

CACORE CSM 4.0 TECHNICAL GUIDE



Center for Bioinformatics

This is a U.S. Government Work

November 30, 2007



caBIG™ cancer Biomedical
Informatics Grid™



Credits and Resources

<i>caCORE CSM Development and Management Teams</i>			
<i>CSM Development Team</i>	<i>Other Development Teams</i>	<i>Guide</i>	<i>Program Management</i>
Vijay Parmar ¹	Satish Patel ¹	Vijay Parmar ¹	Avinash Shanbhag ³
Kunal Modi ¹	Dan Dumitru ¹	Kunal Modi ¹	Charles Griffin ¹
Aynur Abdurazik ²		Charles Griffin ¹	
		Wendy Erickson-Hirons ⁴	
¹ Ekagra Software Technologies	² Science Applications International Corporation (SAIC)	³ National Cancer Institute Center for Bioinformatics	⁴ Northern Taiga Ventures, Inc.

<i>Contacts and Support</i>	
NCICB Application Support	http://ncicb.nci.nih.gov/NCICB/support Telephone: 301-451-4384 Toll free: 888-478-4423

Submitting a Support Issue

A GForge Support tracker group, which is actively monitored by CSM developers, has been created to track any support requests. If you believe there is a bug/issue in the CSM software itself, or have a technical issue that cannot be resolved by contacting the [NCICB Application Support](#) group, please submit a new support tracker using the following link: https://gforge.nci.nih.gov/tracker/?atid=131&group_id=12&func=browse. Make sure to review any existing support request trackers prior to submitting a new one in order to help avoid duplicate submissions.

Release Schedule

This guide has been updated for the caCORE CSM 4.0 release. It may be updated between releases if errors or omissions are found. The current document refers to the 4.0 version of caCORE CSM, released in November 2007 by the NCICB.

NOTE: In November of 2008, this guide was updated to fix an error found by developers. The term ‘encryption-enable’ was changed to ‘encryption-enabled’ where appropriate.

Table of Contents

Chapter 1	Using This Guide	1
	Purpose	1
	Release Schedule	1
	Audience	1
	Organization of This Guide	2
	Additional caCORE Documentation	2
	Document Text Conventions	2
Chapter 2	CSM Overview	5
	Introduction	5
	Security Concepts	6
	Minimal System Requirements	8
Chapter 3	Using the CSM API	9
	Workflow	9
	API Services	10
	AuthenticationManager	10
	AuthorizationManager	10
	Authentication	10
	Integrating with the CSS Authentication Service	10
	Importing the CSM Authentication Manager Class	10
	Using the CSM Authentication Manager Class	11
	Installation and Deployment Configurations	12
	JAR Placement	12
	Configuring Lockout in Authentication Manager	12
	RDBMS Credential Provider Properties and Login Module Configuration	13
	LDAP Credential Provider Properties and Login Module Configuration	16
	Activating CLM Audit Logging	20
	Authorization	20
	Overview to Integrating CSM APIs	20
	Integrating with the CSM Authorization Service	20
	Software Products and Scripts	21
	Installation and Deployment Configurations	23
	Jar Placement	23
	Database Properties and Configuration	23
	Activate CLM Logging	25
	User Provisioning Tool	25
	Audit Logging	26
	Introduction	26
	Jar Placement	26
	Enabling CLM APIs in Integration with CSM APIs	26
	Deployment Steps	29

Chapter 4	Using the User Provisioning Tool	31
Introduction		31
Workflow		31
Super Admin		31
Admin.....		31
Login		32
Common Basic Functions.....		32
Create New Element		32
Example Error Messages		33
Search For and Select Existing Elements		34
Update an Element.....		35
Delete an Element		36
Assignments and Associations.....		36
Assign or Deassign Privileges, Roles, ProtectionGroups, Groups.....		36
Assign or Deassign Users and Protection Elements.....		37
Super Admin Mode		39
Overview.....		39
Workflow.....		39
Navigation.....		39
Application		40
Managing Users		41
Assigning Privileges		42
Admin Mode.....		44
Overview.....		44
Workflow.....		46
Navigation.....		46
UPT Installation and Deployment.....		58
Release Contents		58
Installation Modes.....		59
Deployment Checklist.....		62
Deployment Steps		62
Chapter 5	Using the CSM Web Services	67
Introduction		67
Web Service WSDL and Operation		67
Security Web Service WSDL.....		67
Login Operation		67
CheckPermission Operation.....		69
Workflow for CSM Security Web Service		71
Installation of CSM Security Web Service		71
Step 1: Create and Prime Database		71
Step 2: Configure Datasource		72
Step 3: Configure the JBoss JAAS Login parameters		72
Step 4: Deploy the Security WS war file.....		73

Chapter 6	CSM Instance Level and Attribute Level Security	75
Introduction		75
Instance Level		75
Requirements Addressed		75
Overall Design		76
Provisioning Instance Level Security		76
Using Instance Level Security		79
Known Issues		80
Attribute Level		80
Requirements Addressed		80
Overall Design		81
Provisioning Attribute Level Security		81
Using Attribute Level Security		81
Known Issues		82
Chapter 7	CSM Acegi Adapter	83
Introduction		83
Implementation		83
Method Level Security		84
Method Parameter Level Security		85
Workflow		85
Integrating and Configuring		85
Configure Acegi Security		85
Database Properties and Configuration		86
Configure JAAS LoginModule		90
User Provisioning via UPT		92
Chapter 8	CSM caGrid Integration	93
Introduction		93
Authentication		93
CSM Configuration for IdP/Authentication Service		93
Authorization		95
Using Grid Group Names for Check Permission		95
Migrating from CSM v3.2 to CSM v4.0		96
MySQL Migration		96
Oracle Migration		97
Appendix A	References	99
Articles		99
caBIG Material		99
caCORE Material		99
Software Products		99
Appendix B	CSM Acegi Sample Configuration File	101

Glossary.....	105
Index.....	107

Chapter 1 Using This Guide

This chapter introduces you to the caCORE CSM Technical Guide.

Topics in this chapter include:

Purpose Release Schedule Audience Additional caCORE Documentation Organization Document Text Conventions

- [Purpose](#) on this page
- [Release Schedule](#) on this page
- [Audience](#) on this page
- [Organization of This Guide](#) on page 2
- [Document Text Conventions](#) on page 2

Purpose

This guide 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.

This document is a master document that covers all CSM modules and shows how to deploy and integrate the CSM services, including Authentication, Authorization, User Provisioning Tool, CSM Security Web Services, CSM Acegi Adapter, and CSM caGrid Integration. This document covers the User Guide and Application Developers Guide for all modules of CSM including CSM API, CSM UPT, CSM Security Web services, CSM Acegi Adapter, and CSM caGrid Integration.

Release Schedule

This guide is updated for each caCORE CSM release. It may be updated between releases if errors or omissions are found. The current document refers to the 4.0 version of caCORE CSM, which was released in November 2007 by the NCI CBIIT (formerly the National Cancer Institute Center for Bioinformatics (NCICB)).

NOTE: In November of 2008, this guide was updated to fix an error found by developers. The term 'encryption-enable' was changed to 'encryption-enabled' where appropriate.

Audience

The primary audience of this guide is the application developer who wants to integrate security. This guide assumes that you are familiar with the Java programming language and/or other programming languages, database concepts, and the Internet. If you intend to use caCORE CSM resources in software applications, it assumes that you have experience with building and using complex data systems.

Organization of This Guide

This brief overview explains what you will find in each section of this guide.

- [Chapter 1](#), this chapter, provides an overview of this guide.
- [Chapter 2](#) and [Chapter 3](#) provide detailed knowledge and workflow for a user to successfully integrate CSM into their applications.
- [Chapter 4](#) provides the workflow and details about the Authorization Policy provisioning necessary to use CSM for Authentication or Authorization.
- [Chapter 5](#) explains how to expose the CSM authentication and Authorization service features to web service consumers.
- [Chapter 6](#) describes Instance and Attribute level security in CSM v4.0.
- [Chapter 7](#) explains how method level and method parameter level security is implemented and available out of the box for applications that use or want to use Acegi and leverage CSM Authentication and Authorization features. This chapter also provides a workflow and steps necessary to integrate the CSM Acegi adapter into existing or new applications using the Acegi framework.
- [Chapter 8](#) describes how to level CSM in the caGrid environment.
- [Appendix A](#) provides a list of references used to produce this guide or referred to within the text.
- [Appendix B](#) provides the CSM Acegi Sample configuration file.

Additional caCORE Documentation

- The caCORE CSM 4.0 Release Notes contain a description of enhancements and bug fixes included in this release.
- The caCORE CSM 4.0 JavaDocs contain the current caCORE CSM API specification.
- The caCORE SDK 4.0 Developer's Guide contains detailed instruction on the use of the SDK and how it aids in creating a caCORE-like software system.

Document Text Conventions

The following table (Table 1-1) shows various typefaces to differentiate between regular text and menu commands, keyboard keys, tool bar buttons, dialog box options, and text that you type. This illustrates how text conventions are represented in this manual:

<i>Convention</i>	<i>Description</i>
Notes	Notes: Notes are enclosed for emphasis
Bold	Bold type is used for emphasis, buttons, or tabs to select on windows, and names of dialog boxes.
TEXT IN SMALL CAPS	TEXT IN SMALL CAPS is used for keyboard keys that you press (for example, ALT+F4)

Convention	Description
Text in italics	Italics are used to reference other documents, sections, figures, and tables.
Special typestyle	Special typestyle is used for filenames, directory names, commands, file listings, and anything that would appear in a Java program, such as methods, variables, and classes.
<i>Bold italics typestyle</i>	Bold italics is used for information the user needs to enter
{ }	Curly brackets are used for replaceable items (for example, replace {home directory} with its proper value such as C:\caadapter).

Table 1-1 Document text conventions

Chapter 2 CSM Overview

This chapter provides an overview of the Common Security Module (CSM).

Topics in this chapter include:

- [Introduction](#) on this page
- [Security Concepts](#) on page 6
- [Minimal System Requirements](#) on page 8

Introduction

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

1. **Authentication**- validates and verifies 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 that locks out unauthorized users for a pre-configured amount of time after the number (also pre-configured) of allowed attempts is exceeded.
2. **Authorization** - grants access to data, methods, and objects. CSM incorporates an Authorization schema and database so that users can only perform operations or access data to which they have access rights.
3. **Instance and Attribute Level Security** - allows users to perform instance level filtering of data. The User Provision Tool (UPT) allows administrators to provision security filters for instances of domain classes and the API filters the results of the queries based on access policy. Filtering data is done at the database level with minimum overheads. It also performs attribute level filtering of data based on user permissions.
4. **User Provisioning** - creates or modifies 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 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
5. **Audit Logging** - In an effort to make CSM compliant with CRF 21/ part 11, auditing and logging functionality are provided. CSM uses NCICB's Common Logging Module (CLM), which is another caCORE product, for the purpose of event logging as well as automated object state change logging into a persistent database.

The CSM works with the Java Authentication and Authorization Service (JAAS) to authenticate and authorize, as illustrated in the example in Figure 2-1 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, which contains the Users, Roles, Protection Elements, etc., and their associations. This allows the application to know whether to allow a user to access a particular object. The Authorization data can be stored on a variety of databases and is created and modified by the Application Administrator using the web-based UPT.

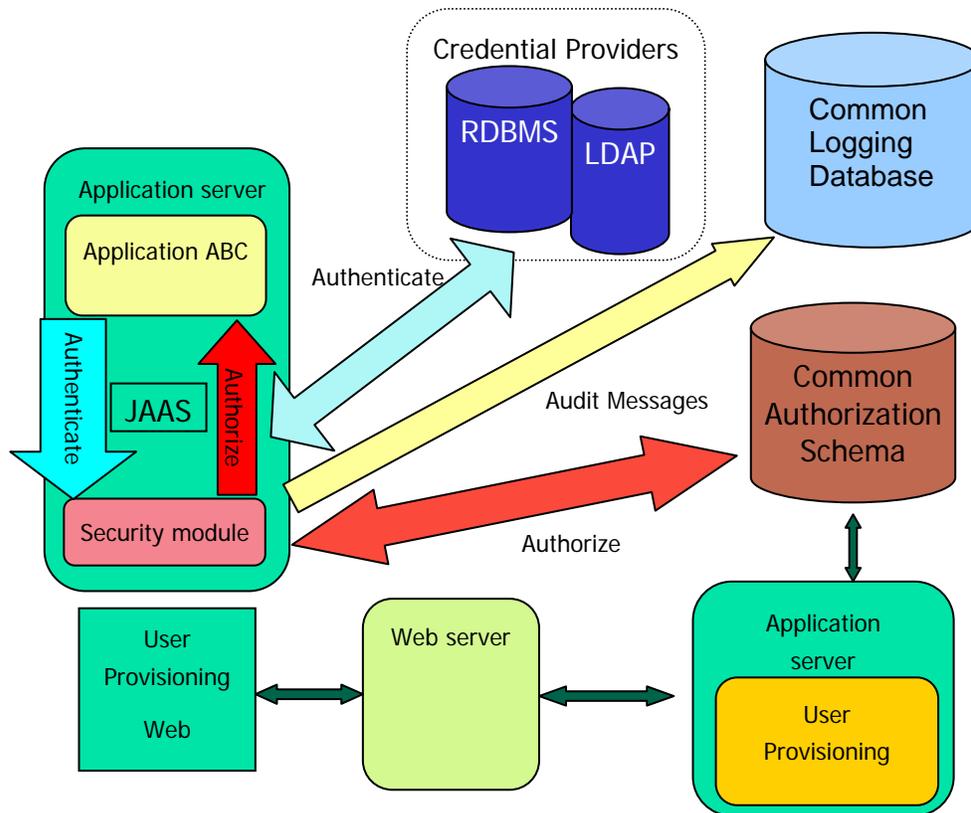


Figure 2-1 CSM Architecture

CSM uses NCICB's Common Logging Module (CLM) to perform all audit and logging functions. CSM logs all of the events and object state changes (security objects are listed below in Table 2-1). These logs are 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. More details regarding audit logging by CSM can be found in [Audit Logging](#) on page 26.

Security Concepts

In order to successfully integrate CSM with an application, it is important to understand the definitions of 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	Someone that requires access to an application. Users can become part of a Group, and can have associated Protection Group and Roles.
Group	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	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 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	Refers to any operation performed upon data. CSM makes use of a standard set of privileges, which aids in standardizing authorization to comply with JAAS and Authorization Policy and allows for adopting technology such as SAML in the future.
Role	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. Table 2-2 shows definitions of related security terms.

Related Concept	Definition
Credential Provider	A credential is a data or set of data that 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.
JAAS	A set of Java packages that enable services to authenticate and enforce access controls upon users. JAAS implements a Java version of the standard Pluggable Authentication Module framework, and supports user- based authorization.
LDAP	Credential providers may choose to store credential information using a directory based on LDAP. An LDAP is simply a set of

Related Concept	Definition
	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 LDAP, credential data is stored in the form of related tables.
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).

Table 2-2 Related security concept definitions

Minimal System Requirements

The software listed in Table 2-3 is required and not included with CSM software. The software name, version, description, and URL hyperlinks are indicated in the table.

Software	Description	Version	URL
JDK	The J2SE Software Development Kit (SDK) supports creating J2SE applications	1.5.0_11 or higher	http://java.sun.com/j2se/1.5.0/download.html
Oracle	Database Server†	9i	http://www.oracle.com/technology/products/oracle9i/index.html
MySQL		5.0.27	http://dev.mysql.com/downloads/mysql/5.0.html
JBoss	Application Server†	4.0.5	http://labs.jboss.com/jbossas/downloads
Tomcat		5.5.20	http://tomcat.apache.org/download-55.cgi
Ant	Build Tool	1.6.5 or higher	http://ant.apache.org/bin/download.cgi

Table 2-3 Minimal software requirements

† Only one is required.

Chapter 3 Using the CSM API

This chapter provides an overview, workflow, and specific deployment and integration steps for the CSM API.

Topics in this chapter include:

- [Workflow](#) on this page
- [API Services](#) on page 10
- [Authentication](#) on page 10
- [Integrating with the CSS Authentication Service](#) on page 10
- [Installation and Deployment Configurations](#) on page 12
- [Authorization](#) on page 20
- [Installation and Deployment Configurations](#) on page 23
- [User Provisioning Tool](#) on page 25
- [Audit Logging](#) on page 26

Workflow

This 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 this chapter in its entirety. It provides an overview, workflow, and specific deployment and integration steps. If using the provisioning service, also read [Chapter 4, Using the User Provisioning Tool](#) starting on page 31.
3. Appoint a Security Schema Administrator who is familiar with the application and its user base. With the User Provisioning Tool (UPT), these individuals input users, 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 information.
5. Decide upon a deployment approach. As discussed in [Installation Modes](#) on page 59, 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 described in detail in this chapter.
7. Decide if you want to enable Audit Logging for your services. If yes then configure [Audit Logging](#) as explained in this chapter.

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.

API Services

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

- AuthenticationManager – for Authentication
- AuthorizationManager – for Authorization and User Provisioning.

AuthenticationManager

The AuthenticationManager is an interface that authenticates a user against a credential provider. See [Integrating with the CSS Authentication Service](#) on page 10 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 available in the CSM API Javadocs.

AuthorizationManager

The AuthorizationManager is an interface that provides run-time methods with the purpose of checking access permissions. See [Integrating with the CSM Authorization Service](#) on page 20 to learn how to integrate with the AuthorizationManager. This manager also provides an interface where application developers can provision user access rights. The user provisioning functionality is primarily used internally by the User Provisioning Tool (UPT); hence, 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](#).

Authentication

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 to their applications. Authentication service allows authentication using LDAP and RDBMS credential providers.

Integrating with the CSS Authentication Service

Importing the CSM Authentication Manager Class

To use the CSM Authentication Service, add the highlighted import statements (last two) as shown in Figure 3-1 to the action classes that require authentication.

```

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

```

Figure 3-1 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 `abccapp` then uses the `AuthenticationManager` to perform the actual authentication using the CSM.

