RELEASE 9.3.3.0.00

IMPLEMENTING A CUSTOM AUTHENTICATION MODULE

ORACLE
ENTERPRISE PERFORMANCE MANAGEMENT SYSTEM

CONTENTS IN BRIEF

About Hyperion Custom Authentication

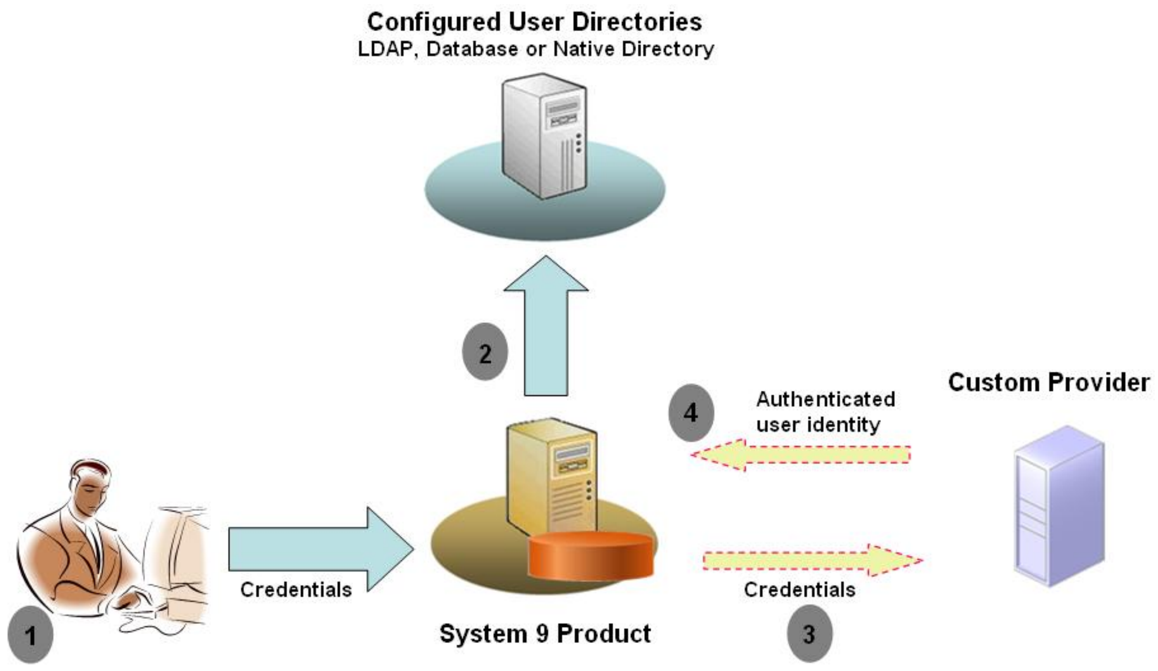
A custom authentication module is a Java module that a customer develops and implements to authenticate Hyperion users. Generally, Hyperion products use a login screen to capture the user name and password, which are used to authenticate users. Instead of using Hyperion authentication, you can write a custom authentication module to authenticate users belonging to the user directory configured in an Hyperion environment.

Implementing a custom authentication module does not involve modifying Hyperion products. You deploy the custom Java module on the server to seamlessly integrate your own authentication module with Hyperion.

A custom authentication module may be used with both the thick clients (for example, Oracle's Hyperion® Smart View for Office) and thin clients (for example, Oracle's Hyperion® Workspace).

The custom authentication module uses the information a user enters when logging into an Hyperion product. If enabled for a user directory, it authenticates users through the custom authentication module. On successfully authenticating the user, the custom authentication module returns the user name.

The following illustration presents a sample custom authentication scenario:


Configured User Directories
LDAP, Database or Native Directory
2
4
Authenticated user identity
Custom Provider
Credentials
Credentials
1
System 9 Product
3

For example, you can use RSA SecurID infrastructure as the custom provider to ensure transparent strong authentication to Hyperion products. The overview of steps:

1. The user enters credentials (generally, user name and password) to access an Hyperion product. These credentials should uniquely identify the user to the provider used by the custom authentication module. For example, if you are using an RSA SecurID infrastructure to authenticate users, the user enters an RSA user ID and PIN (not an Hyperion user ID and password).

2. Using the search order (see *Oracle Hyperion Enterprise Performance Management System Security Administration Guide* for details), Hyperion cycles through configured user directories to locate the user.
   - If the current user directory is not configured for custom authentication, Hyperion tries to locate and authenticate the user through Hyperion authentication.
   - If the user directory is configured for custom authentication, Hyperion delegates the authentication process to the custom module.
3. If Hyperion delegates authentication to the custom module, the custom authentication module accepts the credentials and uses its own logic to direct user authentication against a custom provider; for example, RSA SecurID infrastructure.
4. If the custom authentication module authenticates the user against its provider, it returns the user name to Hyperion, or it returns a Java exception.

