NCICB

Ekagra

High Impact – High Value – Business Results

Common Security Module

CSM Guide for Application Developers

Version No: 2.0

Last Modified: 12/20/2006

Author : Vinay Kumar, Eric Copen, Kalpesh Patel, Kunal Modi

Team : Common Security Module (CSM)

Purchase Order# 3455

Client : National Cancer Institute - Center for Bioinformatics,
National Institutes of Health,
US Department of Health and Human Services

Document History

Document Location

The most current version of this document is located on the CSM website: http://ncicb.nci.nih.gov/core/CSM

Revision History

Version Number	Revision Date	Author	Summary of Changes
0.1	2/24/05	Vinay Kumar, Eric Copen, Kalpesh Patel	Initial Table of Contents
0.2	3/02/05	Eric Copen	Integrating separate existing documents into this document, adding Introduction and CSM Overview Text, and text in other places as needed.
0.3	3/04/05	Kunal Modi	Incorporated Comments and Document Restructuring
1.0	3/04/05	Eric Copen	Prepared for release
1.1	3/21/05	Eric Copen	Incorporated changes from technical writers
1.2	5/31/05	Eric Copen	Changes for 3.0.1 release
1.3	7/08/05	Eric Copen	Changes in response to Jill's comments
1.4	08/15/05	Eric Copen	Added Workflow section
1.5	01/23/06	Steve Hunter, Art	Added CLM section
2.0	12/20/06	Kunal Modi	Added 3.2 Changes

Review

Name	Team/Role	Version	Date Reviewed	Reviewer Comments
Vinay Kumar	Team Lead	0.2	2/14/05	Approved
JJ Maurer	Ekagra Management	0.2	2/15/05	Approved with minor changes
Jill Hadfield, Liz Lucchesi	Technical Writers	1.0	3/04/05 – 3/18/05	Made minor changes for caCORE Technical Guide
Jill Hadfield	Technical Writer	1.2	7/08/05	Approved with minor changes
Jill Hadfield	Technical Writer	1.4	8/15/05	Provided feedback to Workflow
Wendy E.	Technical Writer	2.0	12/20/06	Provided feedback & Baselined

Related Documents

More information can be found in the following related CSM documents:

Document Name
UPT User Guide
Software Architecture Document
CSM Enterprise Architect Model
CSM Reference Implementation Guide

These and other documents can be found on the CSM website: http://ncicb.nci.nih.gov/core/CSM

Table of Contents

CSM Guide for Application Developers

1. Introduction to CSM

1.1 Purpose

This document provides all the information application developers need to successfully integrate with NCICB's Common Security Module (CSM). The CSM was chartered to provide a comprehensive solution to common security objectives so not all development teams need to create their own security methodology. CSM is flexible enough to allow application developers to integrate security with minimal coding effort. This phase of the Common Security Module brings the NCICB team one step closer to the goal of application security management, single sign-on, and Health Insurance Portability and Accountability Act (HIPPA) compliance.

1.2 Scope

This document shows how to deploy and integrate the CSM services, including Authentication, Authorization, and User Provisioning. For specific questions regarding using the UPT, refer to the User Provisioning Tool (UPT) User Guide (http://ncicb.nci.nih.gov/core/CSM).

1.3 Using This Guide

Begin by reading the CSM Overview section to learn the CSM concepts and how they apply to your own application. Read Workflow for CSM Integration to understand how to successfully integrate with CSM. Next, The Three Services on page 10 explains the three manager interfaces and the methods to incorporate them. The Deployment Models section on page 12 describes how to deploy the services and how to integrate with them. The deployment and integration sections (Integrating with the CSM Authentication Service on page 34, Integrating with the CSM Authorization Service on page 36, and Integrating with the User Provisioning Service on page 37) consist of multiple step-by-step guides to help you with a variety of configurations.

NOTE:

- This guide has been updated to include all enhancements available as part of the CSM v3.2 release. There is an explicit mention about these new v3.2 changes through out the document.
- Also note that when you are using the CSM v3.2 configuration settings then you can skip many of the earlier configuration setting required as mentioned in the sections.
- In order to use CSM v3.2 with an existing CSM v3.1 database, you need to follow the steps that are mentioned in a separate migration guide

2. CSM Overview

2.1 Explanation

The CSM provides application developers with powerful security tools in a flexible delivery. CSM provides solutions for:

1) **Authentication** - validating and verifying a user's credentials to allow access to an application. CSM, working with credential providers (Lightweight Directory Access Protocol (LDAP), Relational Database Management Systems (RDBMS), etc.), confirms that a user exists and that the password is valid for that application. It also provides a lockout manager which locks out unauthorized users for a pre-configured amount of time after the (also pre-configured) number of allowed attempts is reached.
2) **Authorization** - granting access to data, methods, and objects. CSM incorporates an Authorization schema and database so that users can only perform the operations or access the data to which they have access rights.
3) **User Provisioning** - creating or modifying users and their associated access rights to your application and its data. CSM provides a web-based UPT that can easily be integrated with a single or multiple applications and authorization databases. The UPT provides functionality to create authorization data elements like Roles, Protection Elements, Users, etc., and also provides functionality to associate them with each other. The runtime API can then use this authorization data to authorize user actions. The UPT consists of two modes – Super Admin and Admin.
   a. **Super Admin** – accessed by the UPT's overall administrator; used to register an application, assign administrators, and create or modify standard privileges.
   b. **Admin** – used by application administrators to modify authorization data, such as roles, users, protection elements, etc
4) **Audit Logging -** In an effort to make CSM compliant with CRF 21/ part 11, CSM will provide auditing and logging functionality. CSM uses another caCORE product called Common Logging Module (CLM) for the purpose of event logging as well as automated object state change logging into a persistant database.

Figure 2-1 shows how CSM works with an application and independent entities, such as the credential providers and authorization schema, to perform authentication and authorization.

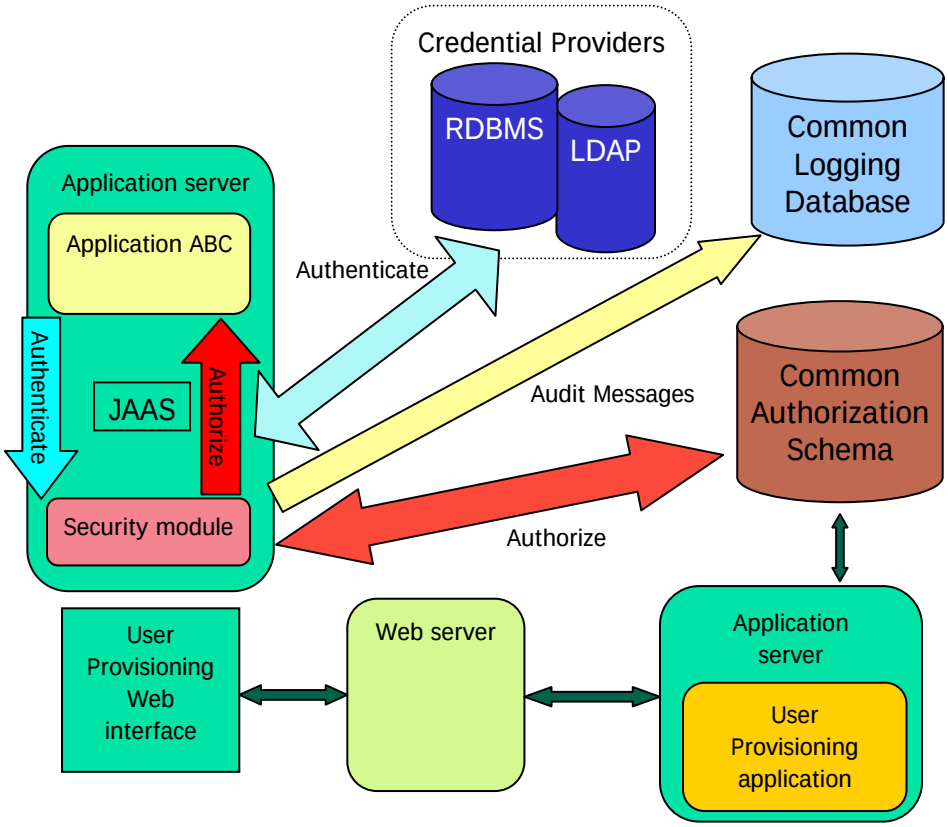

Figure 2-1 CSM interactions for authentication and authorization (see text)

CSM works with Java Authentication and Authorization Service (JAAS) to authenticate and authorize for the Application ABC. To authenticate, it references credential providers such as an LDAP or RDBMS. CSM can be configured to check multiple credential providers in a defined order. To authorize, CSM refers to the Authorization Schema. The Authorization Schema contains the Users, Roles, Protection Elements, etc., and their associations, so that the application knows whether or not to allow a user to access a particular object. The Authorization data can be stored on a variety of databases. It is created and modified by the Application Administrator using the web-based UPT.

CSM now uses CLM to perform all the Audit and Logging. CSM logs all of the events and object state changes (security objects stated below in Table 2-1). These logs will be stored in a separate Common Logging Database for backup and review. Since logging can be configured using log4j, client applications have control over the logging of audit trails.

2.2 Security Concepts

In order to successfully integrate CSM with an application, it is important to understand the definitions for the security concepts defined in Table 2-1. Application Developers should understand these concepts and begin to understand how they apply to their particular application.

Security Concept	Definition
Application	Any software or set of software intended to achieve business or technical goals.
User	A User is someone that requires access to an application. Users can become part of a Group, and can have an associated Protection Group and Roles.
Group	A Group is a collection of application users. By combining users into a Group, it becomes easier to manage their collective roles and access rights in your application.
Protection Element	A Protection Element is any entity (typically data) that has controlled access. Examples include Social Security Number, City, and Salary. Protection Elements can also include operations, buttons, links, etc.
Protection Group	A Protection Group is a collection of application Protection Elements. By combining Protection Elements into a Protection Group, it becomes easier to associate Users and Groups with rights to a particular data set. Examples include Address and Personal Information.
Privilege	A Privilege refers to any operation performed upon data. CSM makes use of a standard set of privileges. This will help standardize authorization to comply with JAAS and Authorization Policy and allow for adoption of technology such as SAML in the future.
Role	A Role is a collection of application Privileges. Examples include Record Admin. and HR Manager.