Using the CSM Authentication Manager Class

Figure 3-2 illustrates an example of how to use the CSM `AuthenticationManager Service` 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("abccapp");

    boolean loginOK =
    authenticationManager.login(credentials.getUsername(),
    credentials.getPassword());

    if (loginOK)System.out.println("SUCESSFUL LOGIN");

    else System.out.println("ERROR IN LOGIN");

}catch (CSEException cse){
    System.out.println("ERROR IN LOGIN");
}

```

Figure 3-2 Example code to use the CSM `AuthenticationManager Service` 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 `CSEException` is thrown with the appropriate error message embedded.

Installation and Deployment Configurations

This section serves as a guide to help developers integrate applications with CSM’s Authentication Service. It outlines a step by step process that addresses what developers need to know in order to successfully integrate CSM’s Authentication, which includes:

- CSM API jar placement
- Database properties and configuration
- LDAP properties and configuration
- If audit logging, CLM API jar placement and configuration.

The CSM Authentication Service is available for any application and it can be used exclusively and is effective on its own. CSM’s Authentication Service does not need to replace existing authentication in an application. It can be used to supplement an application’s current authentication mechanism. Currently, only RDBMS-based and LDAP-based authenticated is supported.

JAR Placement

The CSM API’s Application is available as a JAR file, `csmapi.jar`, which needs to be placed in the class path of the application. Along with this JAR, there are many supporting JARs on which the CSM API depends. In case of web applications, these should be added in the folder `<application-web-root>\WEB-INF\lib`.

Configuring Lockout in Authentication Manager

If desired the application developers can use the optional user lockout feature provided by CSM’s default JAAS implementation of Authentication Manager. Three properties are available to configure the lockout feature and its use. For the client application to use the lockout manager all the 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 property specifies the time in milliseconds that the user will be locked out after the configured number of unsuccessful login attempts has been navigate toed.
- **allowed-login-time:** This property 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 property specifies the number of unsuccessful login attempts allowed before the user account is locked out.

The default values for the lockout parameters are as given below.

- lockout-time = 1800000 milliseconds
- allowed-login-time = 60000 milliseconds

- allowed-attempts = 3

Alternatively, the user, in the client application class, can call and provide values for the lockout parameters by using the following method of SecurityServiceProvider Class.

```
public static AuthenticationManager
getAuthenticationManager(String applicationContextName, String
lockoutTime, String allowedLoginTime, String allowedAttempts)
...

```

RDBMS Credential Provider Properties and Login Module

Configuration

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 that is fired against the RDBMS tables to verify the user id and the password passed for authentication

The next section shows how to configure using JAAS or the JBoss `login-config.xml` file.

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 3-3 .

```

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

```

Figure 3-3 Configuring a login module

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

```

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

```

Figure 3-4 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 necessary for overall authentication to be successful.
- The ModuleOptions are a set of parameters that 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:@oracle_db_server.nci.nih.gov:1521:abcappdb"

    user="USERNAME"

```

```
passwd="PASSWORD"
```

```
query="SELECT * FROM users WHERE username=? and password=?"
```

As shown in Figure 3-4, since 'abcapp' application has only one credential provider, only one corresponding entry was made in the configuration file. If the application uses multiple credential providers, then the LoginModule's can be stacked. A single configuration file can contain entries for multiple applications.

Configuring a 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 that is located at `{jboss-home}\server\{server-name}\conf\login-config.xml`.

Shown in Figure 3-5 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:@oracle_db_server:1521:abcappdb</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>
      <module-option name="encryption-enabled">YES</module-option>
    </login-module>
  </authentication>
</application-policy>
```

Figure 3-5 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 that 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 that 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>
```

Enabling Encryption in the RDBMS Login Module

Since CSM v3.2, the RDBMS Login Module is now enhanced to support encrypted passwords. CSM 4.0 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 RDBMS Login Module to use encryption as shown Figure 5.5 in the JBoss login-config.xml entry where

```
<module-option name="encryption-enabled">YES</module-option>
```

`Encryption-enabled` 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.

LDAP Credential Provider Properties and Login Module Configuration

The CSM 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:

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

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

- **IdapAdminUserName** – 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.
- **IdapAdminPassword** – Password for the LDAP Admin User mentioned above.

Configuring LDAP Login Module in JAAS

For `abcapp`, which uses `LDAPLoginModule`, the JAAS config file entry is shown in Figure 3-6.

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

Figure 3-6 Example JAAS configuration file entry

As shown in Figure 3-6:

- 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 necessary 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"`

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.

Configuring 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 that is located at `{jboss-home}\server\{server-name}\conf\login-config.xml`.

Shown in Figure 5.7 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 5.7 Example LDAP JBoss configuration file

As shown in Figure 5.7:

- 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 that 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 that 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>
```

Configuring LDAP Login Module using Anonymous Bind

If an application uses an LDAP Server that does not 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 3-7 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 3-7 Example LDAP JBoss configuration file for LDAP Servers requiring Binding

As shown in Figure 3-7:

- 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 that 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 that 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>

```

Activating CLM Audit Logging

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 described in [Audit Logging](#) on page 26.

Authorization

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.

Overview to Integrating CSM APIs

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

Integrating with the CSM Authorization Service

Importing and Using the CSM Authorization Manager Class

To use the CSM Service, add the highlighted import statements (last two) as shown in Figure 3-8 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 3-8 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 3-9 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 3-9 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 that it is trying to access, and the operation that 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. If an authorization error occurs, a `CSException` is thrown with the appropriate error message embedded.

Software Products and Scripts

Table 3-1 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 and is 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 that 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 3-1 Authorization software products

File	Description
hibernate.cfg.xml	The sample XML file that contains the hibernate-mapping and the database connection details.
AuthSchemaMySQL.sql OR AuthSchemaOracle.sql OR AuthSchemaPostgres.sql	This Structured Query Language (SQL) script is used to create an instance of the Authorization database schema that 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.
DataPrimingMySQL.sql OR DataPrimingOracle.sql OR DataPrimingPostgres.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

<i>File</i>	<i>Description</i>
mysql-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.
OR	
oracle-ds.xml	
OR	
Postgres-ds.xml	

Table 3-2 Authorization configuration and SQL files

Installation and Deployment Configurations

This section serves as a guide to help developers integrate applications with CSM's Authorization Service. It outlines a step by step process that addresses what developers need to know in order to successfully integrate CSM's Authorization, which includes:

- CSM API jar placement
- Database properties and configuration
- If audit logging, CLM API jar placement and configuration.

Jar Placement

The CSM Application is available as a JAR that 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`.

Database Properties and Configuration

Create and Prime 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 must be the same.

1. Log into the database using an account id that has permission to create new databases. Since the CSM caCORE 3.0.1 release, you can 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 steps 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 example, "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. 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. 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: The default password is always “changeme” and this should be 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.

Configure Datasource

1. Modify the provided `mysql-ds.xml` or `oracle-ds.xml` file that 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.

Figure 3-10 is an example of the `mysql-ds.xml` file.

```

<datasources>
  <local-tx-datasource>
    <jndi-name>csmupt</jndi-name>
    <connection-url>jdbc:mysql://mysql_db: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://mysql_db: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 3-10 Example *mysql-ds.xml* file

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

Activate CLM Logging

In order to activate the CLM's Audit Logging capabilities for Authorization, follow the steps to deploy Audit Logging service as described in [Audit Logging](#) on page 26.

User Provisioning Tool

The CSM User Provisioning Tool 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.

See [Chapter 4, Using the User Provisioning Tool](#) starting on page 31 for details on usage of the UPT. This chapter also explains how to deploy the UPT from start to finish – from uploading the Web Application Archive (WAR) and editing configuration files to syncing the UPT with the application.

Audit Logging

Introduction

In an effort to make CSM compliant with CRF 21/ part 11, CSM provides auditing and logging functionality. Currently CSM is using log4j for logging application logs. However, CRF21/ part 11 requires that certain messages be logged in a specific way. For example, all objects should be logged in a manner that allows them to be audited at a later stage. There are two types of audit logging: Event logging and Object state logging. Audit logging capability are 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 is able to utilize the inbuilt audit logging functionality. The logging results are saved into a database or a flat text file depending on the configuration. In addition, the logging can be enabled and disabled for any fully qualified class name.

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

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

Jar Placement

The Audit Logging Application is available as a JAR and is 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.

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 the CSM.

Event Logging

Both the Authentication and Authorization services 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 as shown in JDBC Appender section below.

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 as shown in JDBC Appender section below.

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 shown in Figure 3-11.