   The user name returned by the custom authentication module must be identical to a user name in one of the user directories that is enabled for custom authentication.
   - If the custom authentication module returns a user name, Hyperion locates the user in a user directory that is enabled for custom authentication. At this stage, Hyperion does not search the user directories that are not configured for custom authentication.
   - If the custom authentication module throws an exception or returns a null user, Hyperion continues to search for the user in the remaining user directories in the search order. If a user that matches the credentials is not found, Hyperion displays an error.

Use Case Examples and Limitations

Custom authentication implementation scenarios include the following:

- Adding one-time password support
- Performing authentication against a Resource Access Control Facility (RACF)
- Adding Simple Authentication and Security Layer (SASL) bind to LDAP-enabled user directories instead of simple LDAP binds

Authentication with challenge/response mechanism may not work well if you implement a custom authentication module. Custom messages thrown by the custom authentication module are propagated to the clients.

Design and Coding Considerations

Subtopics

Search Order

In addition to Native Directory, multiple user directories can be configured in Oracle's Hyperion® Shared Services. After configuring, you should add external user directories to the search order, which determines the sequence in which Shared Services cycles through the user directories to locate users. You can modify the search order from Oracle's Hyperion® Shared Services User Management Console. Excepting Native Directory, you can remove configured user directories from the search order. Hyperion does not use the user directories that are not included in the search order. See the *Oracle Hyperion Enterprise Performance Management System Security Administration Guide*.

The search order determines the order in which Hyperion cycles through the user directories to authenticate users. If the user is authenticated in a user directory, Hyperion stops the search and returns the user. Hyperion denies authentication and returns an error if the user cannot be authenticated against any user directories in the search order.

Impact of Custom Authentication on Search Order

The use of custom authentication affects how Hyperion security interprets the search order.

If the custom authentication module returns a user name, Hyperion locates the user only in a user directory that is enabled for custom authentication. At this stage, Hyperion ignores user directories that are not configured for custom authentication.

Understanding the Custom Authentication Flow

The following use case scenarios are used to explore custom authentication flow:

- "Use Case Scenario 1" on page 4
- "Use Case Scenario 2" on page 6
- "Use Case Scenario 3" on page 6

Use Case Scenario 1

Table 1 details the Hyperion user directory configuration and search order used in this scenario. This scenario assumes that the custom authentication module uses an RSA infrastructure to authenticate users.

Table 1 Setup for Scenario 1

User Directory Type and Name	Search Order	Custom Authentication	Sample User Names	Password[1]

Native Directory	1	Disabled	`test_user_1`
			`test_user_2`
			`test_user_3`
				`password`

LDAP-Enabled
SunONE_West
	2	Disabled	`test_ldap1`
			`test_ldap_2`
			`test_user_3`
			`test_ldap_4`
				`ldappassword`

LDAP-Enabled
SunONE_East
	3	Enabled	`test_ldap1`
			`test_ldap_2`
			`test_user_3`
				`ldappassword` on SunONE and `RSA PIN` in custom module

[1]For simplicity, it is assumed that all users use the same user directory password.

To initiate the authentication process, a user enters a user name and password in the login screen of an Hyperion product.

In this scenario, the custom authentication module performs the following actions:

- Accepts a username and RSA PIN as the user credentials
- Returns a username in username@providername format or as username, for example, `test_ldap_2@SunONE_East` or `test_ldap_2` to Hyperion security.

Table 2 User interaction and results

User Name and Password	Authentication Result	Login User Directory
`test_user_1/password`	Success	Native Directory
`test_user_3/password`	Success	Native Directory
`test_user_3/ldappassword`	Success	SunONE_West (search order 2)[1]
`test_user_3/RSA PIN`	Success	SunONE_East (search order 3)[2]
`test_ldap_2/ldappassword`	Success	SunONE_West (search order 2)
`test_ldap_4/RSA PIN`	Failure Hyperion displays an authentication error.[3]	

[1]The custom authentication cannot authenticate this user because the user entered Hyperion credentials. Hyperion can identify this user only in a user directory that is not enabled for custom authentication. The user is not in Native Directory (search order number 1)but is identified in SunONE West (search order number 2).

[2]Hyperion does not find this user in Native Directory (search order number 1) or SunONE West (search order number 2). The custom authentication module validates the user against RSA Server and returns `test_user_3@SunONE_EAST` to Hyperion. Hyperion locates the user in SunONE East (search order number 3), which is a custom authentication–enabled user directory.

[3]Oracle recommends that all users authenticated by the custom module be present in a custom authentication–enabled user directory included in the search order. Login fails if the user name that is returned by the custom authentication module is not present in a custom authentication–enabled user directory included in the search order.