Table 2-1 Security concept definitions

CSM users need to identify aspects of the application that should be labeled as Protection Elements. These elements are combined to Protection Groups, and then users are assigned Roles for that Protection Group.

Shown in Table 2-2 are definitions of related security terms.

Related Concept	Definition
Credential Provider	A credential is a data or set of data which represents an individual unique to a given application (username, password, etc.). Credential providers are trusted organizations that create secure directories or databases that store credentials. In an authentication transaction, organizations check with the credential providers to verify entered information is valid. For example, the NCI network uses a credential provider to verify that a user name and password match and are valid before allowing access.
LDAP	Credential providers may choose to store credential information using a directory based on LDAP. An LDAP is simply a set of protocols for accessing information directories. Using LDAP, client programs can login to a server, access a directory, and verify credential entries.
RDBMS	Credential providers may choose to store credential information with a RDBMS. Unlike with LDAP, credential data is stored in the form of related tables.

Table 2-2 Related security concept definitions

2.3 Workflow for CSM Integration

This workflow section outlines the basic steps, both strategic and technical, for successful CSM integration.

1) Decide which services you would like to integrate with an application. If the application should authenticate users against an LDAP or other directory, select Authentication. If granular data protection is important, also integrate with the authorization and provisioning services. These options allow administrators to specify which users have access to particular components of the application.

2) Read the *CSM Guide for Application Developers* (this document). It provides an overview, workflow, and specific deployment and integration steps. If using the provisioning service, also read the *UPT User Guide*.

3) Appoint a Security Schema Administrator who is familiar with the application and its user base. Using the User Provisioning Tool (UPT), these individuals inputs user, roles, etc., and ultimately give privileges to users for certain application elements.

4) Determine a security authorization strategy. In this step, the Schema Administrator and the application team determines what data or links should be protected and what groups of people should have access to what.

5) Decide upon a deployment approach. As discussed in Section 4.3.3, authorization data can be stored on separate servers or as part of a common authorization

schema. Similarly, the UPT can be hosted locally or commonly. Your decision may be made based on speed, security, user commonality, or other factors.

6) Deploy Authentication, Authorization, and User Provisioning. These steps are listed in detail in this document.

7) Decide if you want to enable Audit Logging for these services or not. If yes then configure Audit Logging as explained later in the document

8) Input the authorization data using the UPT.

9) Integrate the application code using the integration steps for Authentication, Authorization, and User Provisioning.

10) Test and refine CSM integration with your application. Confirm that your authorization policy and implementation meets requirements.

3. The Three Services

The Security APIs consist of three primary components - Authentication, Authorization and User Provisioning. The following three corresponding managers control these components:

- AuthenticationManager
- AuthorizationManager
- UserProvisioningManager

3.1 AuthenticationManager

The AuthenticationManager is an interface that authenticates a user against a credential provider. See *Integrating with the CSM Authorization Service* on page 45 to learn how to integrate with the AuthenticationManager. Developers will work primarily with the login method. Detailed descriptions about each method's functionality and its parameters are present in the CSM API Javadocs.

3.2 AuthorizationManager

The AuthorizationManager is an interface which provides run-time methods with the purpose of checking access permissions and provisioning certain authorization data. See *Integrating with the CSM Authorization Service* on page 45 to learn how to integrate with the AuthorizationManager. Detailed descriptions about each method's functionality and its parameters are present in the CSM API Javadocs.

3.3 UserProvisioningManager

The UserProvisioningManager is the interface used by the UPT. This manager provides an interface where application developers can provision user access rights. Since the UserProvisioningManager is primarily used internally by the UPT Tool, there is no integration shown in this document. Detailed descriptions about each method's functionality and its parameters are present in the CSM API Javadocs.

4. Deployment Models

4.1 Authentication

4.1.1 Introduction

The CSM Authentication Service provides a simple and comprehensive solution for user authentication. Developers can easily incorporate the service into their applications with simple configuration and coding changes. This service allows authentication using LDAP and RMDBS credential providers.

4.1.2 Purpose

This section serves as a guide to help caCORE developers integrate existing applications with the CSM application. This section outlines a step by step process that addresses what developers need to know in order to successfully integrate, including:

- Jar placement
- Configuring the ApplicationSecurityConfig.xml (only needed for releases prior to v3.2)
- Database properties and configuration
- LDAP properties and configuration

4.1.3 Scope

The CSM Authentication Service is available for all caCORE applications. Although it can be used exclusively and is effective on its own, it does not need to replace existing authentication. Rather, it can be used to supplement your application's current authentication mechanism. Currently, only RDBMS-based and LDAP-based authentication is supported by CSM.

4.1.4 Definitions, Acronyms, and Abbreviations

Shown in Table 4-1 are definitions important for understanding the rest of the section.

Term	*Definition*
ABC Application	In a few instances we refer to an ABC application (abcapp) which is simply a sample application. Use of this example helps to illustrate how to integrate an application in CSM. It has been integrated with the CSM code to perform the authentication using the ABC database.
Login Module	Responsible for authenticating users and for populating users and groups. A Login Module is a required component of an authentication provider, and can be a component of an identity assertion provider if you want to develop a separate LoginModule for perimeter authentication. LoginModules that are not used for perimeter authentication also verify the proof material submitted (for example, a user password).
JAAS	Set of Java packages that enable services to authenticate and enforce access controls upon users. JAAS implements a Java version of the

Term	Definition
	standard Pluggable Authentication Module framework, and supports user- based authorization.

Table 4- 1Definitions for important terms

4.1.5 JAR Placement

The CSM Application is available as a JAR which needs to be placed in the classpath of the application. Along with this JAR, there are many supporting JARs on which the CSM API depends. These should be added in the folder `<application-web-root>\WEB-INF\lib.`

4.1.6 Authentication Properties and Configuration

4.1.6.1 Requirements

If preferred, the client application `abcapp` can use its own AuthenticationManager instance instead of the default JAAS implementation. In order to configure its own implementation of the AuthenticationManager, the client application needs its own entry in the `ApplicationSecurityConfig.xml` file. If no entry is found for the given application context name in the `Authentication.Properties` file, then the default JAAS implementation is used for performing the authentication.

4.1.6.2 Configuring an Authentication Manager

With version 3.2, if you are planning to use the default AuthenticationManager Implementation provided by CSM then there is no need to configure the ApplicationSecurityConfig.xml file. You can skip this entire section. However, since v3.2 is backward compatible, users can still continue using the old method of configuration using the ApplicationSecurityConfig.xml file as needed.

Also developers can specify their own AuthenticationManager implementation class by making an entry in `ApplicationSecurityConfig.xml` against the application context name as shown in Figure 4-1. Note that the application name must match the application context name provided at the time of obtaining the instance of the AuthenticationManager using the SecurityServiceProvider. Also the class name provided should be fully qualified.

```
<application>
        <context-name>
                FooApplication
        </context-name>
        <authentication>
                <lockout-time>
                        30000
                </lockout-time>
                <allowed-login-time>
                        30000
                </allowed-login-time>
                <allowed-attempts>
                        2
                </allowed-attempts>
                <authentication-provider-class>
                        com.Foo.AuthenticationManagerClass
                </authentication-provider-class>
        </authentication>
                :
                :
</application>
```

Figure 4-1 Specifying AuthenticationManager implementation class & Lockout Configuration

1. The location of the ApplicationSecurityConfig.xml needs to be specified to the API. This is done using a system property. In JBoss, edit the JBoss `properties-service.xml` to provide a startup parameter to the JBoss server. This file is located at the following path: `{jboss-home}/server/standard/deploy/properties-service.xml` where `{jboss-home}` is the base directory where JBoss is installed on the server.
2. Add the following entry to the existing properties in the properties-service.xml file:

   ```
   <attribute name="Properties"> <!-- could already exist -->
   :
   gov.nih.nci.security.configFile=/foo/bar/ApplicationSecurityConfig.xml
   :
   </attribute> <!-- could already exist -->
   ```

   The `gov.nih.nci.security.configFile` is the name of the property which points to the fully qualified path `foo/bar/ApplicationSecurityConfig.xml` where the `ApplicationSecurityConfig.xml` has been created above. The name of the property has to be the `gov.nih.nci.security.configFile` and cannot be modified as it is a system-wide property.
3. Save this file in a deploy folder (for example, `{jboss-home}/server/default/deploy/`)

Note: When deploying to JBoss 3.2.3, the `properties-service.xml` file is already located in the folder: `{jboss-home}/server/default/deploy/`.

4.1.6.3 Configuring Lock out in the Authentication Manager

With version 3.2 if there is no ApplicationSecurityConfig.xml specified, then the default lockout parameters are used. The default values for the lockout parameters are as given below

- **lockout-time –** 1800000 milliseconds
- **allowed-login-time** – 60000 milliseconds
- **allowed-attempts –** 3

Alternatively, user can call provide values for the lockout parameters by using the following method of the SecurityServiceProvider Class

```
public static AuthenticationManager getAuthenticationManager(String applicationContextName,
String lockoutTime, String allowedLoginTime, String allowedAttempts) throws CSException,
CSConfigurationException
```

Also if developers are using the ApplicationSecurityConfig.xml file then they can now use the optional user lockout feature provided by CSM's default JAAS implementation of Authentication Manager. To facilitate these changes, there are three new properties that have been added to the ApplicationSecurityConfig.xml file: lockout-time, allowed-login-time, and allowed-attempts. For client application to use lockout manager, all three properties must have valid values or the lockout manager will be disabled. . To be valid, these values must be non-zero positive integers.

- **lockout-time** - This specifies the time in milliseconds that the user will be locked out after the configured number of unsuccessful login attempts has been reached.
- **allowed-login-time** - This specifies the time in milliseconds in which the configured number of unsuccessful login attempts must occur in order to lock the user out.
- **allowed-attempts** - This specifies the number of unsuccessful login attempts allowed before the use account is locked out.