```
<?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" />
</category>

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

</log4j:configuration>
```

Figure 3-11 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 CLM features without using CSM, 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. In addition, the log locator tool can be used for the purpose of viewing the logs.

Deployment Steps

In order for a client application to enable 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 must be created that will persist the audit logs that are generated as a basis of usage of the CSM APIs
2. Refer to the [CLM 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 3-11).
2. Replace the <<SERVER_NAME>>, <<PORT>> and the <<CLM_SCHEMA_NAME>> with 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 and Authorization or object state audit logging for the Authorization; the corresponding logger needs to be configured.

Note: The names of loggers must not differ from the sample.

5. In the case of the 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.

The configuration steps for setting up the browser are mentioned in the [CLM Guide](#) for application developers.

Chapter 4 Using the User Provisioning Tool

This chapter provides an overview of the User Provisioning Tool (UPT), outlines a suggested workflow, and explains how to perform all UPT operations.

Topics in this chapter include:

- [Introduction](#) on this page
- [Workflow](#) on page 31
- [Common Basic Functions](#) on page 32
- [Assignments and Associations](#) on page 36
- [Super Admin Mode](#) on page 39
- [Admin Mode](#) on page 44
- [UPT Installation and Deployment](#) on page 58

Introduction

The User Provisioning Tool (UPT) provides a graphical user interface 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 intended audience of this chapter is all users of the UPT, including Super Administrators who may add applications and associated administrators, and Administrators who will perform provisioning for a particular application.

Workflow

The UPT includes two modes – Super Admin and Admin. The Super Admin operations are typically performed first, as they register the application and application administrators. The primary mode operations, including authorization user provisioning, occur next.

Super Admin

When first deploying the UPT for a particular application, the developer registers the application in the Super Admin mode. Once the application is registered, the Super Admin can add users who will serve as application administrators. The Super Admin can also register additional applications as they become available. This document details these steps in [Super Admin Workflow](#) on page 39.

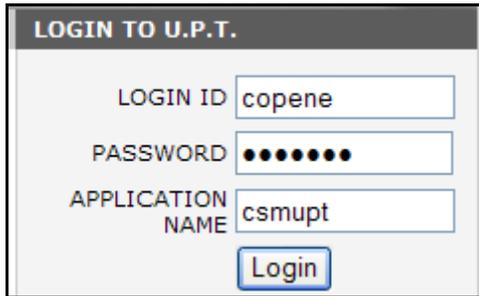
Admin

The primary (Admin) mode is for performing user provisioning for a particular application. The Admin mode follows a simple workflow of creating elements, assigning them, and then associating them. This document details these steps in [Workflow](#) on page 46.

Login

The Login page includes summary text, **What's New**, **Did You Know**, and most importantly the Login section itself: **Login ID**, **Password**, and **Application Name**. For a majority of UPT implementations, the NCICB LDAP serves as the authentication mechanism. Therefore the user's Login ID will be the same as the user's NCICB user name (in Figure 4-1 and Figure 4-2, user Eric Copen's NCICB user name is **copene**). Similarly, the Password equals the NCICB password. The rules from the authentication system are applied to the user name and password.

If logging on as Super Admin, enter the Application Name **csmupt** (Figure 4-1). If logging in as an Admin, enter the appropriate application name. For example, **Security** is used in Figure 4-2.



LOGIN TO U.P.T.

LOGIN ID

PASSWORD

APPLICATION NAME

Figure 4-1 Login as a Super Admin



LOGIN TO U.P.T.

LOGIN ID

PASSWORD

APPLICATION NAME

Figure 4-2 Login as an Admin

Since UPT uses CSM's Authentication Manager, it can be configured to lock a user out if they try to make an unauthorized entry into the UPT. If configured appropriately, UPT can lock the user out after a pre-configured number of unsuccessful attempts have been navigated in the allowed login time frame. Once locked out, the user can log in only after the configured amount of lockout time has elapsed. This provides security from hacking attempts to break into the UPT.

Common Basic Functions

Within the UPT, there are several common operations that are repeated for most elements. These operations include **Create New**, **Search** and **Update**, **Delete**, and **Assign/Associate**. This section describes how these operations are performed..

Create New Element

When creating a new element follow the steps outlined below. The same basic steps can be followed to create any element; in this example, a User is created.

Step 1: On the element Home page select **Create a New...**(Figure 4-3)



Figure 4-3 New and Existing User options

Step 2: Enter details (Figure 4-4):

* indicates a required field

ENTER THE NEW USER DETAILS	
* User Login Name	<input type="text" value="smithj"/>
User First Name	<input type="text" value="John"/>
User Last Name	<input type="text" value="Smith"/>
User Organization	<input type="text" value="NIH"/>

Figure 4-4 Entering new user details

Step 3: Select **Add** to save the new element (in this case User) to the database. This save occurs immediately. **Back** acts exactly like the back button in a browser – returning the user to the home page. **Reset** clears the data from the entire form. Remember that no data is saved until the **Add** button is selected.

Step 4: Upon a successful save, the system displays **Add Successful** just below the menu and before the text. In addition, a new set of buttons appears below the details table in Admin mode (Figure 4-5).



Figure 4-5 A new set of buttons appear below the menu after you have successfully added a new user

Note: The additional set of buttons is visible in Admin mode only. The Super Admin mode shows limited buttons.

Example Error Messages

The User Interface performs basic data validation, including field lengths and formats. Figure 4-6 is an example of a message displayed when a user enters an improperly formatted email address:



Figure 4-6 Error message after entering incorrect email address

The system displays the message in Figure 4-7 (or similar) if a user tries to add an entry (for example, *smithj*) when it already exists in the system:



Figure 4-7 Error message after entering a user already in the system

Search For and Select Existing Elements

When searching for and selecting an element, follow the steps below. The same basic steps can be followed for any element; in this example, a Role is searched for and selected.

Step 1: On the element Home page select **Select an Existing...**(Figure 4-8).

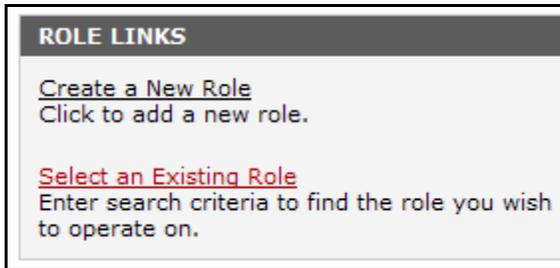


Figure 4-8 Selecting an existing Role

Step 2: Enter search criteria (Figure 4-9). Use the * character to perform wildcard searches. For example, searching for Role* returns Role_name_1, Role_name_2, or any other role beginning with role. A search of *1 returns anything ending with 1 – Role_name_1, Role_name_101, Role_name_51, etc. **Select Search** for results. **Back** returns the user to the home page. **Reset** clears the data.



Figure 4-9 Entering search criteria for Role

Step 3: The system returns a list of matching roles. The results are sorted alphabetically for all search result screens. (Figure 4-10):

SEARCH RESULTS		
Select	Role Name	Role Description
<input type="radio"/>	Role_name_1	Role_Desc_1
<input type="radio"/>	Role_name_101	Role_Desc_10
<input type="radio"/>	Role_name_51	Role_Desc_5

Figure 4-10 Role search results

Step 4: Select the desired element, in this case **Role_name_1**, by clicking on the radio button in the **Select** column (Figure 4-11). You can select one element at a time to view.

<input checked="" type="radio"/>	Role_name_1	Role_Desc_1
----------------------------------	-------------	-------------

Figure 4-11 Example of selecting an element with a radio button

Step 5: Click on the **View Details** button below the Search Results table. The system then displays this element's details.

Example Error Messages

If the search criteria results in no matches, the system displays an error indicating there are no matches in a search. Modify the search criteria and repeat until the intended results appear.

Update an Element

When updating an element follow these steps. The same basic steps can be followed for any element; in this example, a Protection Element is updated.

Step 1: Navigate to the details screen in one of two ways – either create a new element (see [Create New](#) on page 32) or search for and select an existing element (see [Search For and Select Existing Elements](#) on page 34). The details screen (Figure 4-12, Figure 4-13, and Figure 4-13) displays information such as name and description.

PROTECTION ELEMENT DETAILS	
* Protection Element Name	Test PE Name1103749550261
Protection Element Description	Test Desc

Figure 4-12 Protection element details

Step 2: Simply replace existing text, and select **Update**.

PROTECTION ELEMENT DETAILS	
* Protection Element Name	Test PE Name1103749550261
Protection Element Description	This is my new text I want to update.

Figure 4-13 Entering text for a Protection Element

Step 3: Upon a successful update, the system displays **Update Successful** just below the menu and before the text.

Example Error Messages

The User Interface performs basic data validation, including field lengths and formats. The systems also check for duplicates; it prevents changing the element name to one that already exists.

Delete an Element

When deleting an element, follow these steps. The same basic steps can be followed for any element; in this example, a Group is deleted.

Step 1: Navigate to the Group Details screen. From the home page, either create a new Group (see [Create New](#) on page 32) or search for and select an existing element (see [Search For and Select Existing Elements](#) on page 34).

Step 2: Click on the **Delete** button.

Step 3: A prompt displays with the text **Are you sure you want to delete the record?**. Click **OK** to confirm. Clicking **Cancel** negates the operation and returns the display to the Details screen.

Step 4: Upon confirming the deletion, the system returns you to the Group home page and displays in blue text the words, **Delete Successful**.

Assignments and Associations

The elements Role, Protection Group, and Group are simply collections of other elements – Privileges, Protection Elements, and Users respectively. Provisioning includes assigning elements to elements or removing elements from an element (called *deassigning*). For example, assigning Users to Groups greatly improves the ease by which one can provision access rights. An Admin can instantly assign a role and protection group to an entire group of people instead of repeating the same assignment for each individual.

Assign or Deassign Privileges, Roles, ProtectionGroups, Groups

Step 1: Navigate to the Association screen. From the element home page, either create a new element (see [Create New](#) on page 32) or search for and select an existing element (see [Search For and Select Existing Elements](#) on page 34). The element's Details screen displays a button containing the text **Associated**, **Assign**, or something similar depending on the element type.

Step 2: With this UI implementation, associations can be established or removed by simply selecting elements and moving them from one box to another. The box on the top lists the Available Groups (unassigned) and the box below lists the Groups assigned to the User – Group_Northeast, Group_ProjectLead, and Group_Research_A (Figure 4-14). Highlight a Group and select **Assign** to move it to the Assigned Groups box. Select **Deassign** to move it back to the Available Groups box.

There are multiple ways to highlight the elements within the box:

1. Select one by clicking on the user name entry.
2. Select multiple users' entries by holding down control while selecting and/or deselecting.
3. Select multiple by holding down the SHIFT key while selecting the first and then last of a collection.

Assign or Deassign multiple **Groups** for the selected **User**. To remove the complete association Deassign all the **Groups**.

AVAILABLE GROUPS
Group_Operations
Group_Patient
Group_Research_B
Group_Sales
Group_Southeast
Group_Tampa

ASSIGNED GROUPS
Group_Northeast
Group_ProjectLead
Group_Research_A

Figure 4-14 Available and Assigned Groups lists

Step 3: Save the association by clicking **Update Association**. No association is saved until this button is selected.

Assign or Deassign Users and Protection Elements

Assign or Deassign Users and ProtectionElements:

Step 1: Navigate to the Association screen. From the element home page, either create a new element (see [Create New](#) on page 32) or search for and select an existing element (see [Search For and Select Existing Elements](#) on page 34). The element's Details screen displays a button containing the text **Associated**, **Assign**, or something similar depending on the element type.

Step 2: With this UI implementation, associations can be established by selecting 'Assign' (Figure 4-15). The box lists the Assigned ProtectionElements. Select **Deassign** to deassign and remove a PE from the Assigned PEs box.

There are multiple ways to highlight the elements within the box:

1. Select one by clicking on the user name entry.
2. Select multiple user's entries by holding down control while selecting and/or deselecting.
3. Select multiple by holding down the shift button while selecting the first and then last of a collection.

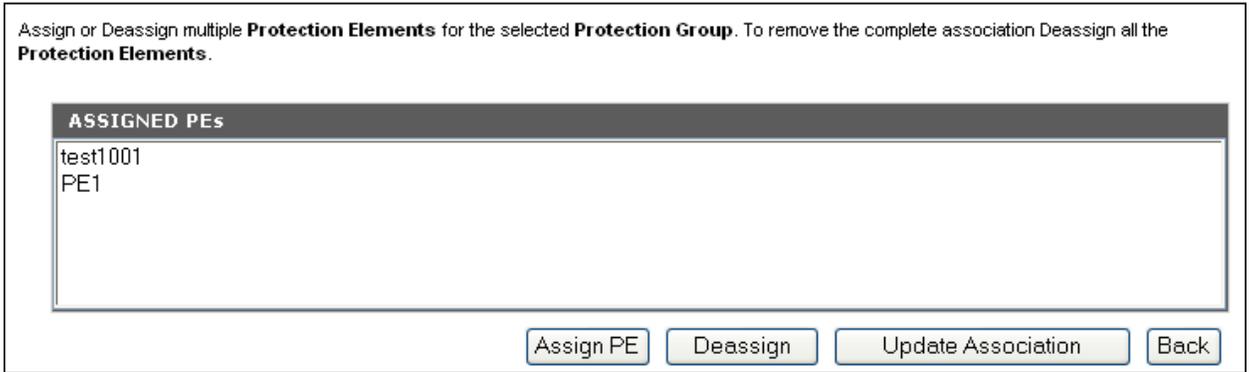


Figure 4-15 Assigned PEs list

Click **Assign PE** to display the Protection Element Search Criteria page (Figure 4-16).

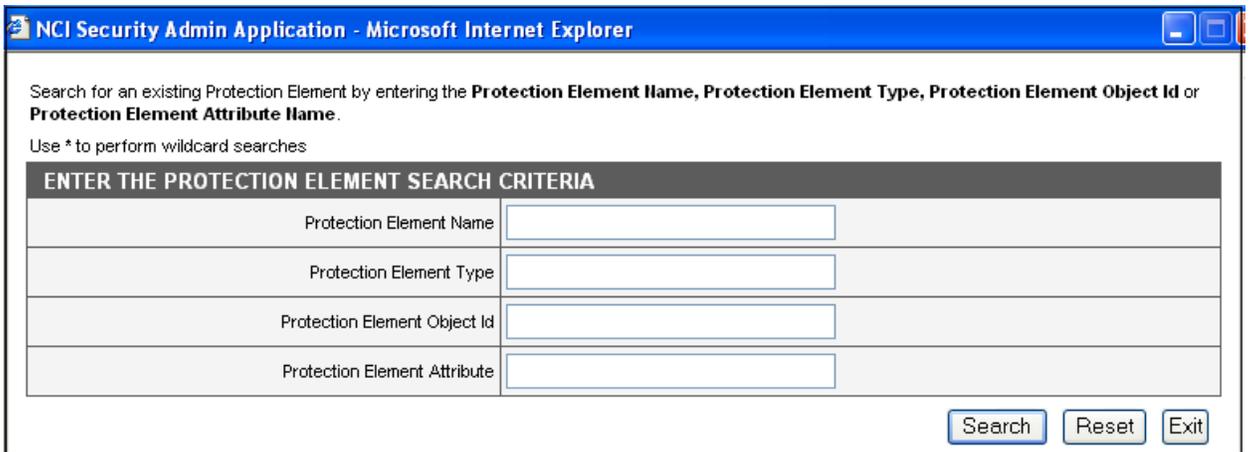


Figure 4-16 Protection Element search criteria page

Click **Search** to display the sorted search results for the given search criteria. One or more checkboxes can be selected for assignment by checking and clicking **Assign PE**. The selected PEs are added to the Assigned PE's box (Figure 4-18).

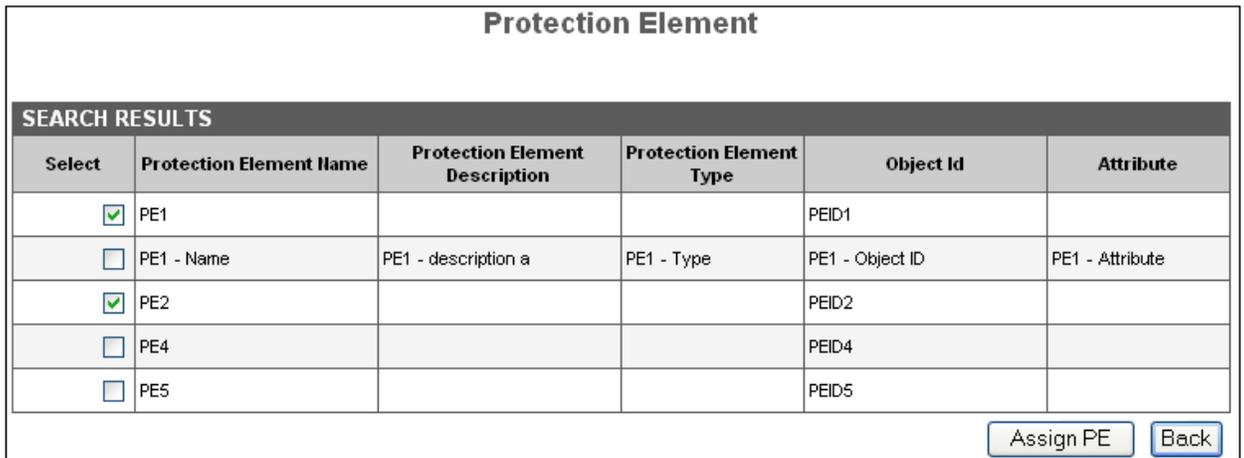


Figure 4-17 Protection Element search result page

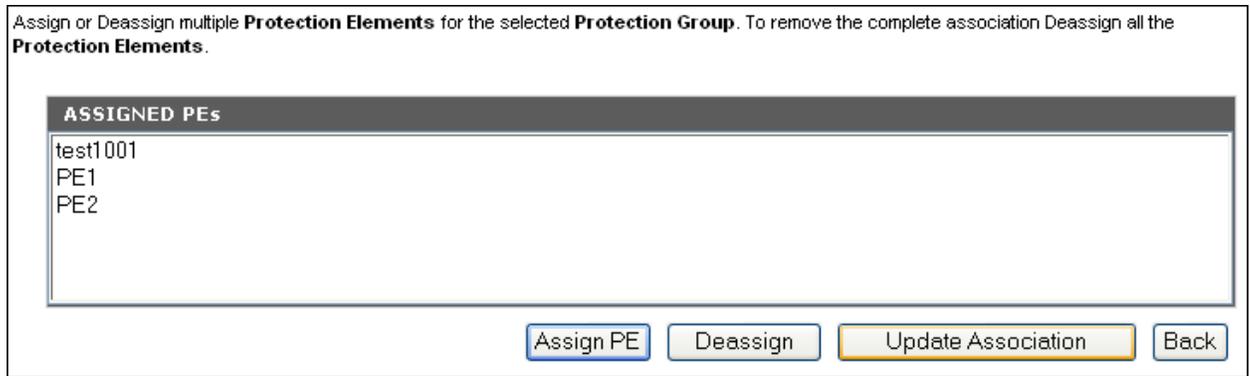


Figure 4-18 Assigned PEs list

Step 3: Save the association by clicking **Update Association**. No association is saved until this button is selected.

Super Admin Mode

Overview

The Super Admin Mode includes operations pertaining to Users (Application Administrators), Applications, and Privileges. Super Admins may add, remove, or modify Application details. They may also assign users to these Applications, modify user details, and remove users. Lastly, they may modify existing CSM Standard Privileges or create new application-specific privileges.

Workflow

The CSM team designed the UPT as a flexible tool with a flexible workflow. Any operation can be completed quickly. However, at first it may be difficult to know where to start. The following is a suggested workflow for getting started in the Super Admin Mode:

1. **Application** – when first deploying the UPT for a particular application, the developer registers the application.
2. **Application** – add and update Application details.
3. **User** – add and update users who will serve as Application Administrators.
4. **Application** – assign users to applications.
5. **Privilege** – if necessary, add or edit CSM Standard Privileges.

Navigation

Use the gray menu to navigate through the Super Admin section. From the Home page, the menu looks like this:



Figure 4-19 Home Page menu options

The menu option with a blue background designates the current location. Roll over the other choices until they turn blue, and then click to navigate to that section. The **Log Out** selection returns the user to the Login page.

Application

In the Application section, a Super Admin can add an application to the UPT and add or modify details. The following available operations can be performed:

Create a New Application

- a. Go the Application home page.
- b. Select **Create a New Application**.
- c. Enter data into the Application Details form.
 1. **Application Name** – uniquely identifies the Application, required field.
 2. **Application Description** – a brief summary describing the Application.
 3. **Declarative Flag** – indicates whether application uses Declarative security.
 4. **Application Active Flag** – indicates if the Application is currently active.
 5. **Database URL** – The JDBC Database URL for the given application.
 6. **Database User Name** – The Username for the application database.
 7. **Database Password** – The Password for the application database
 8. **Database Dialect** – The Dialect for the application database.
 9. **Database Driver** –The Driver for the application database
- Note:** The Database fields should either be completed together or left blank completely. They are all required fields if at least one of them is populated.
- d. Select **Add** button.

Select an Existing Application and Update

- e. Go to the Application home page.
- f. Click on **Select an Existing Application**.
- g. Enter data into the Application Search Criteria form.
 1. **Application Name** – uniquely identifies the Application.
- h. Click on the radio button corresponding with the intended Application name.
- i. Select **View Details**.
- j. Enter data into the Application Details form.
 1. **Application Name** – uniquely identifies the Application, required field.
 2. **Application Description** – a brief summary describing the Application.
 3. **Declarative Flag** – indicates whether application uses Declarative security.
 4. **Application Active Flag** – indicates if the Application is currently active.
 5. **Database URL** – The JDBC Database URL for the given application.
 6. **Database User Name** – The Username for the application database.
 7. **Database Password** – The Password for the application database
 8. **Database Dialect** – The Dialect for the application database.
 9. **Database Driver** –The Driver for the application database

Note: the Database fields should either be completed together or left blank completely. They are all required fields if at least one of them is populated.

- k. Select **Update** button.

Delete an Existing Application

- l. Navigate to the Application Details form by either creating a new **Application** or **Selecting an Existing Application**.
- m. Select **Delete**.
- n. In the pop-up window, click **Okay** to confirm intent to delete the Application.

Application and Admin Association

- o. Navigate to the Application Details form by either creating a new Application or Selecting an Existing Application.
- p. Select **Associated Admins**.
 1. Associate Users (see [Assignments and Associations](#) on page 36).
- q. Click on the **Assign** and **Deassign** buttons until the proper association is displayed.
- r. Save the association by clicking on **Update Association**. No association is saved until this button is selected.

Managing Users

In this section, Users can be assigned as UPT administrators for their particular application(s). They will have the right to create and modify Roles, Groups, etc. New Users can also be created or existing User details modified. Following are the available operations:

Create a New User

- a. Go to the User home page.
- b. Select **Create a New User**.
- c. Enter data into the User Details form.
 - **Name** – uniquely identifies the User, required field.
 - **First Name** and **Last Name** – attributes that help identify the User.
 - **Organization** – Organization for which the User works. An example is the National Cancer Institute (NCI).
 - **Department** – Department for which the User works. An example is caArray.
 - **Title** – Title for User.
 - **Phone Number** – provides contact information, typically the direct business phone number for the User. The phone number field accepts the following formats: 0123456789, 012-345-6789, (012)3456789, (012)345-6789, (012)-345-6789
 - **Email Id** – provides the email contact details for the User. An email ID must contain an '@' sign.
 - **Password** – an optional field used if the schema for Authorization will also be used for Authentication. The only characters visible within this field are stars '*' so the password is not visible on the screen.
 - **Confirm Password** – a copy of the password field. It ensures the intended password was entered correctly. This field must match the password field exactly.
 - **User Start Date** and **User End Date** – Indicates user start date and end date..
- d. Select **Add** button.

Select an Existing User and Update

- a. Go to the User home page.
- b. Click **Select an Existing User**.
- c. Enter data into the User Search Criteria form.
 - **User Name** – uniquely identifies the User.
- d. Click on the radio button corresponding with the intended User name.
- e. Select **View Details**.
- f. Enter data into the User Details form.
 - **Name** – uniquely identifies the User, required field.
 - **First Name** and **Last Name** – attributes that help identify the User.
 - **Organization** – Organization for which the User works. An example is the National Cancer Institute (NCI).
 - **Department** – Department for which the User works. An example is caArray.
 - **Title** – Title for User.
 - **Phone Number** – provides contact information, typically the direct business phone number for the User. The phone number field accepts the following formats: 0123456789, 012-345-6789, (012)3456789, (012)345-6789, (012)-345-6789
 - **Email Id** – provides the email contact details for the User. An email ID must contain an asterisk.
 - **Password** – an optional field used if the schema for Authorization will also be used for Authentication. The only characters visible within this field are stars '*' so the password is not visible on the screen.
 - **Confirm Password** – a copy of the password field. It ensures the intended password was entered correctly. This field must match the password field exactly.
 - **User Start Date** and **User End Date** – determine the period for which the User is a valid User.
- g. Select **Update** button.

Delete an Existing User

- a. Navigate to the User Details form by either creating a new User or Selecting an Existing User.
- b. Select **Delete**.
- c. In the pop-up window, click **Okay** to confirm intent to delete the User.

Assigning Privileges

A Privilege refers to any operation performed upon data. Assigning Privileges helps control access to important components of an application (Protection Elements).

The UPT installs with CSM Standard Privileges that were agreed upon by the Security Working Group. These privileges include the following:

Standard Privileges

Within CSM, users may possess one or more of the following privileges for a particular

protection element (Table 4-1).

Privilege Name	Privilege Definition	Applying the Privilege (Example)
CREATE	This privilege grants permission to a user to create an entity. This entity can be an object, a database entry, or a resource such as a network connection.	A user can create a database entry.
ACCESS	This privilege allows a user to access a particular resource. Examples of resources include a network connection, database connection, socket, module of the application, or even the application itself.	A user can gain access to a particular module in an application.
READ	This privilege permits the user to read data from a file, URL, socket, database, or an object. This can be used at an entity level signifying that the user is allowed to read data about a particular entry (which can be object or database row, etc.)	A user can view personal information such as a Social Security Number.
WRITE	This privilege allows a user to write data to a file, URL, socket, database, or object. This can also be used at an entity level signifying that the user is allowed to write data about a particular entity (which may include an object, database row, etc.)	A user can add text to a database entry.
UPDATE	This privilege grants permission at an entity level and signifies that the user is allowed to update and modify data for a particular entity. Entities may include an object, an attribute of the object, a database row, etc.	A user can modify an object's attribute data.
DELETE	This privilege permits a user to delete a logical entity. This entity can be an object, a database entry, a resource such as a network connection, etc.	A user can delete record.
EXECUTE	This privilege allows a user to execute a particular resource. The resource can be a method, function, behavior of the application, URL, button etc.	A user can click on a button to perform a method.

Table 4-1 Standard privileges

If necessary, new application-specific Privileges can also be created or existing Privilege details modified. The available operations are described in the following subsections.

Create a New Privilege

- a. Go to the Privilege home page.
- b. Select **Create a New Privilege**.
- c. Enter data into the Privilege Details form.
 - **Name** – uniquely identifies the Privilege, required field.
 - **Description** – a brief summary describing the Privilege.
- d. Select **Add** button.

Select an Existing Privilege and Update details

- a. Go to the Privilege home page.
- b. Click **Select an Existing Privilege**.
- c. Enter data into the Privilege Search Criteria form. Search **Privilege** name.
- d. Click on the radio button corresponding with the intended **Privilege** name.
- e. Select **View Details**.
- f. Enter data into the Privilege Details form.
 - **Name** – uniquely identifies the Privilege, required field.
 - **Description** – a brief summary describing the Privilege.
- g. Select **Update** button.

Delete an Existing Privilege

- a. Navigate to the Privilege Details form by either creating a new Privilege or Selecting an Existing Privilege.
- b. Select **Delete**.
- c. In the pop-up window, click **Okay** to confirm intent to delete.

Admin Mode

Overview

The Admin Mode of the UPT is divided into six major sections: [Groups](#), [Privileges](#), [Protection Groups](#), [Roles](#), and [Users](#). In these sections an Admin can perform basic functions such as modify, delete, or create, and manage associations between the objects. For example, you may assign Privileges to a Role. Figure 4-20 helps to illustrate how all objects (also referred to as elements) are related in the Authorization schema. Table 4-2 follows with definitions of each category of authorization.

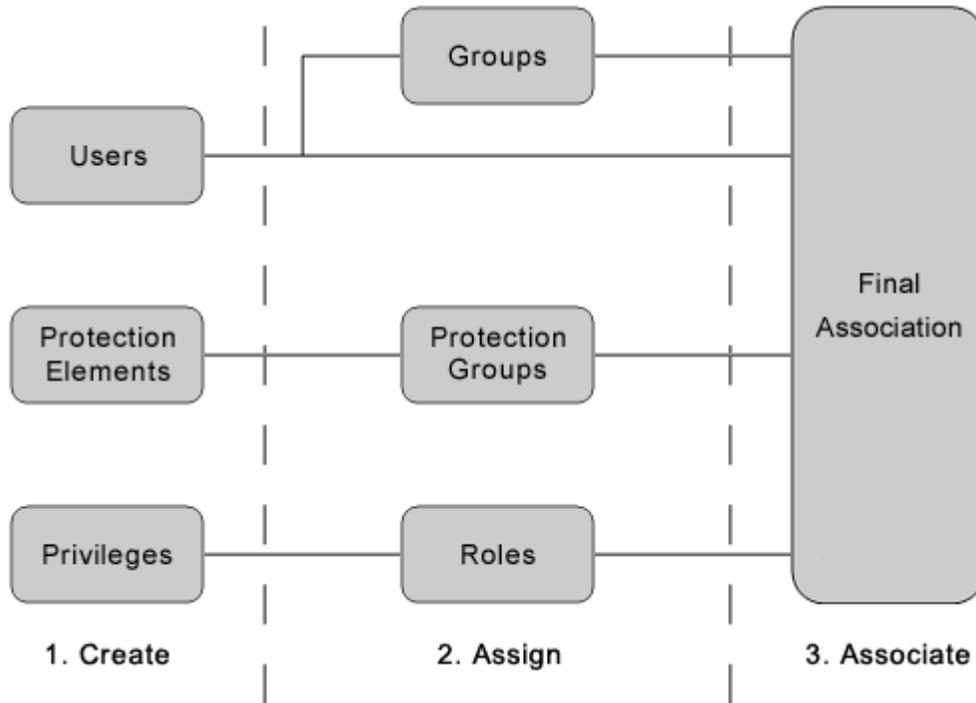


Figure 4-20 Relationships between objects in the Authorization Schema

Definitions for Authorization Status	
User	Someone who requires access to your application. Users can become part of a Group, and can have an associated Protection Group and Roles.
Protection Element	Any entity (typically data) that has controlled access. Examples include Social Security Number, City, and Salary.
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.
Group	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 Group	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.
Role	A collection of application Privileges. Examples include Record Admin and EmployeeModify.

Definitions for Authorization Status	
Final Association	The correlation between a User and his Roles for a particular Protection Group.
Each User (and Group) assumes Roles (rights) for a Protection Group (protected entities). For example, User John has a Role EmployeeModify for all elements in the Address Protection Group. Assign PGs and Roles from the User or Group sections of the UPT.	

Table 4-2 Categories of authorization status

Workflow

The CSM team designed the UPT as a flexible tool with a flexible workflow. Any operation can be completed quickly, however, at first it may be difficult to know where to start. The general concept of the workflow is to create the base elements first and then create the groupings and associations. Here is the suggested workflow for getting started in the Admin Mode:

1. Create base objects – Users and Protection Elements (CSM Standard Privileges are provided).