Use Case Scenario 2

Table 3 details the Hyperion user directory configuration and search order used in this scenario. This scenario assumes that the custom authentication module uses an RSA infrastructure to authenticate users.

In this scenario, the custom authentication module performs the following actions:

- Accepts a user name and RSA PIN as the user credentials
- Returns a user name, for example, `test_ldap_2` to Hyperion security

Table 3 Setup for Scenario 2

User Directory	Search Order	Custom Authentication	Sample User Names	Password[1]

Native Directory	1	Disabled	`test_user_1`
			`test_user_2`
			`test_user_3`
				`password`

LDAP-Enabled, for example, SunONE	2	Enabled	`test_ldap1`
			`test_ldap2`
			`test_user_3`
				`ldappassword` on SunONE and `RSA PIN` in custom module

[1]For simplicity, it is assumed that all users use the same user directory password.

To initiate the authentication process, a user enters a user name and password in the login screen of an Hyperion product.

Table 4 User interaction and results

User Name and Password	Login Result	Login User Directory
`test_user_1/password`	Success	Native Directory
`test_user_3/password`	Success	Native Directory
`test_user_3/ldappassword`	Failure	SunONE[1]
`test_user_3/RSA PIN`	Success	SunONE[2]

[1]Authentication of user against Native Directory fails because of password mismatch. Authentication of user using the custom authentication module fails because password used is not a valid RSA PIN. Hyperion does not try to authenticate this user in SunONE (search order 2), because custom authentication settings override Hyperion authentication in this directory.

[2]Authentication of user against Native Directory fails because of password mismatch. The custom authentication module authenticates the user and returns the user name `test_user_3` to Hyperion.

Use Case Scenario 3

Table 5 details the Hyperion user directory configuration and search order used in this scenario. This scenario assumes that the custom authentication module uses an RSA infrastructure to authenticate users.

For clarity in scenarios such as this, Oracle recommends that your custom authentication module return the user name in `username@providername` format; for example, `test_ldap_4@SunONE`.

Table 5 Setup for Scenario 3

User Directory	Search Order	Custom Authentication	Sample User Names	Password[1]
Native Directory	1	Enabled	`test_user_1` `test_user_2` `test_user_3`	`RSA PIN`
LDAP-Enabled, for example, MSAD	2	Disabled	`test_ldap1` `test_ldap4` `test_user_3`	`ldappassword`
LDAP-Enabled, for example, SunONE	3	Enabled	`test_ldap1` `test_ldap4` `test_user_3`	`ldappassword` on SunONE and `RSA PIN` in custom module

[1]For simplicity, it is assumed that all users use the same user directory password.

To initiate the authentication process, a user enters a user name and password in the login screen of an Hyperion product.

Table 6 User interaction and results

User Name and Password	Authentication Result	Login User Directory
`test_user_1/RSA PIN`	Success	Native Directory
`test_user_3/RSA PIN`	Success	Native Directory
`test_user_3/ldappassword`	Success	MSAD (search order 2)
`test_ldap_4/ldappassword`	Success	MSAD (search order 2)
`test_ldap_4/RSA PIN`	Success	SunONE (search order 3)

User Directories and Custom Authentication Module

To use the custom authentication module, user directories that contain Hyperion user and group information can be individually configured to delegate authentication to the custom module.

Hyperion users who are authenticated using a custom module must be present in one of the user directories included in the search order (see "Search Order" on page 4). Also, the user directory must be configured to delegate authentication to the custom module.

The identity of the user in the custom provider (for example, `1357642` in RSA SecurID infrastructure) may be different from the user name in the user directory (for example, `jDoe` in an Oracle Internet Directory) configured in Shared Services. After authenticating the user, the custom authentication module must return the user name `jDoe` to Hyperion.

Note: As a best practice, Oracle recommends that the user name in the user directories configured in Hyperion be identical to those available on the user directory used by the custom authentication module.

`CSSCustomAuthenticationIF` Java Interface

The custom authentication module must use the `CSSCustomAuthenticationIF` Java interface to integrate with Hyperion security framework. It must return a user name string if custom authentication is successful or a an error message if authentication is not successful. For the authentication process to be completed, the user name returned by the custom authentication module must be present in one of the user directories included in Shared Services search order. Hyperion security framework supports the *`username@providerName`* format.

Note: Ensure that the user name that the custom authentication module returns does not contain an * (asterisk) because Hyperion security framework interprets it as a wildcard character while searching for users.

See "Sample Code 1" on page 8 for `CSSCustomAuthenticationIF` interface signature.

Your custom authentication module can be a single class file. It must be included in `CustomAuth.jar`. The package structure is unimportant.

For detailed information about the `CSSCustomAuthenticationIF` interface, see Security API documentation.