Based on the values provided above the user's account is locked out for lockout-time + allowed-login-time after the allowed-attempts number has been reached.

4.1.7 Database Properties and Login Module Configuration

4.1.7.1 Requirements

In order to authenticate using the RDBMS database, developers must provide:

- The details about the database
- The actual query which will make the database calls

The CSM goal is to make authentication work with any compatible application or credential provider. Therefore we use the same Login Modules to perform authentication, and these must possess a standard set of properties.

The properties needed to establish a connection to the database include:

- **Driver** - The database driver loaded in memory to perform database operations
- **URL** - The URL used to locate and connect to the database
- **User** - The user name used to connect to the database
- **Password** - The password used to connect to the database

The following property provides the query to be used for the database to retrieve the user.

- **Query** - The query which will be fired against the RDBMS tables to verify the user id and the password passed for authentication

The *Configuring a Login Module in JAAS* section on this page shows how to configure using JAAS or the JBoss `login-config.xml` file.

4.1.7.2 Configuring a Login Module in JAAS

Developers can configure a login module for each application by making an entry in the JAAS configuration file for that application name or context.

The general format for making an entry into the configuration files is shown in Figure 4-2.

```
Application 1 {
          ModuleClass  Flag    ModuleOptions;
          ModuleClass  Flag    ModuleOptions;
          ...
      };
Application 2 {
          ModuleClass  Flag    ModuleOptions;
          ...
      };
      ...
```

Figure 4-2 Configuring a login module

For `abcapp`, which uses RDBMSLoginModule, the JAAS configuration file entry is shown in Figure 4-3.

```
abcapp
{
     gov.nih.nci.security.authentication.loginmodules.RDBMSLoginModule Required
     driver="oracle.jdbc.driver.OracleDriver" url="jdbc:oracle:thin:@cbiodb2-
     d.nci.nih.gov:1521:cbdev"
     user="USERNAME"
     passwd="PASSWORD"
     query="SELECT * FROM users WHERE username=? and password=?"
}
```

Figure 4-3 abcapp JAAS configuration file entry

The configuration file entry contains the following:

- The application is `abcapp`.
- The ModuleClass is `gov.nih.nci.security.authentication.loginmodules.RDBMSLoginModule`
- The `Required` flag indicates that authentication using this credential source is a must for overall authentication to be successful.
- The ModuleOptions are a set of parameters which are passed to the ModuleClass to perform its actions. In the prototype, the database details as well as the query are passed as parameters: `driver="oracle.jdbc.driver.OracleDriver" url="jdbc:oracle:thin:@cbiodb2-d.nci.nih.gov:1521:cbdev" user="USERNAME" passwd="PASSWORD" query="SELECT * FROM users WHERE username=? and password=?"`

Since `abcapp` has only one credential provider, only one corresponding entry was made in the configuration file. If the application uses multiple credential providers, then the LoginModules can be stacked. A single configuration file can contain entries for multiple applications.

4.1.8 Configuring a RBMS Login Module in JBoss

If an application uses the JBoss Server, developers can perform login module configuration differently. Rather than creating a JAAS configuration file, simply use the JBoss `login-config.xml` file which is located at `{jboss-home}\server\{server-name}\conf\login-config.xml`.

Shown in Figure 4-4 is the entry for the `abcapp` application:

```
<application-policy name = "abcapp">
       <authentication>
               <login-module code =
"gov.nih.nci.security.authentication.loginmodules.RDBMSLoginModule" flag = "required" >
                       <module-option name="driver"> oracle.jdbc.driver.OracleDriver</module-
option>
                       <module-option name="url">jdbc:oracle:thin:@cbiodb2-
d.nci.nih.gov:1521:cbdev</module-option>
                       <module-option name="user">USERNAME</module-option>
                       <module-option name="passwd">PASSWORD</module-option>
                       <module-option name="query">SELECT * FROM users WHERE username=? and
password=?</module-option>
               </login-module>
       </authentication>
</application-policy>
```

Figure 4-4 Example abcapp entry in login-config.xml

As shown in this example:

- The `application-policy` specifies the application for which we are defining the authentication policy which is `abcapp`.
- The login-module is the LoginModule class which is to be used to perform the authentication task; in this case it is `gov.nih.nci.security.authentication.loginmodules.RDBMSLoginModule`
- The `flag` provided is "`required`".
- The `module-options` list down the parameters which are passed to the LoginModule to perform the authentication task. In this case they are:

```
<module-option
   name="driver">oracle.jdbc.driver.OracleDriver</module-option>

<module-option name="url">jdbc:oracle:thin:@cbiodb2-
   d.nci.nih.gov:1521:cbdev</module-option>

<module-option name="user">USERNAME</module-option>

<module-option name="passwd">PASSWORD</module-option>

<module-option name="query">SELECT * FROM users WHERE username=?
   and password=?</module-option>
```

4.1.9 Enabling Encryption in the RDBMS Login Module

As part of CSM v3.2 the RDBMS Login Module is now enhanced to support encrypted passwords. CSM v3.2 now by default encrypts passwords and stores them into the CSM database. Hence if an application is using the CSM's User Table as credential provider then it needs to specify to the RDMBS Login Module to use encryption as shown below in the JBoss login-config.xml entry

```
<application-policy name = "abcapp">
       <authentication>
              <login-module code =
"gov.nih.nci.security.authentication.loginmodules.RDBMSLoginModule" flag = "required" >
                     <module-option name="driver"> oracle.jdbc.driver.OracleDriver</module-
option>
                     <module-option name="url">jdbc:oracle:thin:@cbiodb2-
d.nci.nih.gov:1521:cbdev</module-option>
                     <module-option name="user">USERNAME</module-option>
                     <module-option name="passwd">PASSWORD</module-option>
                     <module-option name="query">SELECT * FROM csm_users WHERE username=?
and password=?</module-option>
                     <module-option name="encryption-enable">YES</module-option>
              </login-module>
       </authentication>
</application-policy>
```

Where

- `Encryption-enable` option with a `YES` value uses the default CSM encryption to encrypt the user entered password before verifying it against the CSM's User Table.

4.1.10 LDAP Properties and Login Module Configuration

4.1.10.1 Requirements

The default implementation also provides an LDAP-based authentication module to be used by the client applications. In order to authenticate using the LDAP, developers must provide:

- The details about the LDAP server
- The label for the user ID Common Name (CN) or User Identification (UID) in the LDAP server

The properties needed to establish a connection to the LDAP include:

- **ldapHost** – The URL of the actual LDAP server.
- **ldapSearchableBase** – The base of the LDAP tree from where the search should begin.
- **ldapUserIdLabel** – The actual user id label used for the CN entry in LDAP.

For Ldap Credential Providers which doesn't allow anonymous binding to verify the user credentials, then in that case you need to provide the common admin user name and password as additional properties to the LDAP Login module configuration.

- **ldapAdminUserName** – The fully qualified name of the common admin user or the look up which would be used to bind to the LDAP server to be able to verify individual user ids and password
- **ldapAdminPassword** – Password for the LDAP Admin User mentioned above.

4.1.10.2 Configuring a LDAP Login Module in JAAS

For `abcapp`, which uses LDAPLoginModule, the JAAS config file entry is shown in Figure 4-5.

```
abcapp
{
     gov.nih.nci.security.authentication.loginmodules.LDAPLoginModule Required
     ldapHost= "ldaps://ncids2b.nci.nih.gov:636"
     ldapSearchableBase= "ou=nci,o=nih"
     ldapUserIdLabel="cn";
};
```

Figure 4-5 Example JAAS configuration file entry

As shown in Figure 4-5:

- The application is `abcapp`.
- The ModuleClass is `gov.nih.nci.security.authentication.loginmodules.LDAPLoginModule`.
- The `Required` flag indicates that authentication using this credential source is a must for overall authentication to be successful.
- The LDAP details are passed:

  `ldapHost="ldaps://ncids2b.nci.nih.gov:636"`

  `ldapSearchableBase= "ou=nci,o=nih"`

  `ldapUserIdLabel="cn"`

Note: Since `abcapp` has only one credential provider, only one corresponding entry was made in the configuration file. If the application uses multiple credential providers then the LoginModules can be stacked. A single configuration file can contain entries for multiple applications.

4.1.10.3 Configuring a LDAP Login Module in JBoss

If an application uses the JBoss Server, developers can perform login module configuration differently. Rather than creating a JAAS configuration file, simply use the JBoss `login-config.xml` file which is located at `{jboss-home}\server\{server-name}\conf\login-config.xml.`

Shown in Figure 4-6 is the entry for the `abcapp` application:

```
<application-policy name = "abcapp">
  <authentication>
    <login-module code =
"gov.nih.nci.security.authentication.loginmodules.LDAPLoginModule" flag = "required" >
      <module-option name="ldapHost">ldaps://ncids2b.nci.nih.gov:636</module-option>
      <module-option name="ldapSearchableBase">ou=nci,o=nih</module-option>
      <module-option name="ldapUserIdLabel">cn</module-option>
    </login-module>
  </authentication>
</application-policy>
```

Figure 4-6 Example LDAP JBoss configuration file

As shown in Figure 4-6:

- The `application-policy` is the application for which we are defining the authentication policy – in this case `abcapp`.
- The login-module is the LoginModule class which is to be used to perform the authentication task; in this case it is `gov.nih.nci.security.authentication.loginmodules.LDAPLoginModule`.
- The `flag` provided is "`required`".
- The `module-options` list down the parameters which are passed to the LoginModule to perform the authentication task. In this case they are:

  `<module-option name="ldapHost">ldaps://ncids2b.nci.nih.gov:636</module-option>`

  `<module-option name="ldapSearchableBase">ou=nci,o=nih</module-option>`

  `<module-option name="ldapUserIdLabel">cn</module-option>`

4.1.10.4 Configuring a LDAP Login Module Using Anonymous Bind

If an application uses an LDAP Server that doesn't support anonymous binds to perform a lookup, in that case you need to specify an admin (or a lookup user) id and a password to be able to bind to the LDAP server to verify user name and password. In order to do so additional parameters needs to be passed to the LDAP LoginModule entry in the JAAS Login Configuration file. Following is an entry for the same using JBoss's Login-Config.xml file