2. Create collections of these objects (in any order):
 - a. Groups
 - i. Create Groups.
 - ii. Assign Users to Groups.
 - b. Protection Groups
 - i. Create Protection Groups.
 - ii. Assign Protection Elements to Protection Groups.
 - c. Roles
 - i. Create Roles.
 - ii. Assign Privileges to Roles.
3. Associate rights with Users and Groups (in any order).
 - i. Assign a Protection Group and Roles to Users.
 - ii. Assign a Protection Group and Roles to Groups.

Navigation

Use the gray menu to navigate through the Admin section. From the Home page, the menu looks like this:



Figure 4-21 Menu options in the Admin section of the home page

The menu option with a blue background designates the current location. Roll over the other choices until they turn blue, and then click to navigate to that section. The **Log Out** selection returns the user to the Login page.

User

A User is simply someone that requires access to an application. In this section create

new Users, modify existing User details, and associate or disassociate Users with a Protection Group and Roles. The available operations are:

1. Create a New User

- a. Go to the User home page.
- b. Select Create a New User.
- c. Enter data into the User Details form.
 - **Name** – uniquely identifies the User, required field.
 - **First Name** and **Last Name** – attributes that help identify the User.
 - **Organization** – Organization for which the User works. An example is the National Cancer Institute (NCI).
 - **Department** – Department for which the User works. An example is caArray.
 - **Title** – Title for User.
 - **Phone Number** – provides contact information, typically the direct business phone number for the User. The phone number field accepts the following formats: 0123456789, 012-345-6789, (012)3456789, (012)345-6789, (012)-345-6789
 - **Email Id** – provides the email contact details for the User. An email ID must contain an asterisk.
 - **Password** – an optional field used if the schema for Authorization will also be used for Authentication. The only characters visible within this field are stars '*' so the password is not visible on the screen.
 - **Confirm Password** – a copy of the password field. It ensures the intended password was entered correctly. This field must match the password field exactly.
 - **User Start Date** and **User End Date** – determine the period for which the User is a valid User.
- d. Select **Add** button.

2. Select an Existing User and Update details

- a. Go to the User home page.
- b. Click on **Select an Existing User**.
- c. Enter data into the User Search Criteria form. Search by any combination of the below:
 - **Name** – uniquely identifies the User, required field.
 - **First Name** and **Last Name** – attributes that help identify the User.
 - **Organization** – Organization for which the User works. An example is the National Cancer Institute (NCI).
 - **Department** – Department for which the User works. An example is caArray.
 - **Email Id** – provides the email contact details for the User. An email ID must contain an asterisk.
- d. Click on the radio button corresponding with the intended **User name**.
- e. Select **View Details**.
- f. Enter data into the User Details form.
 - **Name** – uniquely identifies the User, required field.
 - **First Name** and **Last Name** – attributes that help identify the User.
 - **Organization** – Organization for which the User works. An example is the National Cancer Institute (NCI).
 - **Department** – Department for which the User works. An example is caArray.
 - **Title** – Title for User.
 - **Phone Number** – provides contact information, typically the direct business

phone number for the User. The phone number field accepts the following formats: 0123456789, 012-345-6789, (012)3456789, (012)345-6789, (012)-345-6789

- **Email Id** – provides the email contact details for the User. An email ID must contain an asterisk.
 - **Password**– an optional field used if the schema for Authorization will also be used for Authentication. The only characters visible within this field are stars ‘*’ so the password is not visible on the screen.
 - **Confirm Password** – a copy of the password field. It ensures the intended password was entered correctly. This field must match the password field exactly.
 - **User Start Date** and **User End Date** – determine the period for which the User is a valid User.
- g. Select **Update** button.

The User Details page contains the four buttons displayed in Figure 4-22. The numbers above these buttons correspond to the operations that follow:



Figure 4-22 User Details Page button options

3. Assign a User to a Group or Groups ③

- a. Navigate to the User Details form by either creating a new User or Selecting an Existing User.
- b. Select **Associated Groups**.
- c. Determine which of the available Groups to which the User should be assigned. Select these Groups by highlighting them (see [Assignments and Associations](#) on page 36 for details).
- d. Click on the **Assign** and **Deassign** buttons until the proper association is displayed.
- e. Save the association by clicking on **Update Association**.

Note: No association is saved until this button is selected.

4. View User Report ④

This feature is new to the 3.0.1 release in response to a requirement formed by the caCORE team. This reporting functionality shows a user’s privileges for all of his protection elements.

- a. Navigate to the User Details form by either creating a new User or Selecting an Existing User.
- b. Select **Associated PE & Privileges**.
- c. View user’s privileges for each protection element.

5. Update Roles associated with the assigned Protection Groups ⑤

- a. Navigate to the User Details form by either creating a new User or Selecting an

- Existing User.
- b. Select **Associated PG & Roles**. The system displays a list of all associated Protection Groups and their Roles.
 - c. Select the radio button that corresponds with the intended Protection Group.
 - d. Determine which Roles you would like to assign to the User.
 - e. Select the Role by highlighting the name (see [Assignments and Associations](#) for details).
 - f. Click on the **Assign** and **Deassign** buttons until the proper association is displayed.
 - g. Save the association by clicking on **Update Association**.

Note: No association is saved until this button is selected.

6. Assign a Protection Group and Roles to a User ⑥

- a. Navigate to the User Details form by either creating a new User or Selecting an Existing User.
- b. Select **Assign PG & Roles**.
- c. Determine which Protection Group and Roles you would like to assign to the User.
 1. Select the Protection Group by highlighting the name (see [Assignments and Associations](#) for details).
 2. Select the Roles by highlighting them.
- d. Click on the **Assign** and **Deassign** buttons until the proper association is displayed.
- e. Save the association by clicking on **Update Association**.

Note: No association is saved until this button is selected.

7. Delete an Existing User

- a. Navigate to the User Details form by either creating a new User or Selecting an Existing User.
- b. Select **Delete**.
- c. In the pop-up window, click **Okay** to confirm intent to delete.

Protection Element

A Protection Element is any entity (typically data) that is subject to controlled access. CSM allows for a broad definition of Protection Element. Nearly everything in an application can be protected – data, table, buttons, menu items, etc. By identifying individual Protection Elements, it becomes easier to control access to important data. In this section you may create new Protection Elements or modify existing Protection Element details. Here are the available operations:

1. Create a New Protection Element

- a. Go to the Protection Element home page.
- b. Select **Create a New Protection Element**.
- c. Enter data into the Protection Element Details form.
 - **Name** – uniquely identifies the Protection Element, required field.
 - **Object Id** – a string that the Application team assigns to the Protection Element
 - **Attribute Name** – helps to further identify the Protection Element

- **Description** – a brief summary describing the Protection Element.
 - **Update Date** – indicates the date when the Protection Element's Details were last updated
 - **Type** – a string that the application team can assign to indicate type of protection element.
- d. Select **Add** button.
- 2. Select an Existing Protection Element and Update details**
- a. Go to the Protection Element home page.
- b. Click **Select an Existing Protection Element**.
- c. Enter data into the Protection Element Search Criteria form. Search by any combination of the fields below:
- **Name** – uniquely identifies the Protection Element.
 - **Object Id** – a string that the Application team assigns to the Protection Element
 - **Attribute Name** – helps to further identify the Protection Element
- d. Click the radio button corresponding with the intended Protection Element name.
- e. Select **View Details**.
- f. Enter data into the Protection Element Details form.
- **Name** – uniquely identifies the Protection Element.
 - **Object Id** – a string that the Application team assigns to the Protection Element
 - **Attribute Name** – helps to further identify the Protection Element
 - **Type** – a string that the application team can assign to indicate type of protection element.
- g. Select **Update** button.
- 3. Delete an Existing Protection Element**
- a. Navigate to the Protection Element Details form by either creating a new Protection Element or Selecting an Existing Protection Element.
- b. Select **Delete**.
- c. In the pop-up window, click **Okay** to confirm intent to delete.
- 4. Assign a Protection Element to a Protection Group or Protection Groups**
- a. Navigate to the Protection Element Details form by either creating a new Protection Element or Selecting an Existing Protection Element.
- b. Select **Associated PGs**.
- c. Determine which of the available Protection Groups to which the Protection Element should be assigned.
1. Select these Protection Groups by highlighting them (see [Assignments and Associations](#) for details).
- d. Click on the **Assign** and **Deassign** buttons until the proper association is displayed.
- e. Save the association by clicking on **Update Association**.

Note: No association is saved until this button is selected.

Privilege

A Privilege refers to any operation performed upon data. Assigning privileges helps

control access to important components of an application (Protection Elements). CSM provides a standard set of privileges that populate automatically when creating the authorization schema (see [Standard Privileges](#) on page 42).

Because Standard Privileges are provided the Privilege section does not contain Create, Delete, or Update functions. However, you may search for and view existing privileges. Use the Role section to assign privileges to roles.

Select an Existing Privilege

- a. Go to the Privilege home page.
- b. Click **Select an Existing Privilege**.
- c. Enter data into the Privilege Search Criteria form. Search **Privilege** name.
- d. Click on the radio button corresponding with the intended **Privilege** name.
- e. Select **View Details**.
- f. View data in the Privilege Details form.
 - **Name** – uniquely identifies the Privilege, required field.
 - **Description** – a brief summary describing the Privilege.

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. In this section you may create new Protection Groups, modify existing Protection Group details, assign Protection Elements, and assign a parent for a Protection Group.

The Protection Group is the only element that can have a Parent. Using Parents is a way to group Protection Groups within Protection Groups. This makes organizing users and their authorization rights easier.

Here are the available Protection Group operations:

1. Create a New Protection Group

- a. Go to the Protection Group home page.
- b. Select **Create a New Protection Group**.
- c. Enter data into the Protection Group Details form.
 - **Name** – uniquely identifies the Protection Group, required field.
 - **Description** – a brief summary describing the Protection Group.
 - **Large Count Flag** – used to indicate if the Protection Group has a large number of associated Protection Elements.
 - **Update Date** – indicates the date when this Protection Group's Details were last updated
- d. Select **Add** button.

2. Select an Existing Protection Group and Update details

- a. Go to the Protection Group home page.
- b. Click **Select an Existing Protection Group**.
- c. Enter data into the Protection Group Search Criteria form. Search by **Protection Group** name.
- d. Click on the radio button corresponding with the intended **Protection Group** name.
- e. Select **View Details**.
- f. Enter data into the Protection Group Details form.

- **Name** – uniquely identifies the Protection Group, required field.
 - **Description** – a brief summary describing the Protection Group.
 - **Large Count Flag** – used to indicate if the Protection Group has a large number of associated Protection Elements.
 - **Update Date** – indicates the date when this Protection Group's Details were last updated
- g. Select **Update** button.
- 3. Delete an Existing Protection Group**
- a. Navigate to the Protection Group Details form by either creating a new Protection Group or Selecting an Existing Protection Group.
 - b. Select **Delete**.
 - c. In the pop-up window, click **Okay** to confirm intent to delete.
- 4. Assign Protection Elements to the Protection Group**
- a. Navigate to the Protection Group Details form by either creating a new Protection Group or Selecting an Existing Protection Group.
 - b. Select **Associated PEs**.
 - c. Determine which of the available Protection Elements should be assigned to the Protection Group.
 1. Select these **Protection Groups** by highlighting them (see [Assignments and Associations](#) for details).
 - d. Click on the **Assign** and **Deassign** buttons until the proper association is displayed.
 - e. Save the association by clicking on **Update Association**.
- Note:** No association is saved until this button is selected.
- 5. Assign a Parent for the Protection Group**
- a. Navigate to the Protection Group Details form by either creating a new Protection Group or Selecting an Existing Protection Group.
 - b. Select **Associated Parent PG**.
 - c. Determine which available Protection Group should be designated as the Protection Group Parent.
 1. Select the **Parent** by highlighting the name. Only one parent may be assigned.
 - d. Click on the **Assign** and **Deassign** buttons until the proper association is displayed.
 - e. Save the association by clicking on **Update Association**.
- Note:** No association is saved until this button is selected.

Role

A Role is a collection of Privileges. By combining Privileges into a Role, it becomes easier to associate Users and Groups with rights to a particular data set. In this section you may create new Roles, modify existing Role details, and assign or deassign Privileges to the Role. Here are the available operations:

1. Create a New Role

- a. Go to the Role home page.
- b. Select **Create a New Role**.
- c. Enter data into the Role Details form.
 - **Name** – uniquely identifies the Role, required field.
 - **Description** – a brief summary describing the Role.
 - **Active Flag** – indicates if the Role is currently active.
- d. Select **Add** button.

2. Select an Existing Role and Update details

- a. Go to the Role home page.
- b. Click **Select an Existing Role**.
- c. Enter data into the Role Search Criteria form. Search by Role name.
- d. Click the radio button corresponding with the intended Role name.
- e. Select **View Details**.
- f. Enter data into the Role Details form.
 - **Name** – uniquely identifies the Role, required field.
 - **Description** – a brief summary describing the Role.
 - **Active Flag** – indicates if the Role is currently active.
- g. Select **Update** button.

3. Delete an Existing Role

- a. Navigate to the Role Details form by either creating a new Role or Selecting an Existing Role.
- b. Select **Delete**.
- c. In the pop-up window, click **Okay** to confirm intent to delete.

4. Assign Privileges to the Role

- a. Navigate to the Role Details form by either creating a new Role or Selecting an Existing Role.
- b. Select **Associated Privileges**.
- c. Determine which of the available Privileges should be assigned to the Role.
 1. Select these **Roles** by highlighting them (see [Assignments and Associations](#) for details). Click on the **Assign** and **Deassign** buttons until the proper association is displayed.
- d. Save the association by clicking on **Update Association**.

Note: No association is saved until this button is selected.

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. Simply select an existing group, and associate a new Protection Group and Roles. Upon doing so, everyone in that particular Group has the same rights.

Under the User portion of UPT you may assign users to Groups. In this section you may create new Groups, modify existing Group details, and associate or disassociate Groups' Protection Groups and Roles. Here are the available operations:

1. Create a New Group

- a. Go to the Group home page.
- b. Select **Create a New Group**.
- c. Enter data into the Group Details form.
 - **Name** – uniquely identifies the Group, required field.
 - **Description** – a brief summary describing the Group.
- d. Select **Add** button.

2. Select an Existing Group and Update details

- a. Go to the Group home page.
- b. Click on **Select an Existing Group**.
- c. Enter data into the Group Search Criteria form. Search by Group name.
- d. Click on the radio button corresponding with the intended Group name.
- e. Select **View Details**.
- f. Enter data into the Group Details form.
 - **Name** – uniquely identifies the Group, required field.
 - **Description** – a brief summary describing the Group.
- g. Select **Update** button.

The Group Details page contains the four buttons displayed in Figure 4-23. The numbers above these buttons correspond to the operations that follow:



Figure 4-23 Group Details page

3. Associated Users ③

- h. Select Associated Users button.
- i. The Group and User Association screen displays a list of Assigned Administrators.
- j. Click Assign User to assign additional Users to the Group.
- k. Click Deassign User to deassign users.
- l. Select Update Association to save the changed associations.
- m. Select Back to return to the Group details screen.

4. Associated PE & Privileges ④

This feature is new to the 3.0.1 release in response to a requirement formed by the caCORE team. This reporting functionality shows a group's privileges for all of its protection elements.

- a. Navigate to the Group Details form by either creating a new User or Selecting an Existing Group.
- b. Select **Associated PE & Privileges**.
- c. View group's privileges for each protection element.

5. Assign a Protection Group and Roles to a Group ⑤

- d. Navigate to the Group Details form by either creating a new Group or Selecting an Existing Group.

- e. Select **Assign PG & Roles**.
- f. Determine which Protection Group and Roles you would like to assign to the Group.
 1. Select the **Protection Group** by highlighting the name (see [Assignments and Associations](#) for details).
 2. Select the Roles by highlighting them.
- g. Click on the **Assign** and **Deassign** buttons until the proper association is displayed.
- h. Save the association by clicking on **Update Association**.

Note: No association is saved until this button is selected.

6. **Update Roles associated with the assigned Protection Groups** 
 - a. Navigate to the Group Details form by either creating a new Group or Selecting an Existing Group.
 - b. Select **Associated PG & Roles**.
 - c. The system displays a list of all associated Protection Groups and their Roles.
 - d. Select the radio button that corresponds with the intended Protection Group.
 - e. Determine which Roles you would like to assign to the Group.
 1. Select the **Role** by highlighting the name (see [Assignments and Associations](#) for details).
 - f. Click on the **Assign** and **Deassign** buttons until the proper association is displayed.
 - g. Save the association by clicking on **Update Association**.

Note: No association is saved until this button is selected.

7. **Delete an Existing Group**
 - a. Navigate to the Group Details form by either creating a new Group or Selecting an Existing Group.
 - b. Select **Delete**.
 - c. In the pop-up window, click **Okay** to confirm intent to delete.

InstanceLevel

Instance Level is a feature provided by CSM to allow filtering of the instance of data directly at the database level by creating filter criteria's and linking them with allowed values from CSM tables (Figure 4-24). In this section you may create upload an application jar file containing the Hibernate file and the Domain Objects, Create a new Filter Clause or Search for existing filter clauses. Please begin by selecting **Upload the Jar File, Add New Security Filter, or Select an Existing Security Filter**

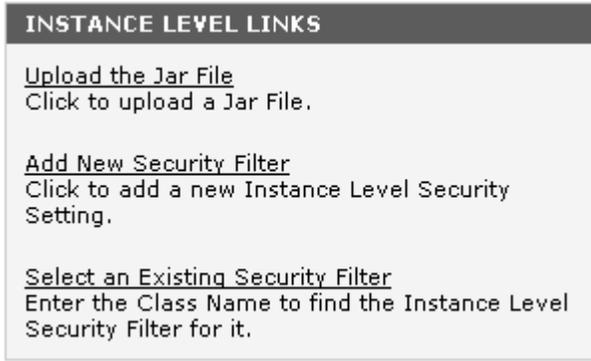


Figure 4-24 Instance Level Links

1. Uploading a File

* indicates a required field

UPLOAD THE APPLICATION JAR FILE	
* Application Jar File	L:\UPT\example-beans.jar <input type="button" value="Browse..."/>
Application Jar File	L:\UPT\example-orm.jar <input type="button" value="Browse..."/>
* Hibernate Configuration File Name	hibernate.cfg.xml

Figure 4-25 Uploading the application jar file

- a. Go to the Instance Level home page.
- b. Select **Upload the Jar File**.
- c. On the File Upload Form enter the following:
 - **Application Jar File** – The path of the application jar file containing hibernate configuration and mapping files and domain object.
 - **Application Jar File** – In case of any SDK generated system there are two jar generated. The second jar can be uploaded using this field
 - **Hibernate Configuration File Name** – The fully qualified name of the hibernate configuration file in the jar.
- d. Select **Upload** button.

2. Add New Security Filter

* indicates a required field

ENTER THE NEW FILTER CLAUSE DETAILS	
* Class Name	gov.nih.nci.cacoresdk.domain.other.levelassociation.Deck
* Filter Chain	gov.nih.nci.cacoresdk.domain.other.levelassociation.Suit - suitCollection gov.nih.nci.cacoresdk.domain.other.levelassociation.Card - cardCollection Add Remove Done
* Target Class Attribute Name	image
Target Class Alias	
Target Class Attribute Alias	

Add Reset Back

Figure 4-26 Entering filter clause details

- a. Go to the Instance Level home page.
- b. Click **Add New Security Filter**.
- c. On the Add New Security Filter screen enter the following:
 - **Class Name** - This the class for which you want to create a filter clause.
 - **Filter Chain** – This is a chain of the associated objects on which the security of the class depends upon. In case of the inherited security you can follow the trail to the target class by selecting the associated class and pressing the **Add** button. You can remove the last associated class by pressing **Remove** button. If the security of the Class is dependant on it own self, then you can select the same Class (with the suffix self) in the Filter Chain. Once you have done selecting the filter chain, you can press **Done** to indicate that.
 - **Target Class Attribute Name** - Field get populated with the all the attributes of the Final Target Class.
 - **Target Class Name** – Alternatively if you want to provide an alias for the Target Class Name then you can do so by providing a value for the Target Class Alias field.
 - **Target Class Attribute Name** – Same way you can provide an alias for the Target Class Attribute Name by providing a value for the Target Class Attribute Alias field.
- d. The click the final **Add** button to add the filter into the screen

3. Selecting an Existing Security Filter

- a. Go to the Instance Level home page.
- b. Click **Select an Existing Security Filter**.
- c. On the Search Criteria Screen enter the following:
 - **Class Name** - This the class for which you want to retrieve the filter clause.

Use * to perform wildcard searches

ENTER THE FILTER CLAUSE SEARCH CRITERIA	
Class Name	<input type="text"/>
<input type="button" value="Search"/> <input type="button" value="Reset"/> <input type="button" value="Back"/>	

Figure 4-27 Entering filter clause search criteria

- d. On the Result screen select the Filter Clause that you want to update or delete.
- e. On the Filter Clause Details Screen there is only one screen editable
 - **Generated SQL** – This is the only editable field on this page. It is the filter SQL that is generated by Hibernate based on filter criteria selected above by the user.

Note: Once you edit the SQL there is no way it can be regenerated without deleting and creating the filter clause again. Also, make sure you follow the Hibernate Filter SQL specifications and have a valid working filtering SQL.

FILTER CLAUSE DETAILS	
Class Name	gov.nih.nci.cabio.domain.Taxon
Filter Chain	geneCollection
Target Class Name	gov.nih.nci.cabio.domain.Gene - geneCollection
Target Class Attribute Name	symbol
Target Class Attribute Type	java.lang.String
Target Class Alias	
Target Class Attribute Alias	
* Generated SQL	TAXON_ID in (select table_name_csm .TAXON_ID from
Update Date	10/02/2007 (MM/DD/YYYY)
<input type="button" value="Update"/> <input type="button" value="Delete"/> <input type="button" value="Back"/>	

Figure 4-28 Filter clause details

- f. In order to update the record click **Upload** button or use the **Delete** button to delete the record.

UPT Installation and Deployment

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 on the NCICB download site (<http://ncicb.nci.nih.gov/download/index.jsp>). The UPT Release contents include the files in Table 4-3.

<i>File</i>	<i>Description</i>
upt.war	The UPT Web 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 that 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 4-3 UPT release contents

Installation Modes

The 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

Single Installation, Single Schema

In the single installation, single schema deployment scheme are shown in Figure 4-29. 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. In addition, all applications can use the same `hibernate-config` file since they point to the same database.

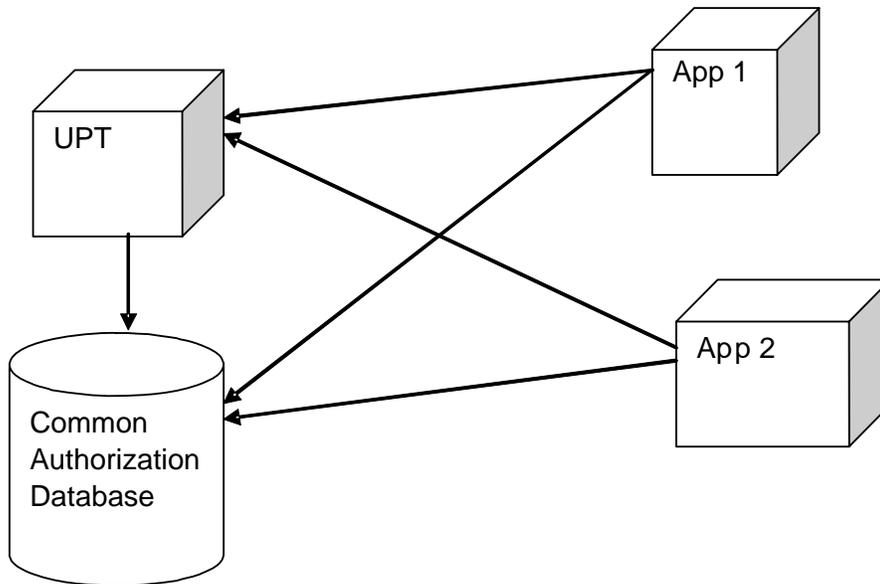


Figure 4-29 Single installation, single schema deployment scheme

Single Installation, Multiple Schema

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 4-30. 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. Therefore, when an application uses the UPT, the UPT communicates to the authorization schema of that application only.

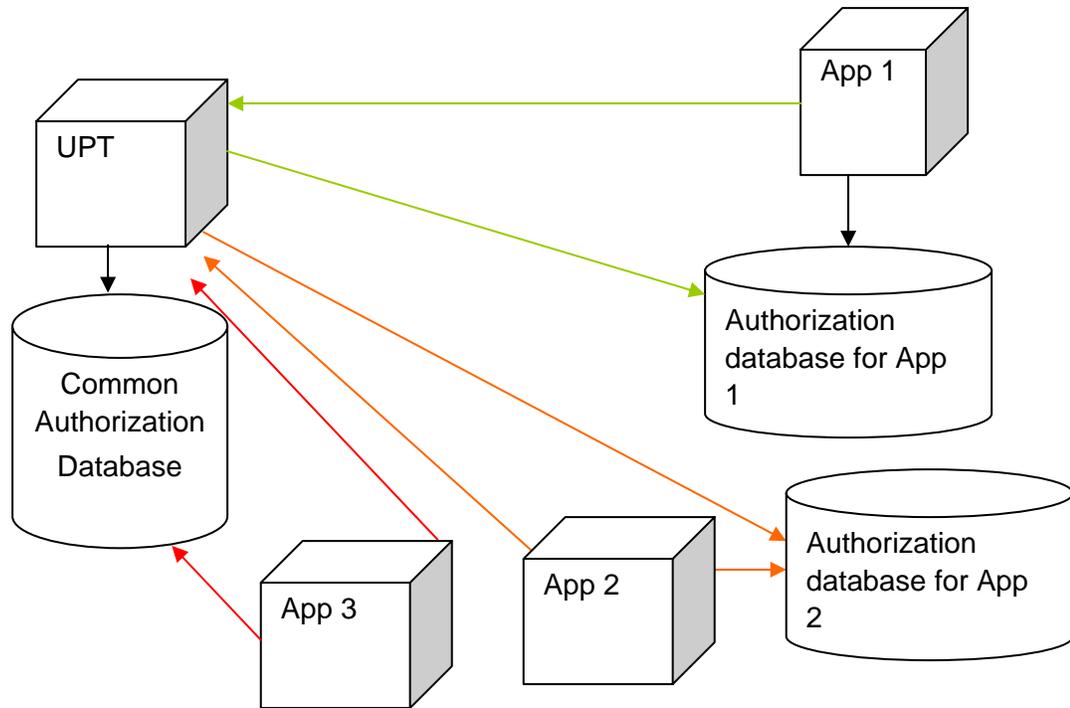


Figure 4-30 Single installation, multiple schemas deployment scheme; the three colors of arrows correspond to the three different applications shown

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 4-31. 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.

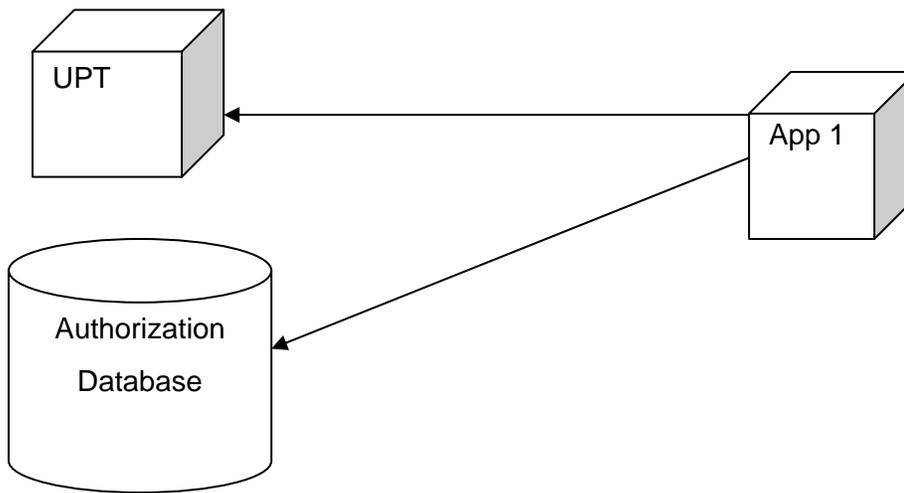


Figure 4-31 Single installation, single schema deployment scheme

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
 - upt.war
 - AuthSchemaMySQL.sql | AuthSchemaOracle.sql
 - DataPrimingMySQL.sql | DataPrimingOracle.sql

Deployment Steps

Step 1: Create and Prime MySQL Database

1. Log into the database using an account id that 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 does not 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 seven entries.

Step 2: Configure Datasource

1. Modify the mysql-ds.xml file that 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 – **csmupt**.
- The <<database_user_id>> with the user id - **ncisecurity**.
<<database_user_password>> with the password of the user account.
- The <<database_url>> with the URL needed to access the Authorization Schema residing on the database server - **jdbc:mysql://<<prod_database_server_name>>:3306/csmupt**

Shown in Figure 4-32 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://Prod_DB.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>
</datasources>
```

