requirement for some of the security input parameters (e.g. listing all users).

Re-Applying Default ACL

This page last changed on Mar 16, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

Because ACLs are published the first time you start the platform and because the default ACL might not meet your needs, you might wish to re-apply a different default ACL to solution repository objects. To do this, follow the steps below but understand that these steps will remove any ACL management that has been done via the Admin > Permissions interface.

Warning

Note that steps below will remove any ACL management that has been done via the Permissions interface!

1. Stop platform (if it is running).
2. Start Hypersonic (or make sure your database server is running).
3. Execute SQL:

```
DROP TABLE PRO_ACLS_LIST;

DROP TABLE PRO_FILES;

DELETE FROM VERSIONMAP WHERE VERSIONKEY='pro-RepositoryFile';
```

   The dropped tables will be re-created when you start the platform.

4. Stop Hypersonic (not necessary for other database servers).
5. Edit `pentaho.xml` to specify new default ACL. (See 13. IAclPublisher Node for information on how to do this.)
6. Start Hypersonic (or make sure your database server is running).
7. Start platform.

Refreshing JBoss Portal Database

This page last changed on Mar 23, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

If your JBoss Portal database becomes corrupted, there's an easy fix. Follow the steps below.

1. Shutdown the PCI.
2. Delete the `server/default/data/portal` directory.
3. Start the PCI.
4. Shutdown the PCI. (By now the JBoss Portal should have re-created all of its tables.)
5. Startup the portal database so that you can add some users and roles. This command should be executed in `server/default/data/portal/hypersonic`.

Starting Hypersonic

```
java -cp "<path-to-hsqldb.jar>" org.hsqldb.Server -database.0 database -dbname.0 database -port 9002
```

6. Execute the following SQL

SQL for user and role creation

```
--add pentaho users
INSERT INTO JBP_USERS
VALUES(3,'suzy',NULL,NULL,'5f4dcc3b5aa765d61d8327deb882cf99','suzy.pentaho@pentaho.org','','2007-02-12 23:45:27.828000000',FALSE,TRUE);
INSERT INTO JBP_USERS
VALUES(4,'joe',NULL,NULL,'5f4dcc3b5aa765d61d8327deb882cf99','joe.pentaho@pentaho.org','','2007-02-12 23:45:42.828000000',FALSE,TRUE);


--assign roles to users
INSERT INTO JBP_ROLE_MEMBERSHIP VALUES(3,2); --suzy gets User role
INSERT INTO JBP_ROLE_MEMBERSHIP VALUES(4,1); --joe gets Admin role
INSERT INTO JBP_ROLE_MEMBERSHIP VALUES(4,2); --joe gets User role


--update sequences
ALTER TABLE JBP_USERS ALTER COLUMN JBP_UID RESTART WITH 5;
```

7. Shutdown the portal database. This command should be executed in `server/default/data/portal/hypersonic`.

Shutting down Hypersonic

```
java -cp "<path-to-hsqldb.jar>" org.hsqldb.util.ShutdownServer -url
"jdbc:hsqldb:hsql://localhost:9002/database" -user "sa" -password ""
```

Removing Security

This page last changed on Mar 16, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

The security features of the Pentaho BI Platform cannot be removed. However, they can be effectively removed by using the following steps. Essentially, the idea is to create a single user and role and give system-wide access to that user.

1. Allow anonymous access to all web resources by editing the `objectDefinitionSource` on the `FilterSecurityInterceptor` to look like the example below.
2. Use `PentahoAllowAnonymousAclVoter` as your `IAclVoter` implementation. See 03. pentaho.xml for a description of how to configure this voter. When configuring this voter, you will define the anonymous user and role. That user and/or role should be used when assigning ACLs.
3. Assign ACLs using the user and role defined in the previous step.

```
<bean id="filterInvocationInterceptor"
  class="org.acegisecurity.intercept.web.FilterSecurityInterceptor">
  <property name="authenticationManager">
    <ref local="authenticationManager" />
  </property>
  <property name="accessDecisionManager">
    <ref local="httpRequestAccessDecisionManager" />
  </property>
  <property name="objectDefinitionSource">
    <value>
      <![CDATA[
        CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
        PATTERN_TYPE_APACHE_ANT
        /**=ROLE_ANONYMOUS
      ]]>
    </value>
  </property>
</bean>
```

Using Security from Action Sequences

This page last changed on Mar 16, 2007 by mlowery.

Useful Information

Starting in version 1.5, security is a feature of the Pentaho BI Platform. Prior to this version, security was only available in the Pentaho *Professional* BI Platform. Furthermore, this document is relevant only to the Pentaho Professional BI Platform version 1.2.1 or later or the Pentaho BI Platform version 1.5 or later. See the Pentaho Professional BI Platform version 1.2.0 security documentation if you're using Pentaho Professional BI Platform version 1.2.0. (You can find the version you are running in several ways: (1) look at the log when the Pentaho BI Platform starts or (2) look at the bottom right of any page within the Pentaho BI Platform.)

In addition to providing access to security information within the web application code, the security system of the platform provides access to security information within action sequences. The information then can be used in JavaScript rules, presented in reporting prompts, provided as input to SQL Lookup Rules, etc.

If you look at the inputs section of an action sequence, you will see inputs typically defined like this:

```
<inputs>
  <someInput type="string">
    <sources>
      <request>someInput</request>
    </sources>
  </someInput>
</inputs>
```

In the above example, the input (called `someInput`) can be found by looking at the request (`HttpServletRequest`, `PortletRequest`, etc.) for a variable called `someInput`. Then, throughout the rest of the action sequence, specific actions can reference that input.

Security Inputs

Pentaho BI Platform extends the inputs to provide a unique type of input--the security input. This input (as of this writing) supports the following input names:

Input Name	Type	Description
`PrincipalName`	`string`	The name of the currently authenticated user.
`PrincipalRoles`	`string-list`	The roles that the currently authenticated user is a member of.
`PrincipalAuthenticated`	`string`	`true` if the user is authenticated, `false` otherwise.
`PrincipalAdministrator`	`string`	`true` if the user is considered a Pentaho Administrator, `false` otherwise.

systemRoleNames	string-list	All the known roles in the system. Use caution since this list could be quite long.
systemUserNames	string-list	All the users known to the system. Use caution since this list could be quite long.

Example

The following input section will get the list of the user's roles, and make it available to all the actions in the action sequence:

```
<inputs>
  <principalRoles type="string-list">
    <sources>
      <security>principalRoles</security>
    </sources>
  </principalRoles>
</inputs>
```