Shown in Figure 4-6 is the entry for the `abcapp` application:

```
<application-policy name = "OpenLDAP">
  <authentication>
    <login-module code =
"gov.nih.nci.security.authentication.loginmodules.LDAPLoginModule" flag = "required" >
      <module-option name="ldapHost">ldap://ncicbds-dev.nci.nih.gov:389</module-option>
      <module-option name="ldapSearchableBase">ou=csm,dc=ncicb-
dev,dc=nci,dc=nih,dc=gov</module-option>
      <module-option name="ldapUserIdLabel">uid</module-option>
      <module-option name="ldapAdminUserName">uid=csmAdmin,ou=csm,dc=ncicb-
dev,dc=nci,dc=nih,dc=gov</module-option>
      <module-option name="ldapAdminPassword">PASSWORD</module-option>
    </login-module>
  </authentication>
</application-policy>
```

Figure 4-7 Example LDAP JBoss configuration file for LDAP Servers requiring Binding

As shown in Figure 4-6:

- The `application-policy` is the application for which we are defining the authentication policy – in this case `abcapp`.
- The login-module is the LoginModule class which is to be used to perform the authentication task; in this case it is `gov.nih.nci.security.authentication.loginmodules.LDAPLoginModule`.
- The `flag` provided is "`required`".
- The `module-options` list down the parameters which are passed to the LoginModule to perform the authentication task. In this case they are:

```
<module-option
name="ldapHost">ldaps://ncids2b.nci.nih.gov:636</module-option>

<module-option name="ldapSearchableBase">ou=nci,o=nih</module-
option>

<module-option name="ldapUserIdLabel">cn</module-option>

      <module-option
name="ldapAdminUserName">uid=csmAdmin,ou=csm,dc=ncicb-
dev,dc=nci,dc=nih,dc=gov</module-option>

      <module-option name="ldapAdminPassword">PASSWORD</module-
option>
```

4.1.11 CSM – caGrid IDP Integration

As Part of v3.2, CSM is also integrated into the caGrid IDP module to facilitate local authentication. In order to support creation of SAML assertions by the IDP, CSM needs to retrieve user attributes from the Credential Providers and supply them back to the caGrid

component. In order to be able to retrieve these attributes, CSM provides configuration settings which can be used to map them to individual credential providers. These attributes are returned as CSM currently return `Principles` in a `JAAS Subject` as part of the following new method added to the AuthenticationManager

```
        public Subject authenticate(String userName, String password) throws CSException,
CSLoginException, CSInputException, CSConfigurationException,
CSInsufficientAttributesException;
```

Following are the attributes that are returned and their corresponding PrincipleNames

- **First Name** - `FirstNamePrincipal`
- **Last Name** - `LastNamePrincipal`
- **Email Id** - `EmailIdPrincipal`
- **First Name** - `LoginIdPrincipal`

Both `RDBMSLoginModule` and `LDAPLoginModule` have been updated to return these attributes. Following two sections talk about how

4.1.11.1 Configuring a LDAP Login Module for CSM – caGrid IDP Integration

If an application uses an LDAP Server from which the user attributes are to be retrieved to the above mentioned attribute mapping should be added in the JAAS login-config file. Following is a sample entry for the same in JAAS login.conf file

```
LDAPGRID{
  gov.nih.nci.security.authentication.loginmodules.LDAPLoginModule Required
  ldapHost="ldap://ncicbds-dev.nci.nih.gov:389"
  ldapSearchableBase="ou=csm,dc=ncicb-dev,dc=nci,dc=nih,dc=gov"
  ldapUserIdLabel="uid"
  ldapAdminUserName="uid=csmAdmin,ou=csm,dc=ncicb-dev,dc=nci,dc=nih,dc=gov"
  ldapAdminPassword="PASSWORD"
  USER_FIRST_NAME="givenName"
  USER_LAST_NAME="sn"
  USER_EMAIL_ID="mail";
};
```

Where

- `USER_FIRST_NAME` is the ldap attribute which stores the first name
- `USER_LAST_NAME` is the ldap attribute which stores the last name
- `USER_EMAIL_ID` is the ldap attribute which stores the email id

4.1.11.2 Configuring a RDBMS Login Module for CSM – caGrid IDP Integration

If an application uses an RDMBS Server from which the user attributes are to be retrieved to the above mentioned attribute mapping should be added in the JAAS login-config file. Following is a sample entry for the same in JAAS login.conf file

```
RDBMSGRID{
  gov.nih.nci.security.authentication.loginmodules.RDBMSLoginModule Required
  driver="org.gjt.mm.mysql.Driver"
  url="jdbc:mysql://cbiovdev5004.nci.nih.gov:3620/newAuthSchema"
  user="ncisecurity"
  passwd="ncisecurity"
  TABLE_NAME="CSM_USER"
  USER_LOGIN_ID="LOGIN_NAME"
  USER_PASSWORD="PASSWORD"
  USER_FIRST_NAME="FIRST_NAME"
  USER_LAST_NAME="LAST_NAME"
  USER_EMAIL_ID="EMAIL_ID";
};
```

Where

- `TABLE_NAME` is the name of the table where the attributes can be found
- `USER_LOGIN_ID` is the name of the column in the table storing the user's login id
- `USER_PASSWORD` is the name of the column in the table storing the user's password
- `USER_FIRST_NAME=` is the name of the column in the table storing the user's first name
- `USER_LAST_NAME=` is the name of the column in the table storing the user's last name
- `USER_EMAIL_ID=` is the name of the column in the table storing the user's email id

NOTE: In order to activate the CLM's Audit Logging capabilities for the Authentication Service, the user needs to follow the steps to deploy Audit Logging service as mentioned in the section below

4.2 Authorization

4.2.1 Introduction

The security APIs have been provided to facilitate the security needs at run time. These APIs can be used programmatically. They have been written using Java, so it is assumed that developers know the Java language.

This section outlines integration steps. For further support, CSM recommends reviewing the CSM Enterprise Architecture Model (found on the NCICB Intranet site: http://ncicbintra.nci.nih.gov/intra/caCORE/documentation).

4.2.2 Software Products

Table 4.2 displays descriptions of software products used for authorization.

Software Product	Description
JBoss Server	The JBoss/Server is the leading open source, standards-compliant, J2EE-based application server implemented in 100% Pure Java. A majority of caCORE applications use this server to host their applications.
MySQL Database	MySQL is an open source database. Its speed, scalability and reliability make it a popular choice for Web developers. CSM recommends storing authorization data in a MySQL database because it is a light database, easy to manage and maintain.
Oracle Database	Oracle's relational database was the first to support the SQL language, which has since become the industry standard. It is a proprietary database which requires licenses.
Hibernate	Hibernate is an object/relational persistence and query service for Java. CSM requires developers to modify a provided Hibernate configuration file (`hibernate.cfg.xml`) in order to connect to the appropriate application authorization schema.

Table 4- 2 Authorization software products

4.2.3 Configuration and SQL Files

File	Description
`ApplicationSecurityConfig.xml`	The XML file containing the configuration data for the appropriate application.
`hibernate.cfg.xml`	The sample XML file which contains the hibernate-mapping and the database connection details.

`AuthSchemaMySQL.sql`
OR
`AuthSchemaOracle.sql`
	This Structured Query Language (SQL) script is used to create an instance of the Authorization database schema which will be used for the purpose of authorization. In 3.0.1 and subsequent releases, this script populates the database with CSM Standard Privileges that can be used to authorize users. The same script can be used to create instances of authorization schema for a variety of applications.

File	Description
`DataPrimingMySQL.sql` OR `DataPrimingOracle.sql`	This SQL script is used for priming data in the authorization schema. Note that if the authorization database is going to host the UPT also then you need to use UPT Data Priming Scripts instead and add the application through the UPT
`mysql-ds.xml` OR `oracle-ds.xml`	This file contains information for creating a datasource. One entry is required for each database connection. Place this file in the JBoss deploy directory.

Table 4- 3 Authorization configuration and SQL files

4.2.4 Integrating CSM APIs – Overview

This section provides instruction for integrating the CSM APIs with JBoss. The integration is flexible enough to meet the needs for several scenarios depending on the number of applications hosted on JBoss and whether or not a common schema is used. Following are the scenarios:

1. JBOSS is hosting a number of applications
   a. use common schema
   b. use separate schema
2. JBOSS is hosting only one application
   a. use common schema
   b. use separate schema

4.2.4.1 Jar Placement

The CSM Application is available as a JAR which needs to be placed in the classpath of the application. Along with this JAR, there are many supporting JARs on which the CSM API depends. These should be added in the folder `<application-web-root>\WEB-INF\lib`.

4.2.5 Deployment Steps

Step 1: Create and Prime MySQL or Oracle Database

Note: When deploying Authorization, application developers may want to make use of a previously-installed common Authorization Schema. In this case, a database already exists, so skip this step. Follow the steps below to install a new Authorization Schema. Note that the Authorization Schema used by the run-time API and the UPT has to be the same.

1. Log into the database using an account id which has permission to create new databases. With CSM caCORE 3.0.1 release you can now use either MySQL or Oracle as your database of choice to host the authorization data. Based on the database you have selected, you must follow the same step during the entire installation

2. In the `AuthSchemaMySQL.sql` or `AuthSchemaOracle.sql` script, replace the

"<<database_name>>" tag with the name of the authorization schema (for e.g. "caArray").

3. Run this script on the database prompt. This should create a database with the given name. The database will include CSM Standard Privileges.

4. Now in the DataPrimingMySQL.sql or DataPrimingOracle.sql file, replace the "<<application_context_name>>" with the name of application. This is the key to derive security for the application. This will be called application context name.