Figure 4-32 Example `mysql-ds.xml` file

2. Place the `mysql-ds.xml` file in the JBoss **deploy** directory - `{jboss-home}/server/default/deploy/`.

Step 3: Configure the JBoss JAAS Login parameters

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 4-33. 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`.

```
<application-policy name = "csmupt">
<authentication>
<login-module code =
"gov.nih.nci.security.authentication.loginmodules.LDAPLoginModule"
flag = "required" >
<module-option
name="ldapHost">ldaps://ncids4a.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 4-33 Example login-config.xml entry

As shown in Figure 4-33:

- The `application-policy` is the name of the application for defining the authentication policy – in this case, **csmupt**.
- The `login-module` is the `LoginModule` class that 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 that 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://ncids4a.nci.nih.gov:636</module-
option>
<module-option
name="ldapSearchableBase">ou=nci,o=nih</module-option>
<module-option name="ldapUserIdLabel">cn</module-option>
```

Step 4: Deploy the UPT war file

Copy the `upt.war` in the deployment directory of JBoss, which can be found at `{jboss-home}/server/default/deploy/`.

Step 5: Enable Audit Logging

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. In addition, the `clm.jar` needs to be placed in the common lib directory of the JBoss server.

Step 6: 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.

Chapter 5 Using the CSM Web Services

This chapter describes the Web Service WSDL and its operation and describes an overall workflow and installation for the CSM web service.

Topics in this chapter include:

- [Introduction](#) on this page
- [Web Service WSDL and Operation](#) on this page
- [Workflow for CSM Security Web Service](#) on page 71
- [Installation of CSM Security Web Service](#) on page 71

Introduction

The CSM Web Services are introduced to expose the CSM Authentication and Authorization service features. The Security Web Services currently provide only two operations; namely Login and CheckPermission. The operations are exposed versions available in CSM APIs.