Authenticate Method

The `authenticate` method from `CSSCustomAuthenticationIF` supports custom authentication. The `authenticate` method accepts credentials (user name and password) that the user entered while trying to access the Hyperion as input parameters. This method returns a string (user name) if custom authentication is successful. It throws a `java.lang.Exception` if authentication is not successful. The user name returned by the method should uniquely identify a user in one of the user directories included in Shared Services search order. Hyperion security framework supports the *`username@providerName`* format.

See "Sample Code 1" on page 8 for authenticate method signature.

Note: To initialize resources; for example, a JDBC connection pool, use the class constructor. Doing so improves performance by not loading resources for every authentication.

Sample Code 1

The following code snippet is an empty implementation of the custom module.

```
package com.hyperion.css.custom;
import java.util.Map;
import com.hyperion.css.common.CSSCustomAuthenticationIF;
import org.apache.log4j.Logger; // imports Log4j's Logger
```

```
  public class CustomAuthenticationImpl
               implements CSSCustomAuthenticationIF {
    //Get the Logger to log exception or debug information
    //Log information is written to the Shared Services security log
    static Logger logger=Logger.getLogger
                        ("com.hyperion.css.commonustom.CustomAuthenticationImpl");
      public String authenticate(Map context,String userName,
                                 String password) throws Exception{
        try{
          //Custom code to find and authenticate the user goes here.
          //The code should do the following:
          //if authentication succeeds:
                //set authenticationSuccessFlag = true
                //return authenticatedUserName
          // if authentication fails:
                //ensure debug is enabled using logger.isDebugEnabled()
                //log an authentication failure
                //throw authentication exception
        }
        catch (Exception e){
          //Custom code to handle authentication exception goes here
          //Create a new exception, set the root cause
          //Set any custom error message
          //Return the exception to the caller
        }
        return authenticatedUserName;
        }
  }
```

Input parameters are as follows:

- context: A map that contains key-value pair of locale information
- user name: An identifier that uniquely identifies the user to the user directory where the custom module authenticates the user. The user enters the value of this parameter while logging into an Hyperion product.
- password: The password set for the user in the user directory where the custom module authenticates the user. The user enters the value of this parameter while logging into an Hyperion product.

Sample Code 2

The following sample code demonstrates custom authentication of users using user name and password contained in a flat file. You must initialize user and password lists in the class constructor to make this work.

```
package com.hyperion.css.security;
import java.util.Map;
import java.util.HashMap;
import com.hyperion.css.common.CSSCustomAuthenticationIF;
import java.io.*;

public class CSSCustomAuthenticationImpl implements CSSCustomAuthenticationIF{
  //get the Logger to log Exception or other info useful for debugging
  /*static Logger logger = Logger.getLogger
    ("com.hyperion.css.custom.CSSCustomAuthenticationImpl"); */
  static final String DATA_FILE = "datafile.txt";
/**
  * authenticate method includes the core implementation of the
  * Custom Authentication Mechanism. If custom authentication is
  * enabled for the provider, authentication operations
  * are delegated to this method. Upon successful authentication,
  * this method returns a valid user name, using which System 9
  * retrieves the user from a custom authentication enabled provider.
  * User name can be returned in the format username@providerName,
  * where providerName indicates the name of the underlying provider
  * where the user is available. authenticate method can use other
  * private methods to access various core components of the
  * custom authentication module.
  *   @param context
  *   @param userName
  *   @param password
  *   @return
  *   @throws Exception
*/
Map users = null;
public CSSCustomAuthenticationImpl(){
  users = new HashMap();
  InputStream is = null;
  BufferedReader br = null;
  String line;
  String[] userDetails = null;
  String userKey = null;
  try{
     is = CSSCustomAuthenticationImpl.class.getResourceAsStream(DATA_FILE);
     br = new BufferedReader(new InputStreamReader(is));
     while(null != (line = br.readLine())){
        userDetails = line.split(":");
          if(userDetails != null && userDetails.length==3){
            userKey = userDetails[0]+ ":" + userDetails[1];
            users.put(userKey, userDetails[2]);
          }
      }
     }
  catch(Exception e){

        }
  finally{
     try{
        if(br != null) br.close();
        if(is != null) is.close();
     }
     catch(IOException ioe){
        ioe.printStackTrace();
     }
  }
}
/* Use this authenticate method snippet to return username from a flatfile */

public String authenticate(Map context,String userName, String password)throws
Exception{
  //UserName : user input for the userName
  //Password : user input for password
  //context  : Map, can be used to additional information required by
  //           the custom authentication module.

  String authenticatedUserKey = userName + ":" + password;

  if(users.get(authenticatedUserKey)!=null)
     return(String)users.get(authenticatedUserKey);
  else throw new Exception("Invalid User Credentials");
    }

/* Refer to this authenticate method snippet to return username in
   username@providername format */

public String authenticate(Map context,String userName, String password)throws
Exception{

  //UserName : user input for userName
  //Password : user input for password
  //context  : Map, can be used to additional information required by
  //           the custom authentication module.

  //Your code should uniquely identify the user in a custom provider
  //and in a configured user directory in Shared Services. system 9
  //Security expects you to append the provider name to the user name.
  //Provider name must be identical to the name of a custom
  //authentication-enabled user directory specified in Shared Services.

  //If invalid arguments, return null or throw exception with
  //appropriate message and set authenticationSuccessFlag = false

  String authenticatedUserKey = userName + ":" + password;
  if(users.get(authenticatedUserKey)!=null)
     String userNameStr = (new StringBuffer())
                           .append((String)users.get(authenticatedUserKey))
                           .append("@").append(PROVIDER_NAME).toString();
            return userNameStr;
  else throw new Exception("Invalid User Credentials");
    }
}
```