5. Now in the DataPrimingMySQL.sql or DataPrimingOracle.sql file, replace the "<<super_admin_login_id>>", "<<super_admin_first_name>>" and "<<super_admin_last_name>>" with the super admin user's login id, first name and the password. NOTE: that the default password is always "changeme" and this should used for logging into the application's UPT for the first time. It should be changed immediately

6. Run this script on the database prompt. This should populate the database with the initial data. Verify this by querying the application table. It should include one record only.

Step 2: Configure Datasource

1. Modify the provided mysql-ds.xml or oracle-ds.xml file which contains information for creating a datasource. One entry is required for each database connection. Edit this file to replace:

   a. the <<application_context_name>> tag with the name of the authorization schema (for example, "***csmupt***").
   b. the <<database_user_id>> with the user id and <<database_user_password>> with the password of the user account, which will be used to access the Authorization Schema created in Step 1 above.
   c. the <<database_url>> with the URL needed to access the Authorization Schema residing on the database server.

   Shown in Figure 4-8 is an example of the mysql-ds.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>

<datasources>

  <local-tx-datasource>
    <jndi-name>csmupt</jndi-name>
    <connection-url>jdbc:mysql://cbiodev104.nci.nih.gov:3306/csmupt</connection-url>
    <driver-class>org.gjt.mm.mysql.Driver</driver-class>
    <user-name>name</user-name>
    <password>password</password>
  </local-tx-datasource>

  <local-tx-datasource>
    <jndi-name>security</jndi-name>
    <connection-url>jdbc:mysql://cbiodev104.nci.nih.gov:3306/csd</connection-url>
    <driver-class>org.gjt.mm.mysql.Driver</driver-class>
    <user-name>name</user-name>
    <password>password</password>
   </local-tx-datasource>

</datasources>
```

Figure 4-8 Example mysql-ds.xml file

2. Place the `mysql-ds.xml` or `oracle-ds.xml` file in the JBoss deploy directory.

Step 3: Create a Directory

1. Create a directory on the server where all the configuration files pertaining to the application will be kept. This directory can have any name and can reside anywhere on the server. However, it should be accessible to the JBoss id running the application.

Note: If the application is deployed on a shared server which hosts other applications that are already using CSM, this folder may already exist.

Step 4: Configure Hibernate

1. The provided `hibernate.cfg.xml` file requires modification to include configuration details to connect to the appropriate application authorization schema. For the property connection.datasource, replace the `<<upt_context_name>>` with the application name for the UPT. For example, the property may contain `java:/security` or `java:/caArray.` This application name should be same as the one created in Step 1.

2. Replace the `<<database_dialect>>` with "***`MySQLDialect`***" if you are using MySQL as the authorization database or "***`OracleDialect`***" if you are using Oracle as the authorization database.

3. In CSM v3.2 release, there is a provision for automatic detection of the hibernate configuration file. If users want to use this facility then the users can skip step 5 and 6 as they are not needed. However in this case, the default implementation of the AuthorizationManager provided by the CSM team would be used. Also the name of the hibernate configuration file should be

   `<<application_context_name>>csm.new.hibernate.cfg.xml`

   In Rename this file as

<<application_context_name>>.hibernate.cfg.xml (e.g. for caArray it will be caArray.hibernate.cfg.xml). Place this file in the directory created in Step 2. Make sure that the JBoss id has access to it.

Note: If the application requires use of a commonly installed Authorization Schema, it can use the same Hibernate configuration.

Step 5: Modify ApplicationSecurityConfig.xml

1. Edit the provided ApplicationSecurityConfig.xml as shown in Figure 4-9. Replace the <<application_context_name>> with the application name. This application name should be the same as the one created in Step 1.

```
<application>
      <context-name>
         <<application_context_name>>
      </context-name>
         <authentication>
            <authentication-provider-class>
                <!-- Fully qualified class name-->
              </authentication-provider-class>
          </authentication>
          <authorization>
               <authorization-provider-class>
                <!-- Fully qualified class name-->
             </authorization-provider-class>
            <hibernate-config-file>
               <!-- Fully qualified file path -->
                    <<hibernate_cfg_file_path>>
            </hibernate-config-file>
          </authorization>
</application>
```

Figure 4-9 Example ApplicationSecurityConfig.xml file

2. Also edit the file to replace the <<hibernate_cfg_file_path>> with the fully qualified path of the Hibernate configuration file <<application_context_name>>.hibernate.cfg.xml (for example, for caArray it will be caArray.hibernate.cfg.xml created in Step 4).

3. Place this file in the directory mentioned in Step 3. Make sure that the JBoss id has access to it.

Note: If the application is deployed on a shared server which hosts other applications that are already using CSM, this file may be present already. Note that there can only be one ApplicationSecurityConfig.xml file per JBoss installation, so simply add a new application entry to the existing file.

Step 6: Make an Addition to the JBoss Startup Properties File

1. Edit the JBoss properties-service.xml to provide a startup parameter to the

JBoss server. This file is located at `{jboss-home}/server/standard/deploy/properties-service.xml` where `{jboss-home}` is the base directory where JBoss is installed on the server. Add the following entry:

```
<attribute name="Properties"> <!-- could already exist -->
:
gov.nih.nci.security.configFile=/foo/bar/ApplicationSecurityConfig.xml
:
</attribute> <!-- could already exist -->
```

- The `gov.nih.nci.security.configFile` is the name of the property which points to the fully qualified path `foo/bar/ApplicationSecurityConfig.xml` where the `ApplicationSecurityConfig.xml` was created in Step 4. The name of the property has to be the `gov.nih.nci.security.configFile` and cannot be modified as it is a system-wide property.

2. Save this file in a deploy folder. An example is: `{jboss-home}/server/default/deploy/.`

Note: When deploying to JBoss 3.2.3, the `properties-service.xml` file is already located in the folder `{jboss-home}/server/default/deploy/.` If the application is deployed on a shared server which hosts other applications that are already using CSM, this property could be present.

Step 7: Activate CLM Audit Logging for the Authorization

1. In order to activate the CLM's Audit Logging capabilities for Authorization, the user needs to follow the steps to deploy Audit Logging service as mentioned in the section below

4.3 Audit Logging

4.3.1 Introduction

In an effort to make CSM compliant with CRF 21/ part 11, CSM will provide auditing and logging functionality. Currently CSM is using log4j for logging application logs. However, CRF21/ part 11 requires that certain messages are logged in a specific way. For example, all objects should be logged in a manner that allows them to be audited at later stage. There are two types of audit logging: Event logging and Object state logging. Audit logging capability will be provided through the Common Logging API that is available from clm.jar. Audit logging is configurable by the client application developer via an application property configuration file. By placing the clm.jar along with the application property configuration file in the same class path as the csmapi.jar file, the client application will be able to utilize the inbuilt audit logging functionality. The logging results will be saved into a database or a flat text file depending on the configuration. In addition, the logging can be enabled and

disable for any fully qualified class name.

4.3.2 Purpose

This section serves as a guide to help caCORE developers integrate Audit Logging for the CSM. This section outlines a step-by-step process that addresses what developers need to know in order to successfully integrate, including:

- Jar placement
- Configuring the JDBC Appender configuration file or the regular log4j configuration file

4.3.3 Jar Placement

The Audit Logging Application is available as a JAR, called clm.jar. This jar along with the csmapi.jar needs to be placed in the classpath of the application. If the client application is integrating the CSM API's as part of a web application on JBoss then clmwebapp.jar should be placed in the lib directory of the WEB-INF folder and the clm.jar should be placed in the common lib directory of JBoss.

4.3.4 Enabling CLM APIs in Integration with CSM APIs.

The various services exposed by CSM have been enabled for the purpose of Audit and Logging using the CLM. If configured properly, client applications using the CSM APIs can enable the internal CLM based Audit and Logging capabilities.

The CLM APIs provide the following major components of the Audit and Logging capabilities provided by CSM.

Event Logging
Both the Authentication and Authorization service have been modified to enable the logging of every event that the user performs. For Authentication Services, the CSM APIs log the login and logout events of the user. In addition, when a user lockout event occurs, a log is generated that records the username that was locked out. For Authorization Service the CSM APIs track all create, update and delete operations that the client application invokes. The 'read' operations are not logged because they are not needed for Audit and Logging.
The UPT can perform all of the audit and logging services because it uses the CSM APIs (which use CLM APIs) to perform operations on the database.

Since the CLM APIs are based on log4j, the following logger names are used in the CSM APIs to perform the event logging.

Authentication Event Logger Name:
```
CSM.Audit.Logging.Event.Authentication
```
Authorization Event Logger Name:
```
CSM.Audit.Logging.Event.Authorization
```

The log4j log level used for all the event logs is INFO

In order to enable these loggers, they should be configured in the `log4j.xml` config file of Jboss.

Object State Logging
The Authorization Service of the CSM is enabled to log the object state changes using the automated object state logger available through CLM APIs. This logger tracks all the object state changes that are made using the CSM APIs. It also uses the log4j based CLM APIs and the following Logger Name:

Authorization Object State Logger Name:
`CSM.Audit.Logging.ObjectState.Authorization`

The log4j log level used for all the object state logs is INFO

In order to enable object state logging for CSM APIs the above mentioned logger should be configured in the log4j.xml config file of JBoss.

User Information
In order to track which user is performing the specific operation for the purpose of Audit Logging, CSM needs to know user information like user id and session id and also the organization to which the user belongs. Since these values are only available with the client application, they need to be passed to the CSM APIs. To accomplish this, the client application must use the utility class "UserInfoHelper" provided by the underlying CLM APIs. This information needs to be set before calling any of the create, update or delete functions of the CSM APIs.

Common Logging Database
This is the persistence storage that the JDBC appender uses to store the Audit Logs. The Log Locator application of CLM connects to this database to allow the user to browse the logs.

JDBC Appender
To persist these Audit logs the CLM provides an asynchronous JDBC Appender. Thus, an application that wants to enable the audit logging for CSM APIs should also configure this Appender. A sample log4j entry is show below.