Web Service WSDL and Operation

Security Web Service WSDL

The CSM Security Web Service WSDL is shown in Figure 5-1. The name of the exposed web service is 'SecurityService'. Currently two operations are available: Login and CheckPermission. The web service operations are explained in detail in the following sections.

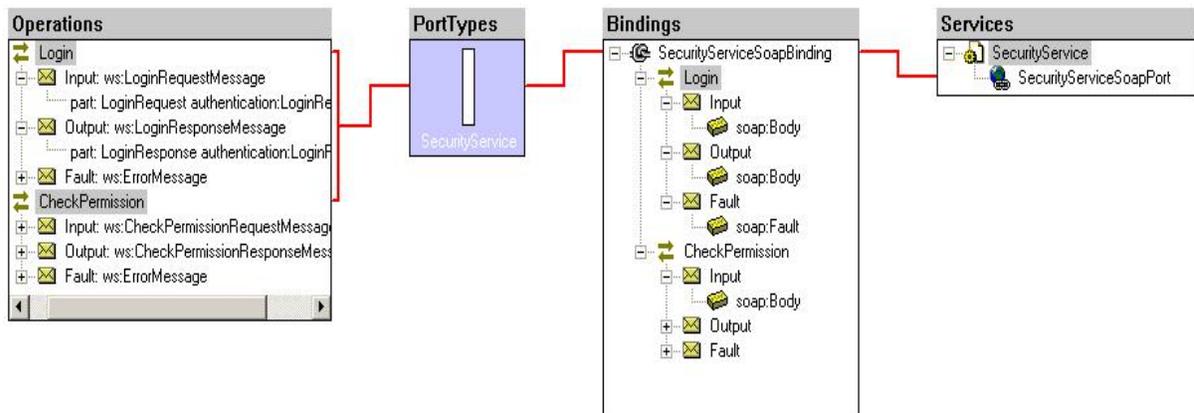


Figure 5-1 Security Web Service WSDL

Login Operation

The Login web service operation is a request/response operation. This operation receives a LoginRequestMessage, performs authentication and responds with LoginResponseMessage to the web service consumer. If there are any problems with the

processing the LoginRequestMessage and/or performing authentication on the user credentials then the web service operation will return a SOAP Fault response error message indicating an error code and the error details.

```
<xs:schema
targetNamespace="http://security.nci.nih.gov/ws/authentication"

xmlns:authentication="http://security.nci.nih.gov/ws/authentication
"

  elementFormDefault="qualified"
  attributeFormDefault="qualified"
  version=".1">
<xs:element name="LoginRequest"
type="authentication:LoginRequest"/>
  <xs:complexType name="LoginRequest">
    <xs:sequence>
      <xs:element name="UserName" type="xs:string"/>
      <xs:element name="Password" type="xs:string"/>
      <xs:element name="ApplicationContext" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="LoginResponse"
type="authentication:LoginResponse"/>
  <xs:complexType name="LoginResponse">
    <xs:sequence>
      <xs:element name="Result" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Figure 5-2 Schema (XSD) for Authentication

As displayed in the Figure 5-2, the LoginRequest message consists of three parameters, Username, password, and ApplicationContext. The Apache AXIS framework validates all request and response messages against the Schema specified in the Security WS WSDL. When the LoginRequest message is received by the web service operation, the User credentials from the LoginRequest message are used by the CSM API to authenticate the user against privilege for the 'ApplicationContext'. If the User is authenticated and has

privilege to access the ApplicationContext then a LoginResponse is returned with result value of 'true'. If the user is not authenticated and does not have access privilege for the 'ApplicationContext' then a LoginResponse is returned with the result value of 'false'.

CheckPermission Operation

The Checkpermission web service operation is a request/response operation. This operation receives a CheckPermissionRequestMessage, performs a permission check, and responds with CheckPermissionResponseMessage. If there are any problems then the web service operation will return a SOAP Fault response error message indicating an error code and the error details.

```

<xs:schema
targetNamespace="http://security.nci.nih.gov/ws/authorization"
  xmlns:authorization="http://security.nci.nih.gov/ws/authorization"
  elementFormDefault="qualified"
  attributeFormDefault="qualified" version=".1">
<xs:element name="CheckPermissionRequest"
type="authorization:CheckPermissionRequest"/>
<xs:complexType name="CheckPermissionRequest">
  <xs:sequence>
    <xs:choice>
      <xs:element name="UserName" type="xs:string"/>
      <xs:element name="GroupName" type="xs:string"/>
    </xs:choice>
    <xs:element name="ObjectId" type="xs:string"/>
    <xs:element name="Attribute" type="xs:string"
nillable="true"/>
    <xs:element name="Privilege" type="xs:string"/>
    <xs:element name="ApplicationContext" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="CheckPermissionResponse"
type="authorization:CheckPermissionResponse"/>
<xs:complexType name="CheckPermissionResponse">
  <xs:sequence>
    <xs:element name="Result" type="xs:boolean"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>

```

Figure 5-3 Schema for Authorization

As displayed in Figure 5-3. The CheckPermission request message consists of User name or Group name, ObjectId, Attribute, Privilege and ApplicationContext. The Apache AXIS framework validates all request and response messages against the Schema specified in the Security WS WSDL. When the CheckPermission request message is received by the web service operation, the CSM API's checkpermission method is invoked to check permission. If the User or Group has permission then a CheckPermissionResponse is returned with result value 'true' otherwise result value is 'false'.

Workflow for CSM Security Web Service

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

1. Read the deployment steps from this document and also read the [Chapter 2](#) and [Chapter 3](#). It provides an overview, workflow, and specific deployment and integration steps.
2. Determine the security requirements and provision security with CSM's UPT.
3. After the Security Web Service is deployed and user security provisioned with UPT. The Security Web Service is ready operable and consumption
4. Using the CSM Web Services Interface use the authentication and authorization operation exposed.
5. Using the LoginRequestMessage invoke and consume Login Web Service Operation.
6. Using the CheckPermissionRequestMessage invoke and consume the CheckPermission Web Service operation.

Installation of CSM Security Web Service

Step 1: Create and Prime Database

1. Log into the database using an account id that 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 does not 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 target applications scheme – **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. For example “doej” for John Doe admin.
 - Also provide the first name and last name for the same by replacing <<super_admin_first_name>> with Doe and <<super_admin_last_name >> with Joe.
5. Replace the <<application_context_name>> with a test application entry – **'abc_app'**. For example, Application name is 'abc_app' and application schema name is 'abc_app'. For the sake of this document we will use schema 'abc_app' and the application as 'abc_app'.

6. Run the script on the database prompt. This should populate the database with the initial data. Verify by querying the application, user, protection_element, and user_protection_element tables. They should have one record each. The database will include CSM Standard Privileges and the privilege table should have seven entries.

Step 2: Configure Datasource

1. Modify the mysql-ds.xml file that contains information for creating a data source. One entry is required for each database connection. Edit this file to replace:
 - The --database_user_name-- with the user id. . --database_user_password--with the password of the user account.
 - The --database_url-- with the URL needed to access the Authorization Schema residing on the database server -
jdbc:mysql://<<stage_database_server_name>>:<<port>>/<<database_name>>
 - Shown in Figure 5-4 is an example mysql-ds.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>abc_app_ds</jndi-name>
    <connection-url>
jdbc:mysql://<<database_server_name>>:<<port>>/<<database_name>></con
nection-url>
    <driver-class>org.gjt.mm.mysql.Driver</driver-class>
    <user-name>name</user-name>
    <password>password</password>
  </local-tx-datasource>
</datasources>
```

Figure 5-4 Example mysql-ds.xml file

2. Place the mysql-ds.xml file in the JBoss **deploy** directory - {jboss-home}/server/default/deploy/

Step 3: Configure the JBoss JAAS Login parameters

1. In order to configure the CSM Web Service to verify against the LDAP, create an entry in the login-config.xml of JBoss as shown in Figure 4-33. This entry configures a login-module against the 'abc_app' application context. The location of this file is {jboss-home}/server/default/conf/login-config.xml.

```

        <application-policy name = "abc_app">
    <authentication>
        <login-module code =
"gov.nih.nci.security.authentication.loginmodules.LDAPLoginModule" flag
= "required" >
            <module-option
name="ldapHost">ldaps://ncids4a.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 5-5 Example login-config.xml entry

As shown in Figure 5-5:

- The `application-policy` is the name of the application for defining the authentication policy – in this case, ‘abc_app’.
- The `login-module` is the `LoginModule` class that 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 that 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://ncids4a.nci.nih.gov:636</module-
option>
<module-option name="ldapSearchableBase">ou=nci,o=nih</module-
option>
<module-option name="ldapUserIdLabel">cn</module-option>

```

Simultaneously you can also point to a RDBMS database containing the username and password information. The configuration steps for the same are provided in [Chapter 2](#) and [Chapter 3](#).

Step 4: Deploy the Security WS war file

Copy the `securityws.war` in the deployment directory of JBoss, which can be found at `{jboss-home}/server/default/deploy/`.

Chapter 6 CSM Instance Level and Attribute Level Security

This chapter describes how to create and use instance and attribute level security.

Topics in this chapter include:

- [Introduction](#) on this page
- [Instance Level](#) on this page
- [Attribute Level](#) on page 80
- [Provisioning Attribute Level Security](#) on page 81

Introduction

Previously CSM APIs provided instance level and attribute level security. This security is now provided in the java tier. The typical flow of events in case of instance level security are as follows. The user fires a business query on the database to obtain the result set. The entire result set is iterated through in java and for each and every record in it, a call is made to the CSM APIs to check whether the user has access to that particular instance. Also, in the case of attribute filtering, for each of the accessible objects in the result set the CSM APIs must be invoked to check which attributes the user can see.

In both solutions mentioned above, there are several issues:

1. The entire result set is to be returned from the database to the application resulting in network traffic and latency.
2. Once the result set is obtained, it needs to be iterated through in java adding to processing time.
3. For each record there is a database call to CSM to determine if the user has access or not.

The CSM 4.0 the design addresses all the performance issues mentioned above.

Instance Level

Requirements Addressed

As part of CSM 4.0, the following functional requirements are addressed and provided as part of the instance level security solution.

1. **Direct Instance Level Security**

This solution provides Direct Instance Level Security, which is defined as the case where security for a particular instance is dependent on itself. A user has access to a particular object based on the value of one of its attributes. There is no relationship or association with another object. This type of instance level security is adhoc and dependant on the associations between that instance and the user by the security admin.

For example, out of 456 patients in a patient table, user ABC has access to 28 based on the patient id.

Out of the total number of patients in the database, the security admin has assigned 28 patient ids to the user ABC. Based on this, the solution should filter any query fired on that patient table such that for user ABC only those 28 records are accessible.

2. Cross Dependant Instance Level Security

This solution provides Cross Dependant Instance Level Security, which is defined as security for a particular instance that is dependent on some other object. A user has access to a particular object based on its association to some other higher level object on which the user has been granted access. There is an association with another object that is generally higher up in the data hierarchy. This type of instance level security is based on the relationship between the queried tables and the table to which the security is assigned. This type of security is used generally where it is much easier to assign and manage security at a higher level of data.

For example, a User has access to only those Lab Results that are associated with a Study (via patients) to which he has access. In this example there can be 1000s of Lab Results where as the Studies could be in the 10s. Also as per the business rule, if you are assigned access to the Study then you can access everything associated with that study. In addition, the assignment and management of security is much easier with Studies as they are fewer in number.

3. Provide Integration of Instance Level Security for an SDK generated system

This solution is integrated with SDK so that it can be provided as an out of the box solution for SDK generated systems.

4. Provide Instance Level Security Support for a Non-SDK system

This solution is adaptable for non-SDK systems with minor modifications. The general principle is the same as that for an SDK generated system. It can be assumed that users will need to configure the solution and adapt it for their application.

Overall Design

In order to provide instance level security, CSM utilizes the filter capability provided by Hibernate. These filters contain filtering queries that are injected to the actual business queries, which are fired by the user. These filters are applied at class level so that whenever the class is queried, the attached filter is appended to the actual business query directly by Hibernate.

CSM provides capabilities for creating these filters through its UPT tool. It allows you to configure these filters for either the Direct or Cross Dependant type of instance level security. These filters contain queries that join with the CSM tables to obtain the instances of data on which the user has access. These filters are stored in the CSM database. At run time the client application calls CSM's helper methods that retrieve these filters from the CSM Database. They also inject these filters into the Hibernate configuration for the appropriate classes.

Since these filters are to be applied for a particular user, the user name is passed as a parameters. At run time filter queries are injected into the actual user queries. This combined query is fired at the database and the resulting data is filtered based on the instances on which the user has access.

Provisioning Instance Level Security

A new menu tab has been added to the UPT for the purpose of provisioning Instance Level Security. This tab allows configuring the filter clauses for various classes in their application.

Once the filtering clauses are configured, the admins can create Protection Elements for the Instances of Objects on which the users have access and assign them access. Details for these operations are provided in [Chapter 4, Using the User Provisioning Tool](#) starting on page 31.

In addition, the Protection Element has been enhanced to include a new value field that the admins can use to provide values for the instances on which the users have access.

Figure 6-1 diagrams the workflow for provisioning instance level security; workflow details are described in the following sections.

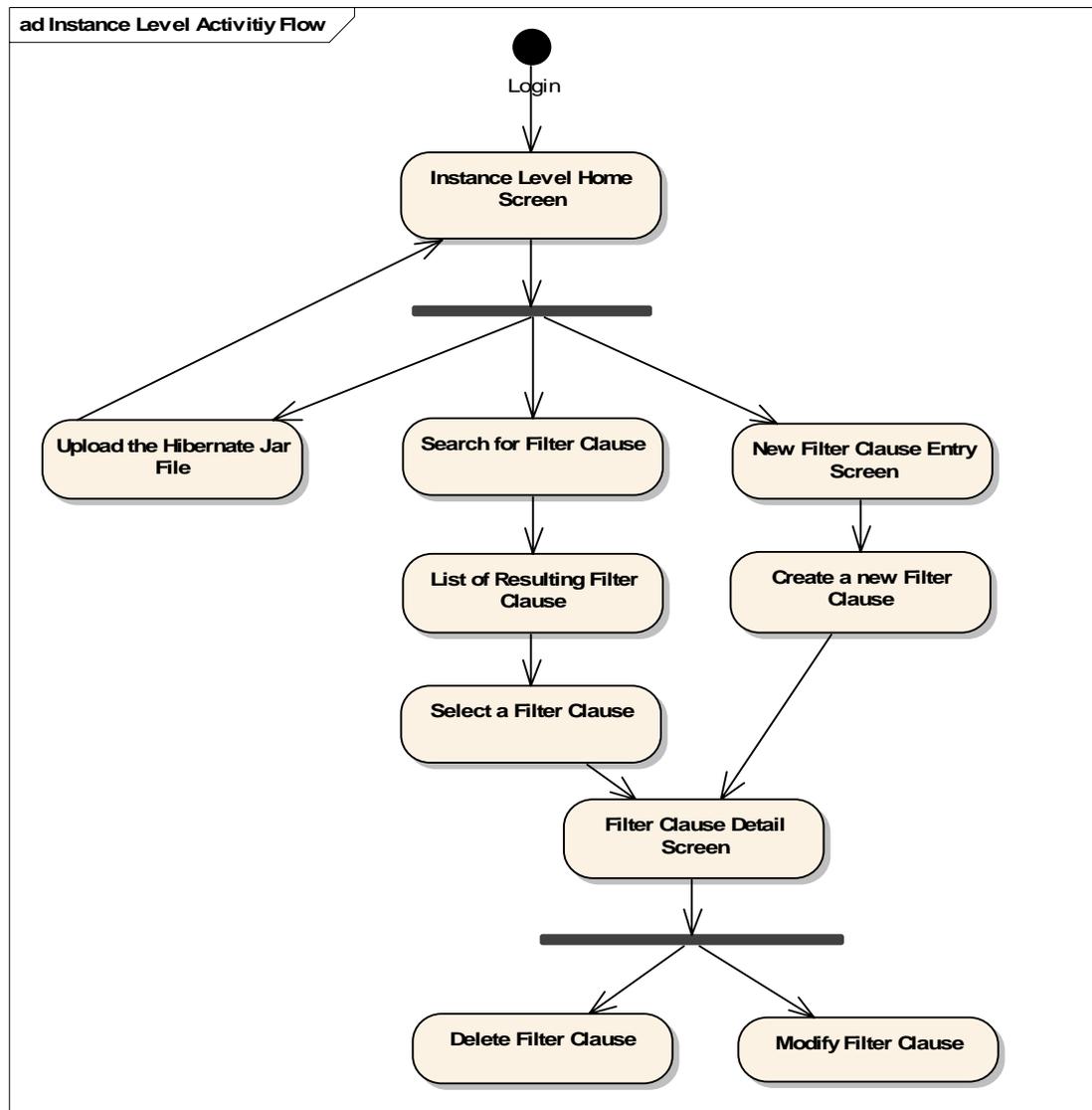


Figure 6-1 Instance Level activity flow

Uploading an Application File

The first step is to upload a file that contains the Hibernate files along with the domain objects. This file should be a valid java archive and contain the following:

- Hibernate Configuration File – with database connection information
- Hibernate Mapping files
- Domain Objects

In the case of an SDK generated system, there are two jar files generated containing the Hibernate and Domain Objects separately. In this case both these files have to be uploaded.

Also a fully qualify Hibernate configuration file name should be provided along with the files. Once the file is successfully uploaded a success message is provided to the user.

Creating a Filter Clause

Once the file containing Hibernate information is uploaded, use it to create filter clause for different objects.

1. On the filter clause screen, select the class for which you want to add the filter. Once the class is selected, the second combo box is automatically populated with the associated classes.

Note: There is an entry for the master class itself in the list. This is to allow for direct instance level security.

If you want to provision a Direct Instance Level Security then select the class itself in the second combo and click **Done**.

2. In the case of Cross Dependant Security, select the associated class in the second combo box.

Note: You can drill down the class hierarchy by clicking **Add** and displays the associated child classes. Once you have navigated to the final class on which the security for the class is dependant, click **Done**.