Data File for Sample Code 2

Ensure that the data file is named `datafile.txt`, which is the name used in the sample code, and that it is included in the Java archive that you create.

Use the following as the contents of the flat file that is used as the custom user directory to support the custom authentication module implemented by Sample Code 2 (see "Sample Code 2" on page 9).

```
xyz:password:admin
test1:password:test1@LDAP1
test1:password:test1
```

```
test1@LDAP1:password:test1@LDAP1
test1@1:password:test1
user1:Password2:user1@NTLM1
user1_1:Password2:user1
user3:Password3:user3
DS_User1:Password123:DS_User1@MSAD1
DS_User1:Password123:DS_User1
DS_User1@1:Password123:DS_User1
```

Use the following as the contents of the flat file that is used as the custom user directory if you plan to return user name in `username@providername` format.

```
xyz:password:admin
test1:password:test1
test1@1:password:test1
user1_1:Password2:user1
user3:Password3:user3
DS1_1G100U_User61_1:Password123:DS1_1G100U_User61
DS1_1G100U_User61_1@1:Password123:DS1_1G100U_User61
TUser:password:TUser
```

Deploying the Custom Authentication Module

Only one custom module is supported for an Hyperion deployment. You can enable custom authentication for one or more user directories in the search order.

The custom authentication module must implement the public interface `CSSCustomAuthenticationIF`, which is defined in the `com.hyperion.css.common` package. This document assumes that you have a fully functional custom module that defines the logic for authenticating users against the user provider of your choice. After you develop and test a custom authentication module, you must implement it in Hyperion environment.

Overview of Steps

To implement the custom authentication module, complete the following steps:

- Stop Hyperion products including Shared Services and any systems that use Shared Services APIs.
- Copy custom authentication module to the classpath. See "Deploying the Custom Authentication Module" on page 13.
- Deploying Shared Services to the application server
- Update user directory configurations. See:
  - "Enabling Custom Authentication Module" on page 16
  - "Disabling Custom Authentication for a Specific External User Directory" on page 17
  - "Enabling Custom Authentication for Native Directory" on page 18

- Complete the deployment of the custom authentication module for Hyperion products that depend on Shared Services to manage security.
- Start Hyperion products and processes.

Prerequisites

- A fully tested Java archive named `CustomAuth.jar`, that contains custom authentication module libraries. `CustomAuth.jar` must implement the public interface `CSSCustomAuthenticationIF`, defined in `com.hyperion.css` package as a part of the standard Shared Services APIs.
- Access to Shared Services as Shared Services administrator

Deploying the Custom Authentication Module

To deploy the custom authentication module, you must add the custom authentication module Java archive (`CustomAuth.jar`) to the classpath of Shared Services. You must also add the archive to the classpath of Hyperion products that depend on Shared Services for managing security. Add the custom authentication module Java archive in the following locations:

- A common location on each non-J2EE Hyperion product host machines. See "Adding Custom Authentication Module to Common Location on Hyperion Host Machines" on page 14.
- Classpath of Hyperion J2EE components.

  For non-J2EE applications, perform product-specific procedures. See:

  - "Adding a Custom Module to the Reporting and Analysis Core Services Classpath" on page 14
  - "Adding a Custom Module to the Essbase Classpath" on page 15
  - "Adding a Custom Module to the Financial Management Classpath" on page 15
- Configure Shared Services to use the custom authentication module. See "Shared Services Procedures " on page 16

Note: After deploying a custom authentication module, review "Documentation Update: Hardening System 9 Security" in the *Hyperion Shared Services Release 9.3.3.0.00 Readme* for information on securing Hyperion products, including the procedure to regenerate SSO Encryption Key.

To view the most recent version of this Readme, see the 9.3.x documentation library on Oracle Technology Network (OTN).