```
<?xml version="1.0" encoding="UTF-8" ?>


<!DOCTYPE log4j:configuration SYSTEM ".\log4j.dtd">

<log4j:configuration xmlns:log4j='http://jakarta.apache.org/log4j/'>
    <appender name="CLM_APPENDER" class="gov.nih.nci.logging.api.appender.jdbc.JDBCAppender">
      <param name="application" value="csm" />
      <param name="maxBufferSize" value="1" />
      <param name="dbDriverClass" value="org.gjt.mm.mysql.Driver" />
      <param name="dbUrl" value="jdbc:mysql://<<SERVER_NAME>>:<<PORT>>/<<CLM_SCHEMA_NAME>>" />
      <param name="dbUser" value="<<DB_USER>>" />
      <param name="dbPwd" value="<<PASSWORD>>" />
          <param name="useFilter" value="true" />
      <layout class="org.apache.log4j.PatternLayout">
           <param name="ConversionPattern"
                       value=":: [%d{ISO8601}] %-5p %c{1}.%M() %x - %m%n" />
      </layout>
    </appender>

    <category name="CSM.Audit.Logging.Event.Authentication">
                      <level value="info" />
                      <appender-ref ref="CLM_APPENDER" />
       </category>
       <category name="CSM.Audit.Logging.Event.Authorization">
                      <level value="info" />
                      <appender-ref ref="CLM_APPENDER" />
```

Figure 4-10 Example log4j.xml file

NOTE: CSM is capable of performing both event and object state audit logging only for the operations and data pertaining to CSM. In order to use the similar functionality, the client application can separately download and install CLM. In this case CLM can be used (even without using CSM) to provide event logging and automated object state logging capabilities using the special appender and schema. Also the log locator tool can be used for the purpose of viewing the logs.

4.3.5 Deployment Steps

In order for a client application to enabling the Audit Logging capabilities provided by CSM (via CLM), the following steps must be performed:

Step 1: Create and Prime MySQL Logging Database

1. A database has to be created which will persist the audit logs that are generated as a basis of usage of the CSM APIs
2. Refer to the CLM's guide for application developers for creating and priming the database for storing the audit logs.

Step 2: Configure the log4j.xml file for JBoss

1. Use the sample log4j file provided in the CSM's release to configure the log4j.xml file for JBoss. (see figure 4-9 above)
2. Replace the <<SERVER_NAME>>, <<PORT>> and the <<CLM_SCHEMA_NAME>> with the fully corresponding values where the schema created in Step 1 is hosted.
3. Replace the values for the <<DB_USER>> with the user name that has access on the schema. Also replace the <<PASSWORD>> with the corresponding password for the user.
4. Based on whether the application wants to enable the event audit logging for Authentication & Authorization or object state audit logging for the Authorization; the corresponding logger needs to be configured. **Note:** The name of the loggers must be exactly the same as mentioned in the sample.
5. Incase of UPT the same log4j config file can be used.

Step 3: View the Logs

1. CLM provides a web-based locator tool that can be used to browse audit logs.
2. The configuration steps for setting up the browser are mentioned in the CLM's guide for application developers.

4.4 Provisioning

4.4.1 Introduction

UPT is a web application used to provision an application's authorization data. The UPT provides functionality to create authorization data elements like Roles, Protection Elements, Users, etc., and also provides functionality to associate them with each other. The runtime API can then use this authorization data to authorize user actions.

This section of the guide explains how to deploy the UPT from start to finish – from uploading the Web Application Archive (WAR) and editing configuration files, to synching the UPT with the application. See the section Integrating with the User Provisioning Service if you need to integrate with an existing UPT deployment.

This section details the UPT release contents, explains multiple ways in which the UPT can be deployed, and outlines the steps that result in a successful deployment.

4.4.2 UPT Release Contents

The UPT is released as a compressed web application in the form of a WAR (Web Archive) File. Along with the WAR, the release includes sample configuration files that help developers configure the UPT with their application(s).

The UPT Release contents can be found in the UPT.zip file found on the NCICB download site (http://ncicb.nci.nih.gov/download/index.jsp). The UPT Release contents include the files in Table 4.4.

File	Description
csmupt.war	The UPT Web Application
ApplicationSecurityConfig.xml	The XML file containing the configuration data for the UPT.
Hibernate.cfg.xml	The sample XML file which contains the hibernate-mapping and the database connection details.
AuthSchemaMySQL.sql OR AuthSchemaOracle.sql	This Structured Query Language (SQL) script is used to create an instance of the Authorization database schema which will be used for the purpose of authorization. In the 3.0.1 and subsequent releases, this script populates the database with CSM Standard Privileges that can be used to authorize users. The same script can be used to create instances of authorization schema for a variety of applications.
DataPrimingMySQL.sql OR DataPrimingOracle.sql	This SQL script is used for priming data in the UPT's authorization schema.
mysql-ds.xml OR oracle-ds.xml	This file contains information for creating a datasource. One entry is required for each database connection. Place this file in the JBoss deploy directory.

Table 4- 4 UPT release contents

4.4.3 UPT Installation Modes

UPT was developed as a flexible application that can be deployed in multiple ways depending on the need or scenario. The three primary modes to install the UPT include the following and are described in the following sections:

- Single Installation, Single Schema
- Single Installation, Multiple Schemas
- Local installation, Local schema

4.4.3.1 Single Installation, Single Schema

In the single installation, single schema deployment scheme as shown in Figure 4-11, there is only one instance of UPT hosted on a Common JBoss Server. A common installation is used to administer the authorization data for all applications. The authorization data for all the applications is stored on a common database. Therefore an application using UPT does not have to install its own authorization schema. Also, all applications can use the same `hibernate-config` file since they point to the same database.

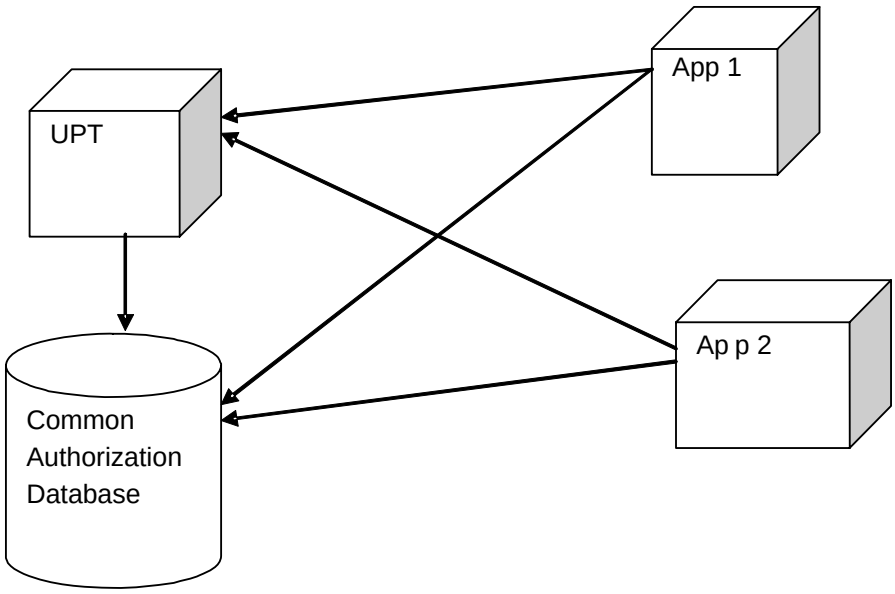

Figure 4-11 Single installation, single schema deployment scheme

4.4.3.2 Single Installation, Multiple Schemas

As in the single schema deployment, the single installation, multiple schemas deployment calls for the UPT to be hosted on a single JBoss Common Server as shown in Figure 4-12. A common installation is also used to administer the authorization data for all applications. What makes this mode different is that an application can use its own authorization schema on a separate database if preferred. The authorization data can sit on individual databases, and at the same time some applications can still opt to use the Common Authorization Schema. Using this mode requires each application to maintain its own `hibernate-config` file pointing to the database where its Authorization Schema is located. So when an application uses the UPT, the UPT communicates to the authorization schema of that application only.

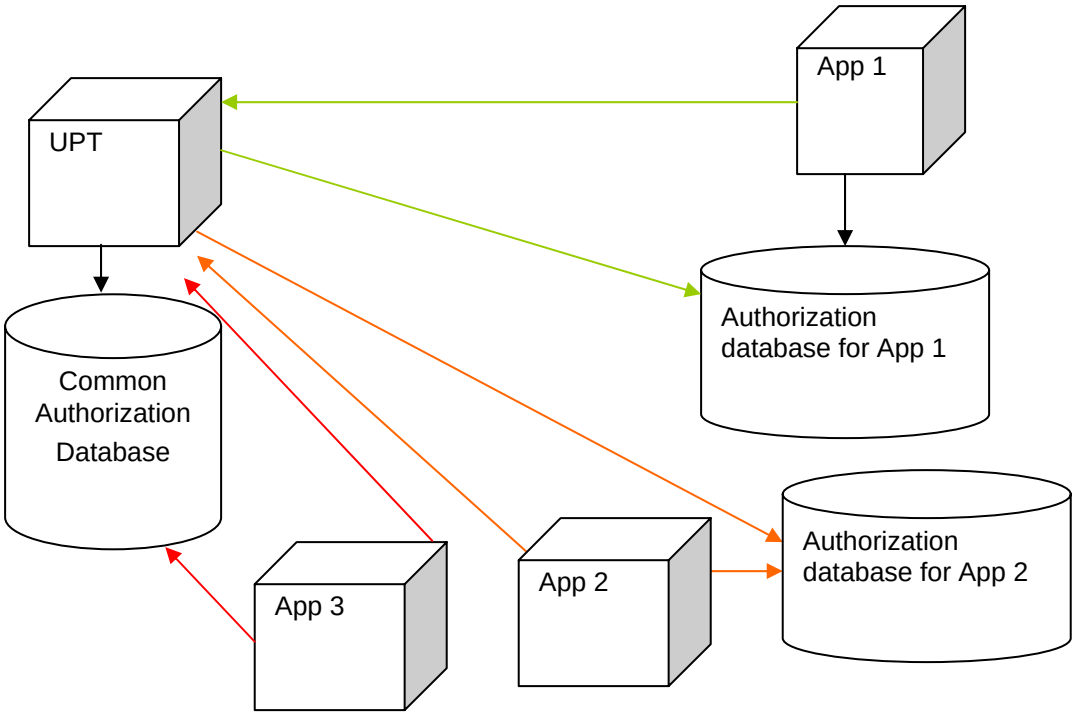

Figure 4-12 Single installation, multiple schemas deployment scheme; the three colors of arrows correspond to the three different applications shown

4.4.3.3 Local Installation, Local Schema

The local installation, local schema deployment is the same as single installation, single schema, except that the UPT is hosted locally by the application as shown in Figure 4-13. This installation of UPT is not shared with other applications. This local installation is used to administer the authorization data for that particular application (or set of related applications) only. The authorization data for the application sits on its own database. In this scenario, the application requires its own `hibernate-config` file pointing to the database where its Authorization Schema is located.

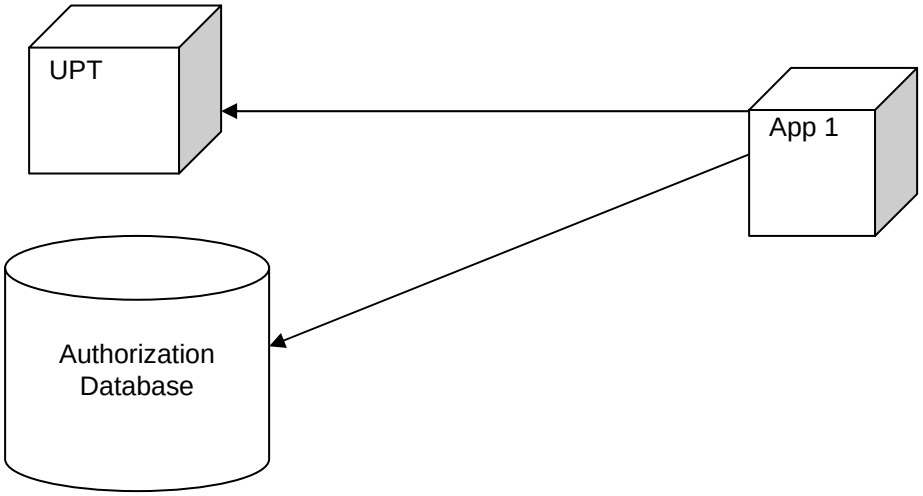

Figure 4-13 Local installation, local schema deployment scheme

4.4.4 Deployment Checklist

Before deploying the UPT, verify the following environment and configuration conditions are met. This software and access credentials/parameters are required.

- Environment
  - JBoss 4.0 Application Server
  - MySQL 4.0 OR Oracle 9i Database Server (with an account that can create databases)
- UPT Release Components
  - csmupt.war
  - AuthSchemaMySQL.sql | AuthSchemaOracle.sql
  - DataPrimingMySQL.sql | DataPrimingOracle.sql
  - ApplicationSecurityConfig.xml
  - hibernate-config file

4.4.5 Deployment Steps

Step 1: Create and Prime MySQL or Oracle Database

1. Log into the database using an account id which has permission to create new databases. As you follow the deployment steps, use the files containing the name corresponding with your database. Make sure that the database you are about to create doesn't already exist. If it does, then drop it to recreate new one.
2. In the AuthSchemaMySQL.sql file replace the <<database_name>> tag with the name of the UPT Authorization schema – csmupt.
3. Run this script on the database prompt. This should create a database with the given name.
4. In the DataPrimingMySQL.sql file, replace:
    - The `<<super_admin_login_id>>` with the login id of the user who is going to act as the Super Admin for that particular installation
    - Also provide the first name and last name for the same by replacing `<<super_admin_first_name>>` with first name and `<<super_admin_last_name >>` with last name.
5. Replace the `<<application_context_name>>` with a application name of the application for which UPT is being hosted
6. Run the script on the database prompt. This should populate the database with the initial data. Verify by querying the csm_application, csm_user, csm_protection_element and csm_user_protection_element tables. They should have one record each. The database will include CSM Standard Privileges and the csm_privilege table should have 7 entries.

Step 2: Configure Datasource

1. Modify the mysql-ds.xml or oracle-ds.xml file which contains information for creating a datasource. One entry is required for each database connection. Edit this file to replace:
    - The `<<application_context_name>>` tag with the name of the authorization schema (for example, "***csmupt***").
    - The <<database_user_id>> with the user id and `<<database_user_password>>` with the password of the user account, which will be used to access the Authorization Schema created in Step 1 above.
    - The `<<database_url>>` with the URL needed to access the Authorization Schema residing on the database server.

Shown in Figure 4-14 is an example `mysql-ds.xml` file.