The attribute list combo is populated with the attributes from the last class in the filter chain. Select the attribute whose value the user will be granted access.

3. Once selected you can also provide an alias for the target class name and attribute. Do this in the case where the attribute selected holds value for some other class. For example, a Patient Object has the attribute Security Key whose value you want to filter the instances. However, from a business perspective, the actual value in the Security Key is the value of the Study Id to which the patient belongs. In this case even though the security filter is set for the Patient based on the Security Key attribute, in a business sense the filtering is happening at the Study Id level. Hence, you can provide this alias that will be used to determine the protection elements on which the user has been granted access.
4. Once everything is selected, click **Add Filter** to create the filter. Once the filter is created a filtering SQL is generated and displayed back to the user.

Note: This field is editable to allow users to modify the SQL in case if they want to optimize it further.

Creating Protection Element

Once the security filters have been created, you need to provision the actual instances on which the user has access. This is done by creating protection elements for these instances and providing the access to the users.

Table 6-1 contains descriptions of the Protection Element Fields that the admin has to create and grant to the user.

<i>Field Name</i>	<i>Description</i>
Protection Element Name	Distinct name that can identify the Protection Element.
Protection Element Description	Description for the Protection Element.
Protection Element Type	Can be left blank.
Protection Element Object Id	The target class name on which the security of the master class depends. If an alias class name is used, then the alias should be entered here.
Protection Element Attribute	The name of the attribute of the target class on which the security of the master class depends. If an alias attribute name is used then the alias should be entered here.
Protection Element Value	The actual value of the attribute on which user has access.
Update Date	Date when the protection element was last updated.

Table 6-1 Protection Element fields

Using Instance Level Security

In order for the Client application to inject Instance Level Security, CSM provides a helper class to assist it. This class contains methods that allow the user to add filters to the Hibernate configuration file at the time of system loading and also to initialize and parameterize these filters at run time for a particular user who is firing the query.

```
public static void addFilters( AuthorizationManager
authorizationManager, Configuration configuration)
```

This method should be called only once for an application just after the Hibernate Configuration object is created by reading the configuration file and before the Session Factory Method is created. This method injects the security filters that are created for this application. It retrieves a list of all the filters that have been defined for this application from the CSM Database. For each filter in the list, it creates a new FilterDefinition (Hibernate) object. It then retrieves the Persistent Class from the passed Configuration Object using the class name for which the filter is defined. It then adds the filter to the persistent class by setting the filtering query.

```
public static void initializeFilters (String userName, Session
session, AuthorizationManager authorizationManager)
```

This method is invoked after obtaining the Session from the SessionFactory and just before executing the user query. This method initializes the filters that are already added to the SessionFactory. This method first obtains the list of all the defined filters from the

SessionFactory in the passed Session object. It then iterates through the filter list and sets the user name and the application name parameter. It retrieves the Application Name from the passed Authorization Manager.

Known Issues

1. In case of eager loading filtering of the child object does not work
2. Hibernate, by default, injects only the filter for the parent object. In case the eager loading mode is set to true, the child object's (the associated objects which are eagerly loaded) filter are not injected. SDK, by default, has eager loading set to false, requiring the user to explicitly turn it on.
3. Multiple filters on a single object will be always ANDed
4. If you have multiple filters defined for a single domain object, Hibernate will inject all of them with the AND condition between them. This is the default behavior of Hibernate and requires programmatic enhancements to handle the ORing of filters.
5. Filtering in the case of inheritance needs to be further investigated.
6. The Hibernate DTD has a limitation that does not allow the user to add a filter for the inherited classes. The DTD allows filters only to be added to the super class. However, the Hibernate API allows adding these filters. This issue will be investigated in detail during implementation and results will be posted accordingly.

Attribute Level

Requirements Addressed

As part of CSM 4.0, the following functional requirements are addressed and provided as part of the attribute level security solution.

1. **Attribute Level Security**

This solution provides Attribute level security at the object level, which is defined as security where you can control access to the attributes of an object. A user can be granted and revoked access to these attributes and, based on the user's access level, those attributes should be visible to the user.

For example, a Patient object has the following five attributes: Name, Address, Social Security, Phone Number, and Disease. A researcher who has access to all the attributes except Social Security should be able to see the Patient object with data for all attributes except the Social Security attribute.

2. **Single or multiple object retrieval**

This solution provides Attribute level security both for queries, which result in a single object returned from the database, as well as a list of the objects returned from the database. In this case, each object in the list should be filtered based attributes to which the user has access too.

3. **Automatically provide Attribute Level Security for an SDK generated system**

Attribute Level Security is integrated with the SDK so that it can be provided as an out of the box solution for SDK generated systems.

4. Provide Attribute Level Security Support for a Non-SDK system

This solution should be adaptable for non-SDK systems with minor modifications, if required. The general principle should be same as that of an SDK generated system. It can be assumed that users will need to configure the solution and adapt it for their application.

Overall Design

CSM utilizes the SessionInterceptor feature provided by Hibernate to inject attribute level security. It traps a user session during the loading of an object from the underlying database. During the load process it intercepts the incoming stream of result data from the underlying database and checks which attributes the user has access to. If the user does not have access then it nullifies the attribute value such that the resulting object contains values for only those attributes to which they do have access.

Since it would need to access the CSM table to check if a user has access to an attribute every time an object is loaded, the solution implements a cache that holds the user's attribute access map. The interceptor checks against this cache to inject attribute level security. Using this method, the overall filtering process is sped up.

Provisioning Attribute Level Security

There are new special changes in the UPT for provisioning Attribute Level Security. If attribute level security is turned on, by default all object attributes are secured. Therefore, if you want to grant access to an attribute to the user then you will have to create a protection element for that attribute and grant access to it to the user like any other protection element.

Table 6-2 contains the description of the Protection Element fields which an admin has to create and grant to the user.

<i>Field Name</i>	<i>Description</i>
Protection Element Name	Distinct name which can identify the Protection Element.
Protection Element Description	Description for the Protection Element.
Protection Element Type	Can be left blank.
Protection Element Object Id	The class name on whose attribute the user is to be granted access.
Protection Element Attribute	The attribute name on which the user is to be granted access.
Protection Element Value	Can be left blank.
Update Date	Date when the protection element was last updated.

Table 6-2 Protection Element fields

Using Attribute Level Security

In order to use Attribute Level Security, the Client Application must attach the attribute level

Session interceptor to its session. This can be done at the time of obtaining the Hibernate Session from the SessionFactory object as shown below. Once the session interceptor is in place, it injects Attribute level security every time an object is loaded from the database for a query.

```
Session session = sessionFactory.openSession(new  
AttributeSecuritySessionInterceptor());
```

Known Issues

1. Eager loading the attribute filtering happens only for the parent object.

The onLoad method is invoked for each record returned from the database. However this works only for the parent object, so if you have eager loading set to true, the child object's (the associated objects that are eagerly loaded) attributes are not filtered. The SDK, by default, has eager loading set to false requiring the user to explicitly turn it on.

2. Primitive attribute type filtering is not possible.

Since a primitive data type cannot be set to null, the current attribute solution does not work if the domain objects contain primitive data types such as attribute. The default values for primitive (0 for int, false for a boolean) can be a valid value. Hence, setting primitive attributes to their default values is also not an option.

3. Filtering on queries with projection on certain attributes does not work

Queries where the user has set a project on certain attributes of the object rather than returning the whole object back do not work. This is because, in the case of projections, Hibernate returns the attribute value directly from the database as a Java data type. As a result, the onLoad method of the session interceptor is not invoked and thereby does not inject the attribute level security.

Chapter 7 CSM Acegi Adapter

This chapter describes the CSM Acegi Adapter, which allows applications to use CSM's Authentication and Authorization under the Acegi Security Framework.

Topics in this chapter include:

- [Introduction](#) on this page
- [Implementation](#) on this page
- [Integrating and Configuring](#) on page 85

Introduction

The Acegi framework (<http://www.acegisecurity.org/>) is quickly becoming the preferred framework for many Spring framework (<http://www.springframework.com/>) powered applications to implement security. Acegi Security is the de facto standard for security in Spring Framework. Existing applications and new applications wanting to leverage CSM can do so now with the CSM Acegi Adapter. The CSM Acegi Adapter allows applications to use CSM's Authentication and Authorization under the Acegi Security Framework.

CSM Acegi Adapter implementation provides Authentication, Authorization - Method Level Security and Object Parameter level security.

Implementation

Acegi Security is widely used within the Spring community for comprehensive security services to Spring-powered applications. It comprises a set of interfaces and classes that are configured through a Spring IoC container. The design of Acegi Security allows many applications to implement the common enterprise application security requirements via declarative configuration settings in the IoC container. Acegi Security is heavily interface-driven, providing significant room for customization and extension. Important Acegi Security, like Spring, emphasizes pluggability.

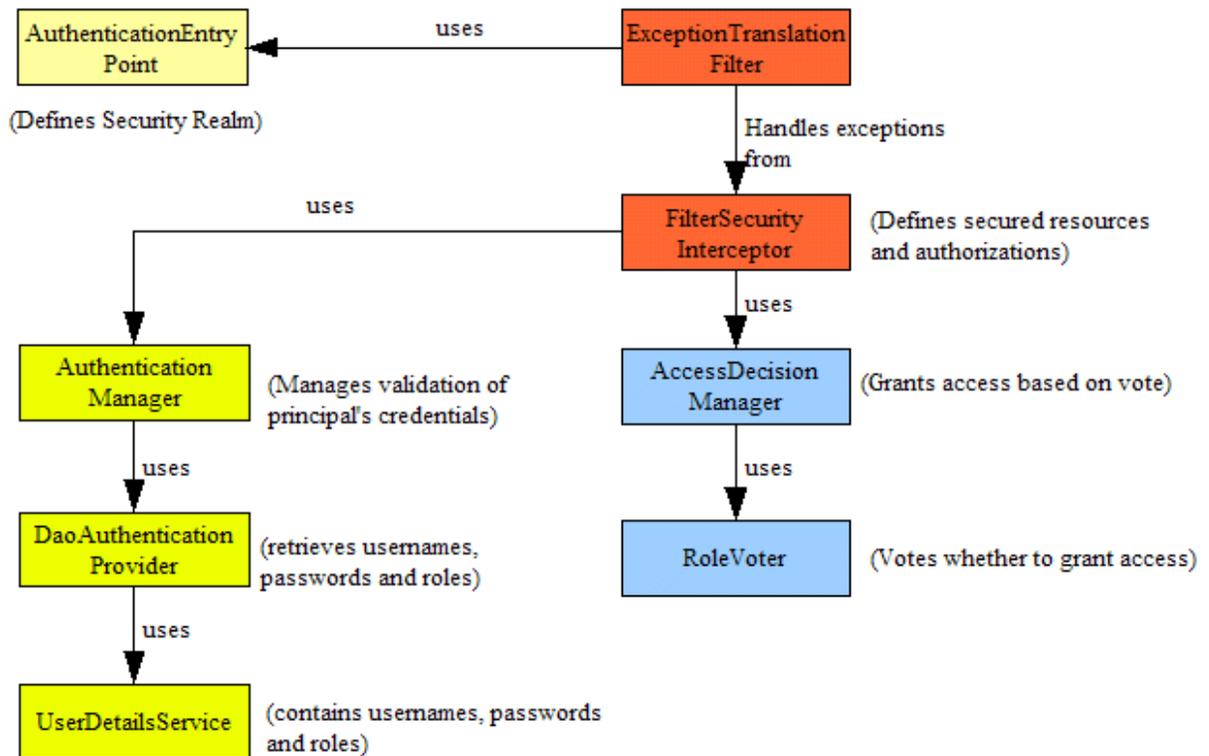


Figure 7-1 Authentication and Authorization in Acegi Framework

Figure 7-1 demonstrates the control flow by Acegi for authentication and authorization. The CSM Acegi Adapter uses this approach to provide the CSM Adapter. Authentication is implemented by extending this design. Acegi provides Interceptors that can be configured through Acegi Security Configurations in Spring. For a detailed understanding of the Acegi Frameworks Authentication and Authorization implementation by CSM please refer to the next section.

Note: The detailed explanation of Acegi interfaces that are implemented by CSM Acegi Adapter is beyond the scope of this guide. Refer the Acegi Security CSM Adapter Design document for details ([acegi security reference documentation](#)).

Currently the CSM Acegi Adapter implementation provides Method Level and Method Parameter Level security.

Method Level Security

The current out-of-the-box implementation of the CSM Acegi Adapter provides method level security. The Adapter implements Acegi’s MethodInterceptor. The CSMMethodSecurityInterceptor, CSM’s custom implementation of the MethodInterceptor, enables security at the method level by intercepting method calls on the secured bean specified in the MethodDefinitionSource. All the methods will be intercepted for each secured bean. Please see the [Workflow](#) and [Integrating and Configuring](#) sections for more details.

Method Parameter Level Security

In this implementation the CSM Acegi Adapter provides method parameter level security. Applications that need method parameter level security have to implement CSM's SecurityHelper. The SecurityHelper interface, provided by CSM, allows the application to control authorization. Refer to the [CSM API source](#) for more details.

Workflow

1. Determine the level of security required for your application – Method level, Object Parameter Level etc.
2. Define the beans that need to be protected.
3. Define appropriate Security Interceptors.
4. Define Security Interceptors for various beans that need protection.
5. Configure the csm-acegi-security.xml acegi security configuration file.
6. Configure a JAAS LoginModule for the Application Context.
7. Configure Database properties.
8. Configure User provisioning using CSM UPT.

Integrating and Configuring

This section serves as a guide to help developers integrate applications with CSM Acegi Adapter. It outlines a step by step process that addresses what developers need to know in order to successfully integrate CSM's Acegi Adapter into their applications, which includes:

- Configure Acegi Security in csm-acegi-security.xml
- Database properties and configuration
 - Configure Datasource OR
 - Configure Hibernate configuration file
- LDAP properties and configuration
- Provision user access authorization policy

Configure Acegi Security

1. Define the beans that need to be protected.

For Example from Appendix A:

```
<bean id='applicationService'
class='test.gov.nih.nci.security.acegi.sdk.ApplicationServiceImpl' />
```

This configuration will secure ApplicationServiceImpl class and intercept all its method calls.

2. Define the SecurityHelper Impl Class. This class needs to be implemented by the developers that want to integrate CSM Adapter into their new or existing Application with Acegi Security Framework. In this implementation it is a custom CSMMethodSecurityInterceptor that intercepts any method calls on the 'applicationService' bean.

Example:

```
<bean id='securityHelper'  
class='test.gov.nih.nci.security.acegi.sdk.SecurityHelperImpl' />
```

3. List the beans that need to be protected by the 'securityInterceptor' for the 'autoProxyCreator'.

Example:

```
<bean id='autoProxyCreator'  
class='org.springframework.aop.framework.autoproxy.BeanNameAutoPr  
oxyCreator'>  
  <property name='interceptorNames'>  
    <list>  
      <value>securityInterceptor</value>  
    </list>  
  </property>  
  <property name='beanNames'>  
    <list>  
      <value>applicationService</value>  
    </list>  
  </property>  
</bean>
```

4. Specify the Application Context that will be used for CSM's Authentication and Authorization service.

Example:

```
<bean id="userDetailsService"  
  class="gov.nih.nci.security.acegi.authentication.CSMUserDetail  
sService">  
<!-- Specify the Application Context required by CSM -->  
  <property name="csmApplicationContext">  
    <value>acegittest</value>  
  </property>  
</bean>
```

Database Properties and Configuration

Create and Prime 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. 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 that has permission to create new databases. 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 example, "acegittest").

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: The default password is always “changeme” and this should be 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.

Configure Datasource

1. Modify the provided `mysql-ds.xml` or `oracle-ds.xml` file that 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, “*acegittest*”).
 - 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 7-2 is an example of the `mysql-ds.xml` file.

```
<datasources>
  <local-tx-datasource>
    <jndi-name>csmupt</jndi-name>
    <connection-url>jdbc:mysql://mysql_db: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>acegitest</jndi-name>
    <connection-url>jdbc:mysql://mysql_db: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 7-2 Example *mysql-ds.xml* file

2. Place the *mysql-ds.xml* or *oracle-ds.xml* file in the JBoss deploy directory. If the integrating Application does not want to use datasources then the Hibernate configuration file can be used.

Configure Hibernate Configuration File

1. Modify the provided *mysql-ds.xml* or *oracle-ds.xml* file that 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 7-3 is an example of the *acegitest.new.csm.hibernate.xml* file for application context 'acegitest'.

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration
DTD 2.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-2.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property
name="connection.url">jdbc:mysql://<<server>>:<<port>>/acegitest</property>
    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="connection.username">USERNAME</property>
    <property name="connection.password">PASSWORD</property>
    <property
name="connection.driver_class">org.gjt.mm.mysql.Driver</property>
    <property name="hibernate.show_sql">>false</property>
    <property name="connection.zeroDateTimeBehavior">convertToNull</property>
    <property name="hibernate.cache.use_query_cache">>false</property>
    <property name="hibernate.cache.use_second_level_cache">>false</property>

    <mapping
resource="gov/nih/nci/security/authorization/domainobjects/Privilege.hbm.xml"/>
    <mapping
resource="gov/nih/nci/security/authorization/domainobjects/Application.hbm.xml"/>
    <mapping
resource="gov/nih/nci/security/authorization/domainobjects/Role.hbm.xml"/>
    <mapping
resource="gov/nih/nci/security/dao/hibernate/RolePrivilege.hbm.xml"/>
    <mapping resource="gov/nih/nci/security/dao/hibernate/UserGroup.hbm.xml"/>
    <mapping
resource="gov/nih/nci/security/dao/hibernate/ProtectionGroupProtectionElement.hbm.
xml"/>
    <mapping
resource="gov/nih/nci/security/authorization/domainobjects/Group.hbm.xml"/>
    <mapping
resource="gov/nih/nci/security/authorization/domainobjects/User.hbm.xml"/>
    <mapping
resource="gov/nih/nci/security/authorization/domainobjects/ProtectionGroup.hbm.xml

```

Figure 7-3 Example *acegitest.new.csm.hibernate.cfg.xml*

```
resource="gov/nih/nci/security/authorization/domainobjects/UserProtectionElement.
hbm.xml" />

    </session-factory>

</hibernate-configuration>
```

Figure 7-4 Example *acegittest.new.csm.hibernate.cfg.xml* (con't)

Configure JAAS LoginModule

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 7-5.

