Table of Contents

The security manager is used when the client or a user has to be authenticated. In this case, the login data will be transmitted by the client to the server for verification. The server will always delegate the verification of login data to a defined security manager.

After successful verification, the client can complete the login process. In the case of an error, a detailed error message will be shown.

Our goal is the implementation and use of a security manager. The valid user/password combinations are shown in a hash table.

Implementing the Security Manager

Create the server class `apps.firstapp.security.HashtableSecurityManager` and implement the interface `com.sibvisions.rad.server.security.ISecurityManager`. For example:

HashtableSecurityManager.java

```
/*
 * Copyright 2009 SIB Visions GmbH
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you
may not
 * use this file except in compliance with the License. You may obtain
a copy of
 * the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT
 * WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See
the
 * License for the specific language governing permissions and
limitations under
 * the License.
 *
 *
 * History
 *
 * 28.09.2009 - [JR] - creation
 */
package apps.firstapp.security;

import java.util.Hashtable;

import javax.rad.remote.IConnectionConstants;
import javax.rad.server.ISession;

import com.sibvisions.rad.server.security.ISecurityManager;
```

```
/**
 * The <code>HashtableSecurityManager</code> is a {@link Hashtable}
 * based {@link ISecurityManager} implementation.
 *
 * @author René Jahn
 */
public class HashtableSecurityManager implements ISecurityManager
{
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   // Class members
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   /** contains username/password mapping. */
   private Hashtable<String, String> htUsers = new Hashtable<String,
String>();

   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   // Initialization
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   /**
    * Creates a new instance of <code>HashtableSecurityManager</code>
with
    * predefined users.
    */
   public HashtableSecurityManager()
   {
      htUsers.put("username", "password");
      htUsers.put("jvx", "jvx");
   }

   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   // Interface implementation
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   /**
    * {@inheritDoc}
    */
   public synchronized void validateAuthentication(ISession pSession)
   {
      checkUser(pSession);
   }

   /**
    * {@inheritDoc}
    */
   public synchronized void changePassword(ISession pSession)
   {
      checkUser(pSession);

      //check old password with current password!
```

```
      if (!htUsers.get(pSession.getUserName()).equals(
pSession.getProperty(IConnectionConstants.OLDPASSWORD)))
      {
         throw new SecurityException("Invalid password");
      }

      //user is valid -> change the password
      htUsers.put(pSession.getUserName(),
(String)pSession.getProperty(IConnectionConstants.NEWPASSWORD));
   }

   /**
    * {@inheritDoc}
    */
   public synchronized void logout(ISession pSession)
   {
   }

   /**
    * {@inheritDoc}
    */
   public synchronized IAccessController getAccessController(ISession
pSession)
   {
      return null;
   }

   /**
    * {@inheritDoc}
    */
   public synchronized void release()
   {
   }

   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   // User-defined methods
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   /**
    * Checks if the user is known and the password is valid.
    *
    * @param pSession the session to authenticate
    */
   private void checkUser(ISession pSession)
   {
      String sPwd = htUsers.get(pSession.getUserName());

      if (sPwd == null)
      {
         throw new SecurityException("User not found!");
      }
```

```
      if (!sPwd.equals(pSession.getPassword()))
      {
         throw new SecurityException("Invalid password");
      }
   }

}   // HashtableSecurityManager
```

The interface `com.sibvisions.rad.server.security.ISecurityManager` defines the methods

```
public void validateAuthentication(ISession pSession) throws Exception;

public void changePassword(ISession pSession) throws Exception;

public void logout(ISession pSession);

public IAccessController getAccessController(ISession pSession);

public void release();
```

`validateAuthentication` verifies that the user exists and that the password is valid. If the login data is invalid or access is denied for any other reason, a predefined exception is thrown. The exception message is translated at the client and displayed to the user if necessary.

`changePassword` allows a password change. If the change fails or is not required, an exception is thrown. The exception message is translated at the client and displayed to the user if necessary.

`logout` tells the security manager that the session was ended. This is either done manually by the user or automatically after the expiration of the timeout period.

`getAccessController` limits access to Life Cycle Objects. This method is called when a master connection is established to control access to sub-connections and sub-sessions. When a sub-connection is established, we can verify if access is allowed using IAccessController.isAllowed.

`release` tells the security manager that all used resources should be released.

Using the Security Manager

The security manager has to be defined in the configuration file of the application (config.xml), to ensure that it is used at the next login attempt. For example:

config.xml

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

<application>
  <securitymanager>
```

```
    <class>apps.firstapp.security.HashtableSecurityManager</class>
  </securitymanager>

  ...

</application>
```

Note

The standard configuration only instantiates a security manager once per application, and it is reused at every login attempt. It is, therefore, important to pay attention to synchronization and to make sure that the constructor does not contain essential code.

It's also possible to create a security manager per session, but this needs manual configuration.