```
<?xml version="1.0" encoding="UTF-8"?>

<datasources>

  <local-tx-datasource>
    <jndi-name>csmupt</jndi-name>
    <connection-url>jdbc:mysql://cbiodev104.nci.nih.gov:3306/csmupt</connection-url>
    <driver-class>org.gjt.mm.mysql.Driver</driver-class>
    <user-name>name</user-name>
    <password>password</password>
  </local-tx-datasource>

  <local-tx-datasource>
    <jndi-name>security</jndi-name>
    <connection-url>jdbc:mysql://cbiodev104.nci.nih.gov:3306/csd</connection-url>
    <driver-class>org.gjt.mm.mysql.Driver</driver-class>
    <user-name>name</user-name>
    <password>password</password>
   </local-tx-datasource>

</datasources>
```

Figure 4-14 Example mysql-ds.xml file

2. Place the `mysql-ds.xml` or `oracle-ds.xml` file in the JBoss deploy directory.

NOTE: For v3.2 of the UPT, there is not need to follow the step 3,4, 5, and 6.

Step 3: Create Directory

1. Create a directory on the server where all the configuration files pertaining to the UPT will be kept. This directory can have any name and can reside anywhere on the server. However, it should be accessible to the JBoss id running the UPT.

Step 4: Configure Hibernate

1. The provided `hibernate.cfg.xml` file needs to be modified to include configuration details to connect to the appropriate UPT Authorization Schema. For the property connection.datasource, replace the `<<upt_context_name>>` with the application name for the UPT. For example, the property may contain `java:/upt or java:/csmupt`. This application name should be the same as the one created in Step 1.

2. Replace the `<<database_dialect>>` with "***`MySQLDialect`***" if you are using MySQL as the authorization database or "***`OracleDialect`***" if you are using Oracle as the authorization database.

3. Rename this file to `upt.hibernate.cfg.xml` (add upt Prefix). Place this file in the directory created in Step 3. Make sure that the JBoss id has access to it.

Step 5: Modify ApplicationSecurityConfig.xml

1. Edit the provided ApplicationSecurityConfig.xml.

2. Replace the `<<upt_context_name>>` with the application name for the UPT. This application name should be same as the one created in Step 1.

3. Replace the <<hibernate_cfg_file_path>> with the fully qualified path of the hibernate configuration file upt.hibernate.cfg.xml created in Step 3.

4. Place this file in the directory. Make sure that the JBoss id has access to it.

Step 6: Make an Addition to the JBoss Startup Properties File

1. Edit the JBoss properties-service.xml to provide a startup parameter to the JBoss server. This file is located at the following path: {jboss-home}/server/standard/deploy/properties-service.xml where {jboss-home} is the base directory where JBoss is installed on the server. Add the following entry to the existing properties:

```
<attribute name="Properties"> <!-- could already exist -->
:
gov.nih.nci.security.configFile=/foo/bar/ApplicationSecurityConfig.xml
:
</attribute> <!-- could already exist -->
```

   - The gov.nih.nci.security.configFile is the name of the property which points to the fully qualified path foo/bar/ApplicationSecurityConfig.xml where the ApplicationSecurityConfig.xml has been created in Step 4. The name of the property must be the gov.nih.nci.security.configFile and cannot be modified, as it is a system-wide property.

2. Save this file in a deploy folder (for example, {jboss-home}/server/default/deploy/).

Note: When deploying to JBoss 3.2.3, the properties-service.xml file is already located in the folder {jboss-home}/server/default/deploy/.

Step 7: Configure the JBoss JAAS Login Parameters

The suggested Authentication Credential Provider for UPT is NCICB LDAP. In order to configure the UPT to verify against the LDAP, create an entry in the login-config.xml of JBoss as shown in Figure 4-15. This entry configures a login-module against the UPT application context. The location of this file is {jboss-home}/server/default/conf/login-config.xml where {jboss-home} is the base directory where JBoss is installed on the server.