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

Figure 7-5 *Configuring a login module*

For *acegittest*, which uses *RDBMSLoginModule*, the JAAS configuration file entry is shown in Figure 7-6.

```

acegittest
{
gov.nih.nci.security.authentication.loginmodules.RDBMSLoginModule
Required
driver=" org.gjt.mm.mysql.Driver"
url=" jdbc:mysql://<<server>>:<<port>>/acegittest "
    user="USERNAME"
    passwd="PASSWORD"
query="SELECT * FROM users WHERE username=? and password=?"
encryption-enabled="YES";
}

```

Figure 7-6 Acegittest application JAAS configuration file entry

The configuration file entry contains the following:

- The application is `acegittest`.
- 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 that 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=" org.gjt.mm.mysql.Driver "`

`url=" jdbc:mysql://<<server>>:<<port>>/acegittest "`

`user="USERNAME"`

`passwd="PASSWORD"`

`query="SELECT * FROM users WHERE username=? and password=?"`

`encryption-enabled="YES"`

As shown in the code above, since 'acegittest' application has only one credential provider, only one corresponding entry was made in the configuration file. If the application uses multiple credential providers, then the `LoginModule`'s can be stacked. A single configuration file can contain entries for multiple applications.

User Provisioning via UPT

- Create Protection Elements for objects that need to be secured.
- Create Protection Group for the Protection Elements.
- Create a Role with Privilege assigned to it.
- Create a User.
- Assign Protection Group and Role to the Users that are allowed access.

Chapter 8 CSM caGrid Integration

This chapter describes how CSM can be leveraged in the caGrid environment to authenticate and authorize the user trying to make a service call.

Topics in this chapter include:

- [Introduction](#) on this page
- [Authentication](#) on page 93
- [Authorization](#) on page 95
- [Migrating from CSM v3.2 to CSM v4.0](#) on page 96

Introduction

caGrid (<http://cagrid.org>) is a core infrastructure project of the cancer Bio Informatics Grid. It consists of architectural components and tools that enable any applications to be deployed on the grid as a node. It also provides tools for discovering these services and invoking them.

In order to be able to securely invoke the grid services, the caGrid architecture needs to authenticate and authorize the user trying to make the service call. This would require both an authoring mechanism to provide appropriate permissions to the user and a run time mechanism to verify these granted permission.

Since CSM provides the above mentioned capabilities, the below mentioned solution describes how CSM can be leveraged in the grid environment.

Authentication

CSM is enhanced to return a subject for a user upon authentication. This subject contains the user's attributes like Last Name, First Name, and Email Id that are required to prepare the SAML that is to be sent to Dorian (<http://www.cagrid.org/mwiki/index.php?title=GAARDS:Main>).

CSM Configuration for IdP/Authentication Service

As part of v3.2, CSM was 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 that 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** -
gov.nih.nci.security.authentication.principal.FirstNamePrincipal
- **Last Name** -
gov.nih.nci.security.authentication.principal.LastNamePrincipal
- **Email Id** -
gov.nih.nci.security.authentication.principal.EmailIdPrincipal
- **First Name** -
gov.nih.nci.security.authentication.principal.LoginIdPrincipal

Both RDBMSLoginModule and LDAPLoginModule have been updated to return these attributes. The next two sections discuss how this is accomplished.

Configuring RDBMS Login Module for CSM-caGrid IDP Integration

If an application uses an RDBMS Server from which the user attributes are to be retrieved, the above mentioned attribute mapping should be added in the JAAS login-config file. Figure 8-1 is a sample entry of the JAAS login.conf file.

```
RDBMSGRID{  
    gov.nih.nci.security.authentication.loginmodules.RDBMSLoginModule  
    Required  
  
    driver="org.gjt.mm.mysql.Driver"  
  
    url="jdbc:mysql://mysql_db_server:3620/CSMAuthSchema"  
  
    user="USER "  
  
    passwd="PASSWORD"  
  
    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";  
  
};
```

Figure 8-1 Sample JAAS login.conf file

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 [Audit Logging section](#) below.

Configuring 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. Figure 8-2 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";
};
```

Figure 8-2 Sample JAAS login.conf file

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

Authorization

Using Grid Group Names for Check Permission

As part of the CSM caGrid Integration, CSM now allows users to check permission using the

Grid Grouper Group Name. Earlier the check permission method took only the user name and checked permission for that particular user. However now new methods have been introduced which can take in a group name and check permission against the group name.

Alternatively there are other two methods provided which return the list of all the groups which have the said privilege on a particular resource. Figure 8-3 contains the method definition. More details are provided in the javadocs.

```
public boolean checkPermissionForGroup(String groupName, String objectId,
String attributeName, String privilegeName) throws CSEException;

public boolean checkPermissionForGroup(String groupName, String objectId,
String privilegeName) throws CSEException;

public List getAccessibleGroups(String objectId, String privilegeName) throws
CSEException;

public List getAccessibleGroups(String objectId, String attributeName, String
privilegeName) throws CSEException;
```

Figure 8-3 Methods for checking permission against a group name

Note: If you are using Group level security then at the time of provisioning you need make sure that the group name provided to the group (via UPT) is same as the Grid Grouper group name.

Migrating from CSM v3.2 to CSM v4.0

MySQL Migration

The following procedure defines in detail the steps needed to update the MySQL database from an existing 3.1 authorization schema to a new 4.0 authorization schema:

1. Obtain the CSM API v4.0 Release from NCICB Download Center (<http://ncicb.nci.nih.gov/download>)
2. In the MigrationScript3.2MySQL.sql from the CSM API v4.0 Release, change the <<database_name>> with the name of the database.
3. Go to the directory that contains the executables for MySQL and provide the following command.

```
mysql --user=[user_name] --password=[password] -h [hostname]
[auth_schema] < MigrationScript4.0MySQL.sql
```

- [user_name] is the user name used to connect the MySQL database
- [password] is the password for the user name
- [hostname] is the host URL where the MySQL database is hosted. If you are running this command from the same machine where MySQL is hosted, you do not need to provide this parameter.
- [auth_schema] is the name of the database created using the new authorization schema.
- [MigrationScript4.0MySQL.sql] is the file containing the data exported from the old schema, which needs to be loaded into the new schema

4. Verify that there are no errors in the SQL Script executed. Also make sure that the database has been appropriately updated.

Oracle Migration

The following procedure defines in detail the steps needed to update the Oracle database from an existing 3.2 authorization schema to a new 4.0 authorization schema:

1. Obtain the CSM API v4.0 Release from NCICB Download Center (<http://ncicb.nci.nih.gov/download>).
2. Log onto Oracle Server into the Schema where the CSM Database is present using either SQL Plus or TOAD or any other tool.
3. Copy all the SQL commands from MigrationScript4.0Oracle.sql from the CSM API v4.0 Release, and paste them on the SQL Editor/Console. Now execute all these commands in a batch.
4. Verify that there are no errors in the SQL Script executed. Also make sure that the database has been appropriately updated.

Appendix A References

Articles

1. The Description Logic Handbook. Franz Baader, et al. (eds.). Cambridge University Press, 1993.
2. Artificial Intelligence. Patrick Winston. Addison-Wesley, 1984.
3. Artificial intelligence. Minsky M, Hillis D, Rudisch G. New England Journal of Medicine. 1980 Jun 26;302(26):1482.
4. Java Programming: <http://java.sun.com/learning/new2java/index.html>
5. Extensible Markup Language: <http://www.w3.org/TR/REC-xml/>
6. XML Metadata Interchange: <http://www.omg.org/technology/documents/formal/xmi.htm>

caBIG Material

1. caBIG: <http://cabig.nci.nih.gov/>
2. caBIG Compatibility Guidelines: http://cabig.nci.nih.gov/guidelines_documentation

caCORE Material

1. NCICB: <http://ncicb.nci.nih.gov/NCICB/infrastructure>
2. caCORE: <http://ncicb.nci.nih.gov/NCICB/infrastructure>
3. caBIO: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/caBIO
4. caDSR: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/cadsr
5. CSM: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/csm

Software Products

1. Java: <http://java.sun.com>
2. Ant: <http://ant.apache.org/>

Appendix B CSM Acegi Sample Configuration File

This appendix provides the CSM Acegi Sample configuration file.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
'http://www.springframework.org/dtd/spring-beans.dtd'>
<beans>

  <!-- This is the bean that needs to be protected. -->
  <bean id='applicationService'
    ='test.gov.nih.nci.security.acegi.xyzApp.ApplicationServiceImpl' />
    <!--The application integrating CSM Acegi adapter needs to provide
    actual implementation for SecurityHelper. The class name to reflect the
    impl of SecurityHelper-->
  <bean id='securityHelper'
    ='test.gov.nih.nci.security.acegi.xyzApp.SecurityHelperImpl' />

  <!-- This bean defines a proxy for the protected bean. Notice that -->
  <!-- the id defined above is specified. When an application asks Spring --
  >
  <!-- for a applicationService it will get this proxy instead. -->
  <bean id='autoProxyCreator'
    ='org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator'>
    <property name='interceptorNames'>
      <list>
        <value>securityInterceptor</value>
      </list>
    </property>
    <property name='beanNames'>
      <list>
        <value>applicationService</value>
      </list>
    </property>
  </bean>

  <!-- This bean specifies which roles are authorized to execute which
  methods. -->
  <bean id='securityInterceptor'
    ='gov.nih.nci.security.acegi.CSMMethodSecurityInterceptor'>
    <property name='securityHelper' ref='securityHelper' />
    <property name='authenticationManager'
      ='authenticationManager' />
    <property name='accessDecisionManager'
      ='accessDecisionManager' />
    <property name='afterInvocationManager'
      ='afterInvocationManager' />
    <property name='objectDefinitionSource'
      ='csmMethodDefinitionSource' />
  </bean>

  <bean id='csmMethodDefinitionSource'
    ='gov.nih.nci.security.acegi.authorization.CSMMethodDefinitionSource'>
    <property name='methodMapCache'
      ='ehCacheBasedMethodMapCache' />
  </bean>
</beans>
```

```

</bean>
<bean id='ehCacheBasedMethodMapCache'
='gov.nih.nci.security.acegi.authorization.EhCacheBasedMethodMapCache'>
  <property name="cache">
    <bean
      ="org.springframework.cache.ehcache.EhCacheFactoryBean">
      <property name="cacheManager">
        <bean
          ="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"
        />
      </property>
      <property name="cacheName" value="userCache" />
    </bean>
  </property>
</bean>

<!-- This bean specifies which roles are assigned to each user. -->
<bean id="userDetailsService"
="gov.nih.nci.security.acegi.authentication.CSMUserDetailsService">
  <!--
-->
  <!-- Specify the Application Context required by CSM -->
  <!--
-->
  <property name="csmApplicationContext">
    <value>acegittest</value>
  </property>
</bean>

<!-- This bean specifies that a user can access the protected methods -->
<!-- if they have any one of the roles specified in the
objectDefinitionSource above. -->
<bean id='accessDecisionManager'
='org.acegisecurity.vote.AffirmativeBased'>
  <property name='decisionVoters'>
    <list>
      <ref bean='roleVoter' />
    </list>
  </property>
</bean>

<!-- The next three beans are boilerplate. They should be the same for
nearly all applications. -->
<bean id='authenticationManager'
='org.acegisecurity.providers.ProviderManager'>
  <property name='providers'>
    <list>
      <ref bean='authenticationProvider' />
    </list>
  </property>
</bean>

<bean id='authenticationProvider'
='gov.nih.nci.security.acegi.authentication.CSMAuthenticationProvider'>
  <property name='userDetailsService' ref='userDetailsService' />
</bean>

```

Appendix B CSM Acegi Sample Configuration File

```
<bean id='roleVoter'  
='gov.nih.nci.security.acegi.authorization.CSMRoleVoter' />  
  
<bean id='afterInvocationManager'  
='gov.nih.nci.security.acegi.CSMAfterInvocationProviderManager'>  
  <property name='providers'>  
    <list>  
      <ref bean='afterInvocationProvider' />  
    </list>  
  </property>  
</bean>  
  
<bean id='afterInvocationProvider'  
='gov.nih.nci.security.acegi.CSMAfterInvocationProvider' />  
  
</beans>
```


Glossary

The following table contains a list of terms used in this document, with accompanying definitions.

Term	Definition
Acegi	Acegi is a security framework that provides a powerful, flexible security solution for enterprise software, with a particular emphasis on applications that use the Spring Framework. Acegi Security provides comprehensive authentication, authorization, instance-based access control, channel security, and human user detection capabilities. See http://www.acegisecurity.org/ for more information.
Ant	Apache Ant is a Java-based build tool used to perform various build related tasks. For more information on how Ant is used within the SDK. See http://ant.apache.org/ for more information on Ant itself.
caGrid	The cancer Biomedical Informatics Grid, or caBIG™, is a voluntary virtual informatics infrastructure that connects data, research tools, scientists, and organizations to leverage their combined strengths and expertise in an open federated environment with widely accepted standards and shared tools. The underlying service oriented infrastructure that supports caBIG™ is referred to as caGrid. See http://www.cagrid.org
Ehcache	Ehcache is a simple, fast and thread safe cache for Java that provides memory and disk stores and distributed operation for clusters. CSM uses ehcache in conjunction with Hibernate. See http://sourceforge.net/projects/ehcache for more information.
Hibernate	Hibernate is an object-relational mapping (ORM) solution for the Java language, and provides an easy to use framework for mapping an object-oriented domain model to a traditional relational database. Its purpose is to relieve the developer from a significant amount of relational data persistence-related programming tasks. See http://www.hibernate.org/ for more information.
JAR	JAR file is a file format based on the popular ZIP file format and is used for aggregating many files into one. A JAR file is essentially a zip file that contains an optional META-INF directory.
JAAS	The JAAS 1.0 API consists of a set of Java packages designed for user authentication and authorization. It implements a Java version of the standard Pluggable Authentication Module (PAM) framework and compatibly extends the Java 2 Platform's access control architecture to support user-based authorization.

Term	Definition
SAML	Security Assertion Markup Language (SAML) is an XML standard for exchanging authentication and authorization data between security domains, that is, between an identity provider (a producer of assertions) and a service provider (a consumer of assertions). SAML is a product of the OASIS Security Services Technical Committee
Spring	Spring Framework is a leading full-stack Java/JEE application framework. Led and sustained by Interface21, Spring delivers significant benefits for many projects, increasing development productivity and runtime performance while improving test coverage and application quality. See http://www.springframework.org/ for more information.
WSDD	An acronym for Web Service Deployment Descriptor, which can be used to specify resources that should be exposed as Web Services. See http://ws.apache.org/axis/java/user-guide.html#CustomDeploymentIntroducingWSDD for more information.
WSDL	An acronym for Web Services Definition Language, which is an XML-based language that provides a model for describing Web services. See http://www.w3.org/TR/wsdl.html or http://en.wikipedia.org/wiki/WSDL for more information.

Index

A

- Acegi adapter
 - database properties and configuration, 86
 - implementing, 83
 - integrating and configuring, 85
 - method level security, 84
 - workflow, 85
- Acegi framework, 83
- Admin
 - defined, 5, 31
- Ant, 8
- Attribute level security
 - defined, 5, 75
 - known issues, 82
 - provisioning, 81
 - using, 81
- Audit logging
 - defined, 5, 26
 - deploying, 29
 - enabling CLM APIs with CSM APIs, 26
- Authentication
 - defined, 5
- AuthenticationManager, 10
- Authorization
 - activate CLM logging, 25
 - database properties and configuration, 23
 - defined, 5
 - importing CSM Authorization Manager class, 20
 - installing and deploying, 23
 - software and scripts, 21
- AuthorizationManager, 10

C

- CLM
 - activate logging for authorization, 25
 - activating audit logging, 20
- Credential Provider, 7
- CSM
 - Acegi adapter, 83
 - API workflow, 9

- architecture, 6
- AuthenticationManager, 10
- AuthorizationManager, 10
- CSM Authentication Service
 - configuring lockout, 12
 - importing, 10
 - installing and configuring, 12
- CSM Security Web Service
 - installing, 71
 - workflow, 71

I

- Instance level security
 - defined, 5, 75
 - known issues, 80
 - provisioning, 76
 - using, 79

J

- JAAS, 7
 - configuring LDAP login module, 17
 - configuring login module, 13
- JBOSS, 8
 - configuring LDAP login module, 17
 - configuring login module, 15
- JDK, 8

L

- LDAP, 8
 - configuring login module in JAAS, 17
 - configuring login module in JBOSS, 17
 - configuring login module using anonymous bind, 18
 - credential provider properties, 16
- Login Module, 8

M

- MySQL, 8

O

Oracle, 8

R

RDBMS, 8

credential provider properties, 13

enabling encryption in login module, 16

S

Security concepts

defined, 7

Spring framework, 83

Super Admin

defined, 5, 31

System requirements, 8

T

Tomcat, 8

U

UPT

Admin, 31

Admin mode, 44

assign privileges, roles, protection groups, groups, 36

assign users, protection elements, 37

assigning privileges, 42

create new element, 32

defined, 25

delete element, 36

example error messages, 33

installing and deploying, 58

Login page, 32

managing users, 41

search for existing element, 34

Super Admin, 31

super admin mode, 39

update element, 35

workflow, 31

User provisioning

defined, 5

User Provisioning Tool. *See* UPT

W

Web Service WSDL

security web service WSDL, 67