Adding Custom Authentication Module to Common Location on Hyperion Host Machines

Add the custom module Java archive (`CustomAuth.jar`) to *`HYPERION_HOME`*`/common/CSS/`*`VERSION_NUM/`*`lib` on each machine that hosts a non_J2EE Hyperion product. Generally, this common location is identified in the application classpath.

➤ To add the custom authentication module to the common location:

1 **Stop all Hyperion products.**

2 **On each Hyperion product host machine, copy the custom authentication module Java archive into** *`HYPERION_HOME`*`/common/CSS/`*`VERSION_NUM`*`/lib`**.**

Adding Custom Authentication Module to J2EE Application Classpath

Use this section to add the custom authentication module Java archive to the classpath of System 9 J2EE Web applications. The following are not J2EE Web applications.

- Oracle's Hyperion® Essbase® – System 9
- Oracle's Hyperion® Reporting and Analysis – System 9 Core Services
- Oracle's Hyperion® Financial Management – System 9

Note: After adding a custom module to the classpath, you must perform additional Shared Services procedures to make custom authentication operational. See "Shared Services Procedures " on page 16.

➤ To add a custom authentication module to the classpath for Hyperion J2EE applications:

1 **Ensure that Hyperion products are not running.**

2 **On the server that hosts Hyperion application, copy custom authentication Java archive into the classpath. The classpath is generally** *`HYPERION_HOME`*`/deployments/`*`APP_SERVER/ PRODUCT_CODE`*`/webapps/`*`APP_CONTEXT`*`/WEB-INF/lib`**.**

For example, the classpath of Shared Services deployed on Tomcat 5 is *`HYPERION_HOME/`*`deployments/`*`Tomcat5`*`/SharedServices9/webapps/interop/WEB-INF/lib`.

Adding a Custom Module to the Reporting and Analysis Core Services Classpath

Make sure that custom authentication module is copied into the Workspace classpath. See "Deploying the Custom Authentication Module" on page 13.

➤ To enable Reporting and Analysis Core Services custom authentication:

1 **Make sure that Workspace and Reporting and Analysis Core Services are shut down.**

2 **Make sure that the custom authentication module Java archive is available in *`HYPERION_HOME/`*`common/CSS/9.3.1/lib` on the machine where Oracle's Hyperion® Reporting and Analysis – System 9 Core Services is deployed.**

3 **Add the custom authentication Java archive to the classpath.**

   a. Using a text editor, open *`HYPERION_HOME`*`/common/workspacert/9.3.1/bin/workspace.app`.

   b. Add an entry such as the following for the custom authentication Java archive:

      `${home}/common/CSS/9.3.1/lib/CustAuth.jar`, where `CustAuth.jar` indicates the name of the custom authentication module Java archive.

   c. Save and close `workspace.app`.

Adding a Custom Module to the Essbase Classpath

To enable custom authentication, you must add the custom authentication class files to the Oracle's Hyperion® Shared Services Java archive (`css-9_3_1.jar`) that Essbase uses.

➤ To add a custom module to the Essbase classpath:

1 **Make sure that Essbase Server is shut down.**

2 **On the machine that hosts Oracle's Hyperion® Essbase® – System 9 Server, add the custom authentication module to `css-9_3_1.jar` .**

   a. Create a backup copy of *`HYPERION_HOME`*`/common/CSS/9.3.1/lib/css-9_3_1.jar`

   b. Using the `jar` command, extract the contents of *`HYPERION_HOME`*`/common/CSS/9.3.1/lib/css-9_3_1.jar` into a temporary directory; for example, `temp1`.

   c. Copy the custom authentication module class files into the temporary directory where you extracted the contents of `css-9_3_1.jar`.

      Note: Be sure to copy the class files into the directory appropriate for your package structure. If you used *`com.yourcompany.customauth`* package, make sure that your class files are copied to `com/yourcompany/customauth` within the temporary directory.

   d. Using the `jar` command, recreate `css-9_3_1.jar` using the contents of the temporary directory. Make sure that the directory structure is maintained in `css-9_3_1.jar`.

3 **Copy `css-9_3_1.jar` from the preceding step to *`HYPERION_HOME`*`/common/CSS/9.3.1/lib`, replacing the existing file.**

Adding a Custom Module to the Financial Management Classpath

To add a custom module to the Financial Management classpath, you must update the Windows registry with the location and name of the of the custom module.