```
<application-policy name = "abcapp">
  <authentication>
    <login-module code =
"gov.nih.nci.security.authentication.loginmodules.LDAPLoginModule"
flag = "required" >
      <module-option
   name="ldapHost">ldaps://ncids2b.nci.nih.gov:636</module-option>
      <module-option name="ldapSearchableBase">ou=nci,o=nih</module-
option>
      <module-option name="ldapUserIdLabel">cn</module-option>
    </login-module>
  </authentication>
</application-policy>
```

Figure 4-15 Example login-config.xml entry

As shown in Figure 4-15:

- The `application-policy` is the name of the application for defining the authentication policy – in this case, "`abcapp`".
- The `login-module` is the `LoginModule` class which is used to perform the authentication task; in this case, it is `gov.nih.nci.security.authentication.loginmodules.LDAPLoginModule`.
- The `flag` provided is "`required`".
- The `module-options` list the parameters which are passed to the `LoginModule` to perform the authentication task. In this case, they are pointing to the NCICB LDAP Server:

```
<module-option name="ldapHost">ldaps://ncids2b.nci.nih.gov:636</module-
   option>
<module-option name="ldapSearchableBase">ou=nci,o=nih</module-option>
<module-option name="ldapUserIdLabel">cn</module-option>
```

Step 8: Deploy the UPT WAR

1. Copy the UPT `upt.war` in the deployment directory of JBoss which can be found at `{jboss-home}/server/default/deploy/` where `{jboss-home}` is the base directory where JBoss is installed on the server.

Step 9: Activate CLM Audit Logging for the UPT Tool

1. In order to activate the CLM's Audit Logging capabilities for UPT, the user needs to following the steps to deploy Audit Logging service as mentioned in the section above.
2. Also the clm.jar needs to be placed in the common lib directory of the JBoss server

Step 10: Start JBoss

1. Once the deployment is completed, start JBoss. Check the logs to confirm there are no errors while the UPT application is deployed on the server.
2. Once the JBoss server has completed deployment, open a browser to access the UPT. The

URL will be `http://<<jboss-server>>/upt`, where the `<<jboss-server>>` is the IP or the DNS name of JBoss Server.

3. The UPT Login Page displays. Enter the UPT Application using the login-id that was assigned to the Super Admin in Step 1 and its password. Also use the UPT Application Name specified in Step 4 for the Application Name.

4. You should be able to login successfully and the UPT Application Home Page displays.

Note: In case of any errors, follow a debugging and trouble shooting procedure to diagnose and solve the issues.

Step 10: Add a New Application

Once the initial setup of UPT is complete, UPT is up and available for the applications to start using provisioning for their authorization data. However, applications must be registered and configured before they can start using the UPT.

1. To register an application, use the UPT front end user interface to create an entry for the new application. Login as a **Super Admin**, go to the **Application** section, and select **Create a New Application**. Once the details are entered, go to the **User** section to create **Users**. Then return to the **Application** section to assign these **Users** as **Application Administrators**.

   For v3.2 of UPT, users can additionally provide the database connection parameters directly into the front end at the time of creating the application. The following are the new parameters which can be added for the newly created application.

   - `Database URL = <<URL of the database>>`
   - `Database Username = <<Username of the Database>>`
   - `Database Password = <<Password of the Database>>`
   - `Database Confirm Password = <<Password of the Database>>`
   - `Database Driver = <<Driver for the Database>>`
   - `Database Dialect = <<Dialect for the database>>`

   If these values are provided there is no need for to perform step 2. The admin users can directly log into their respective application.

2. For v3.2 of UPT, users need

   Once the application registration is complete, it needs to be configured. First, make a new "application" entry in the `ApplicationSecurityConfig.xml` file. (Use the existing UPT application entry as a template - copy, paste, and modify for the new application.)

   a. Replace the `<context-name>` with the new application name.

   b. If the Application will use the default CSM provided Authorization Manager, then leave the `<authorization-provider-class>` blank.

   c. Replace the hibernate-config qualified path to point to the application's hibernate-

config file. (Make sure the hibernate-config file resides in the correct location.) If the application is going to use the Common Authorization Schema (which also hosts the Schema for the UPT itself), then it can use the same `hibernate-config` file. In that case, just copy the entry from UPT's configuration.

5. Integrating with the CSM Authentication Service

5.1 Importing and Using the CSM Authentication Manager Class

To use the CSM Service, add the highlighted import statements (last two) as shown in Figure 5-1 to the action classes that require authentication.

```
import gov.nih.nci.abcapp.UserCredentials;
import gov.nih.nci.abcapp.model.Form;
import gov.nih.nci.abcapp.util.Constants;
import gov.nih.nci.security.SecurityServiceProvider;
import gov.nih.nci.security.AuthenticationManager;
```

Figure 5-1 Example ABC application - Import statements in an action class

The class `SecurityServiceProvider` is the common interface class exposed by the CSM application. It contains methods to obtain the correct instance of the `AuthenticationManager` configured for that application. The client application `abcapp` then uses the `AuthenticationManager` to perform the actual authentication using the CSM.

Figure 5-2 illustrates an example of how to use the `CSMService` class in the ABC application.

```
UserCredentials credentials = new UserCredentials();
credentials.setPassword(Form.getPassword());
credentials.setUsername(Form.getUsername());

//Get the user credentials from the database and login
try {

        AuthenticationManager authenticationManager =
  SecurityServiceProvider.getAuthenticationManager("abcapp");
        boolean loginOK = authenticationManager.login(credentials.getUsername(),
  credentials.getPassword());
        if (loginOK)
        {
          System.out.println(">>>>>>>>>>>>> SUCESSFUL LOGIN <<<<<<<<< ");
        }
        else
        {
          System.out.println(">>>>>>>>>>>>> ERROR IN LOGIN <<<<<<<<< ");
        }
  }
catch (CSException cse){
  System.out.println(">>>>>>>>>>>>> ERROR IN LOGIN <<<<<<<<< ");
  }
```

Figure 5-2 Example code to use the CSMService class in the ABC application

The client class obtains the default implementation of the `AuthenticationManager` by calling the static `getAuthenticationManager` method of the `SecurityServiceProvider` class by passing the application Context name – in this example "`abcapp`". It then invokes the login method - passing the user's ID and password. Note that the application name should match the name used in the configuration files for JAAS to work correctly. If the credentials provided are correct then a Boolean true is returned indicating that the user is authenticated. If there is an authentication error, a `CSException` is thrown with the appropriate error message embedded.

6. Integrating with the CSM Authorization Service

6.1 Importing and Using the CSM Authorization Manager Class

To use the CSM Service, add the highlighted import statements (last two) as shown in Figure 6-1 to the action classes that require authorization.

```
import gov.nih.nci.abcapp.UserCredentials;
import gov.nih.nci.abcapp.model.Form;
import gov.nih.nci.abcapp.util.Constants;
import gov.nih.nci.security.SecurityServiceProvider;
import gov.nih.nci.security.AuthorizationManager;
```

Figure 6-1 Example ABC application - Import statements in an action class

The class `SecurityServiceProvider` is the common interface class exposed by the CSM application. It contains methods to obtain the correct instance of the `AuthorizationManager` configured for that application. The client application `abcapp` then uses the `AuthorizationManager` to perform the actual authentication using the CSM.

Figure 6-2 illustrates an example of how to use the CSMService class in the ABC Application.

```
try {

            AuthorizationManager authorizationManager =
      SecurityServiceProvider.getAuthorizationManager("abcapp");
            boolean hasPermission = authorizationManager.checkPermission("user name" ,
      "resource name", "operation" );
            if (hasPermission)
            {
              System.out.println(">>>>>>>>>>>>> PERMISSION GRANTED <<<<<<<<< ");
            }
            else
            {
              System.out.println(">>>>>>>>>>>>>PERMISSION DENIED <<<<<<<<< ");
            }
      }
catch (CSException cse){
      System.out.println(">>>>>>>>>>>>> ERROR IN AUTHORIZATION <<<<<<<<< ");
      }
```

Figure 6-2 Example code to use the CSMService class in the ABC application

The client class obtains the default implementation of the `AuthorizationManager` by calling the static `getAuthorizationManager` method of the `SecurityServiceProvider` class by passing the application Context name – in this example "`abcapp`". It then invokes the `checkPermission` method – passing the user's ID, the resources which it is trying to access and the operation which it wants to perform. Note that the application name should match the name used in the configuration files as well as configured in the databases for authorization to work correctly. If the user has the required access permission, then a Boolean true is returned indicating that the user is authenticated. In case of any authorization error, a `CSException` is thrown with the appropriate error message embedded.

7. Enabling CLM with the CSM

This section's intended audience is developers wishing to enable CLM APIs for their CSM Integration. (For a complete guide to CLM deployment and configuration, see the section above)

1. The client application must first set the user information before invoking any method calls on the CSM APIs. Use the "`UserInfoHelper`" class from the CLM APIs to set

the user name and session id before each call to the CSM APIs. If the application is not a web-based, then pass null. Technically this needs to be set only once for a thread of operation intiated by the user. For web applications, it can be set only once for each request at the start of the `doPost` method of the servlet (or similar). The following example is a code snippet for the applications to integrate CLM.

```
package test;

import java.io.*;

import javax.servlet.http.*;
import javax.servlet.*;
import gov.nih.nci.logging.api.user.UserInfoHelper;

public class HelloServlet extends HttpServlet {
  public void doPost (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
   UserInfoHelper.setUserInfo(new String("NAME"), new String("SESSIONID"));
   PrintWriter out = res.getWriter();
// invoke some CSM call
    out.println("Hello, world!");
    out.close();
  }
}
```

2. The UPT itself is a client of the CSM APIs. It sets the user name and session id for the user who has logged into the UPT for the purpose of audit logging. Thus when the Audit logging is enabled for UPT,it will start logging the user name and session id along with all the create, update or delete operations that the user performs.

NOTE: The CLM object state logger has issues logging if the transaction managers are set in the hibernate.cfg.xml file when deployed on JBoss server. In this case the transaction manager properties should be removed from the hibernate.cfg.xml file used for CSM APIs to connect to the common authorization schema.

8. Integrating with the User Provisioning Service

This section's intended audience is developers wishing to integrate their application(s) with an existing UPT Deployment. (For a complete guide to UPT deployment, see the section above) Once the initial setup of UPT is complete, UPT is up and available for the applications to start using provisioning for their authorization data. However, applications must be registered and configured before you can start using the UPT.

1. To register an application, use the UPT front end user interface to create an entry for the new application. Simply login as a **Super Admin**, go to the **Application** section, and select **Create a New Application**. Once the details are entered, go to the **User** section to create **Users**. Then return to the **Application** section to assign these **Users** as Application **Administrators**. (If you're not the **Super Admin**, ask him/her to add your **Application** and you as an **Admin**.)

2. Once the application registration is complete, it needs to be configured. First, make a new "application" entry in the ApplicationSecurityConfig.xml file. (Use the existing UPT application entry as a template - copy, paste, and modify for the new application.)
3. Replace the <context-name> with the new application name
4. If the Application will use the default CSM-provided Authorization Manager, then leave the <authorization-provider-class> blank.
5. Replace the `hibernate-config` qualified path to point to the application's `hibernate-config` file. (Make sure the `hibernate-config` file resides in the correct location.) If the application is going to use the Common Authorization Schema (which also hosts the Schema for the UPT itself), then it can use the same `hibernate-config` file. In that case, copy the entry from the UPT's configuration.