➤ To add a custom module to the Financial Management classpath:

1. **Stop all Financial Management processes, especially** `CASSecurity.exe`**.**
2. **Make sure that the custom authentication module Java archive is available in** *`HYPERION_HOME/`*`common/CSS/9.3.1/lib`**.**
3. **In Windows registry, append the location of the custom module Java archive; for example,** `%HYPERION_HOME%\COMMON\css\9.3.1\lib\customAuth.jar` **to the Oracle's Hyperion® Financial Management – System 9 value data of the following key. Be sure to use a semicolon (;) to separate the existing value data from the path that you append.**

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\Hyperion Solutions\Hyperion Financial
   Management\Server\Authentication\ClassPath
   ```

Shared Services Procedures

Subtopics

- Enabling Custom Authentication Module
- Disabling Custom Authentication for a Specific External User Directory
- Enabling Custom Authentication for Native Directory

Enabling Custom Authentication Module

By default, custom authentication is not enabled in Hyperion. After you enable its use, Hyperion security automatically picks up the custom authentication module if its class file uses the expected default class name (`CustomAuthenticationImpl`) and package (`com.hyperion.css.custom`) structure. The name of the Java archive that contains the class file is not relevant (suggested name is `customAuth.jar`). In this scenario, you need not specify the custom authentication class name as described in this discussion.

If you use a custom class name or package structure in the Java archive, you must enter the fully qualified Java class name of the custom authentication class in `CSS.xml`.

Note: By default, all configured user directories other than Native Directory are enabled to use the custom authentication module, if available.

See “Sample `CSS.xml`” on page 19 for sample `CSS.xml`

➤ To enable custom authentication module:

1. **Using a text editor, open** *`HYPERION_HOME`*`/deployments/APP_SERVER/SharedServices9/config/CSS.xml`**.**
2. **Optional: If you are using a custom class name or package structure, create an entry similar to the following. This entry must be outside the** `<spi>` **definition.**

   ```
   <customModules>
        <authenticateModule>FQCN of Custom Authentication class
   ```

```
        </authenticateModule>
    </customModules>
```

Note: The expected value in the `<authenticateModule>` tag is the fully qualified class name of the custom authentication class that implements the CSSCustomAuthenticationIF interface. For example, `com.mycompany.custom.MyCustomAuthentication`.

3 **Save `CSS.xml`.**

Disabling Custom Authentication for a Specific External User Directory

By default, all the providers except Native Directory are enabled to use the custom authentication module, if available. If custom authentication is to be disabled for any of the configured external user directories, you must make the necessary configuration changes by editing `CSS.xml`. See "Sample `CSS.xml`" on page 19 for sample `CSS.xml`.

➤ To disable custom authentication for an external user directory:

1 **Using a text editor, open *`HYPERION_HOME`*`/deployments/APP_SERVER/SharedServices9/config/CSS.xml`.**

2 **Include the following tag inside the definition of each external user directory for which custom authentication is to be turned off.**

```
<authenticationModuleEnabled>false</authenticationModuleEnabled>
```

For example, your user directory definition may be as follows after adding this tag:

```
<ldap name="LDAP1">
      <vendor>Sun One LDAP</vendor>
      <trusted>true</trusted>
      <url>ldap://myServer:4816/dc=hyperion,dc=com</url>
      <authType>simple</authType>
      <maxSize>0</maxSize>
      <identityAttribute>nsuniqueid</identityAttribute>
      <identityAttributeType>String</identityAttributeType>
      <user>
           <url>ou=People</url>
           <loginAttribute>uid</loginAttribute>
           <fnAttribute>givenName</fnAttribute>
           <snAttribute>sn</snAttribute>
           <emailAttribute>mail</emailAttribute>
          <objectclass>
               <entry>person</entry>
               <entry>organizationalPerson</entry>
               <entry>inetorgperson</entry>
          </objectclass>
      </user>
      <group>
           <useGroups>true</useGroups>
          <url />
      </group>
      <authenticationModuleEnabled>false</authenticationModuleEnabled>
  </ldap>
```

3 Save `CSS.xml`.

Enabling Custom Authentication for Native Directory

You can enable custom authentication for Native Directory by updating `CSS.xml`. See "Sample `CSS.xml`" on page 19 for sample `CSS.xml`

➤ To enable custom authentication for Native Directory

1 **Using a text editor, open** *`HYPERION_HOME`*`/deployments/APP_SERVER/SharedServices9/config/CSS.xml`.

2 **Include the following tag inside the Native Directory definition.**

```
<authenticationModuleEnabled>true</authenticationModuleEnabled>
```

For example, your Native Directory definition may be as follows after adding this tag:

```
<native name="Native Directory">
     <password>{CSS}xxxXXXXcgiE/dGr8rFdvQLcA==</password>
     <authenticationModuleEnabled>true</authenticationModuleEnabled>
</native>
```

3 **Make sure that custom authentication is enabled. See "Enabling Custom Authentication Module" on page 16**

4 Save `CSS.xml`.

Testing Your Deployment

If Native Directory is not configured for custom authentication, do not use Native Directory users to test custom authentication.

Note: It is your responsibility to identify and correct any issues with the custom authentication module. Oracle assumes that your custom module works flawlessly to map a user from the user directory used by the custom module to a user on a custom authentication-enabled user directory available in Hyperion search order.

To test your deployment, log into Hyperion using user credentials from the user directory; for example, an RSA SecurID infrastructure, used by the custom module. These credentials may be different from the Hyperion credentials.

Your implementation can be considered successful if Hyperion products allow you to access their resources. An error indicating that the user was not found is not always an indicator of an unsuccessful implementation. In such cases, verify that credentials you entered are present in the custom user store and a matching user is present in one of the custom authentication-enabled user directories in Hyperion search order.

➤ To test custom authentication:

1 **Make sure that Hyperion products are running.**

2 **Launch Oracle's Hyperion® Shared Services User Management Console or Oracle's Hyperion® Workspace.**

3 **Log in as a user defined on the user directory, used by the custom authentication module.**

   a. In **Username**, enter your user identifier; for example, an RSA User ID.

   b. In **Password**, enter a password; for example, an RSA PIN.

   c. Click **Login**.

4 **Verify that you can access Hyperion product resources.**

Sample `CSS.xml`

In the following sample `CSS.xml`:

- Native Directory is enabled to use custom authentication.
- External user directory `LDAP1` is not enabled to use custom authentication
- External user directory `MSAD1` is configured to support custom authentication

```
<?xml version="1.0" encoding="UTF-8"
<css>
    <hub location="http://myServer.hyperion.com:58080">
         <dirPort>58089</dirPort>
    </hub>
    <spi>
         <provider>

             <native name="Native Directory">
                  <password>{CSS}4N6lVcgiE/dGr8rFdvQLcA==</password>
                  <authenticationModuleEnabled>false</authenticationModuleEnabled>
            </native>

         <ldap name="LDAP1">
             <vendor>Sun One LDAP</vendor>
             <trusted>true</trusted>
             <url>ldap://myServer:4816/dc=hyperion,dc=com</url>
             <authType>simple</authType>
             <maxSize>0</maxSize>
             <identityAttribute>nsuniqueid</identityAttribute>
             <identityAttributeType>String</identityAttributeType>
             <user>
                  <url>ou=People</url>
                  <loginAttribute>uid</loginAttribute>
                  <fnAttribute>givenName</fnAttribute>
                  <snAttribute>sn</snAttribute>
                  <emailAttribute>mail</emailAttribute>
                 <objectclass>
                      <entry>person</entry>
                      <entry>organizationalPerson</entry>
                      <entry>inetorgperson</entry>
                 </objectclass>
             </user>
             <group>
                  <useGroups>true</useGroups>
```

```
                <url />
            </group>
            <authenticationModuleEnabled>false</authenticationModuleEnabled>
        </ldap>

        <msad name="MSAD1">
             <vendor>Microsoft</vendor>
             <trusted>true</trusted>
             <url>ldap://myServer-3:389/DC=ctgdev03,DC=hyperion,DC=com</url>
             <userDN>cn=Administrator,CN=Users,DC=ctgdev03,DC=hyperion,DC=com</userDN>
             <password>{CSS}Gvhy1I2E/N6wF7kt6Jzq3Q==</password>
             <authType>simple</authType>
             <maxSize>0</maxSize>
             <identityAttribute>ObjectGUID</identityAttribute>
             <identityAttributeType>Octet String</identityAttributeType>
             <user>
                 <url>OU=Performance</url>
                  <loginAttribute>cn</loginAttribute>
                  <fnAttribute>givenName</fnAttribute>
                  <snAttribute>sn</snAttribute>
                  <objectclass>
                      <entry>person</entry>
                      <entry>organizationalPerson</entry>
                      <entry>user</entry>
                  </objectclass>
             </user>
             <group>
                  <useGroups>true</useGroups>
                  <url />
             </group>
             </msad>

        </provider>
    </spi>
    <searchOrder>
         <el>Native Directory</el>
         <el>LDAP1</el>
         <el>MSAD1</el>
    </searchOrder>
    <token>
         <timeout>480</timeout>
    </token>
    <logger>
         <priority>WARN</priority>
    </logger>
    <customModules>
         <authenticateModule>com.hyperion.css.security.CustomAuthenticationImpl
</authenticateModule>
    </customModules>
</css>
```

Tips and Tricks

After the authentication steps, Hyperion security checks the retrieved user name in a custom authentication-enabled user directory included in the search order before verifying user roles.

If you have configured the same user directory twice, one as custom authentication-enabled and the other as custom authentication-disabled, ensure that you provision the user from the right provider depending on whether the user logs in with an LDAP password or custom authentication credentials.

ORACLE
ENTERPRISE PERFORMANCE
MANAGEMENT SYSTEM