

---

---



# Oracle Knowledge iAuthenticator API Integration Guide

---

*Integrating Oracle Knowledge Applications with Enterprise Security*

---

Release 8.6  
Document Number OKIM-IAI86-00  
March 2015

---

## COPYRIGHT INFORMATION

Copyright © 2002, 2015, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS:

Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

### Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

### Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

---

# Contents

---

## About This Guide 1

In This Guide .....	1
Examples of Product Screens and Text .....	1
Operating System Variations in Examples and Procedures .....	1
References to Web Content .....	2
.....	2

3

## Introduction to iAuthenticator 3

Standard Authentication .....	3
LDAP Authentication .....	3
SSO Authentication .....	4

5

## Using iAuthenticator 5

Connecting Internal and External User Management Systems .....	5
Coordinating the Authentication Process .....	5
Implementation Overview .....	6
iAuthenticator Interfaces and Components .....	8
iUser Interface Methods .....	9
iRoleBasedUser Interface Method .....	11
RoleBasedUser Subclass .....	11
iFieldBuilder for Oracle CRM OnDemand .....	12
IUpdatableUserAuthenticator Interface Methods .....	12

14

## Implementing iAuthenticator 14

Configuring Information Manager .....	14
---------------------------------------	----

16

## Error Conditions 16

Error Condition Solutions .....	16
---------------------------------	----

**iAuthentication Sample 17**

Implementation Sample Code ..... 17

**Implementing IUpdatableUserAuthenticator 20**

Configuring IUpdatableUserAuthenticator ..... 20  
IUpdatableUserAuthenticator Sample Code ..... 21

**API Reference 25**

Package Names of Classes ..... 25  
Public API Classes and Methods ..... 25

# About This Guide

This guide describes how to use the Oracle Knowledge iAuthenticator API (iAuthenticator) to integrate Oracle Knowledge applications with existing enterprise security. This guide is intended for application developers who implement the iAuthenticator API and protocol in pluggable modules, and for systems administrators who configure the use of the modules.

## In This Guide

This guide is divided into the following sections:

<b>“Introduction to iAuthenticator”</b>	Describes how to use the iAuthenticator API to develop custom authentication solutions that meet various business requirements.
<b>“Using iAuthenticator”</b>	Describes how to use iAuthenticator API for external authentication and lists the major interfaces and their methods and classes.
<b>“Implementing iAuthenticator”</b>	Explains how to implement iAuthenticator to one or more installations across physical or virtual machines.
<b>“Error Conditions”</b>	Lists the error conditions that you may encounter during the implementation of a custom authenticator.
<b>“iAuthentication Sample”</b>	Demonstrates an example of iAuthenticator that validates whether or not a user ID and password match the corresponding values in the database.
<b>“Implementing IUpdatableUserAuthenticator”</b>	Describes how to use IUpdatableUserAuthenticator API and lists its methods. It also includes a simple sample of an implementation.
<b>“API Reference”</b>	Lists the full package names of classes mentioned in this guide.

## Examples of Product Screens and Text

The product screens, screen text, and file contents depicted in the documentation are examples. We attempt to convey the product's appearance and functionality as accurately as possible; however, the actual product contents and displays may differ from the published examples.

## Operating System Variations in Examples and Procedures

We generally use Linux screen displays and naming conventions in our examples and procedures. We include other operating system-specific procedures or steps as noted in section headings, or within topics, as appropriate.

We present command syntax, program output, and screen displays:

- in Linux format first
- in other Unix-specific variants only when necessary for proper operation or to clarify functional differences
- in Windows format only when necessary for clarity

## References to Web Content

For your convenience, this guide refers to Uniform Resource Locators (URLs) for resources published on the World Wide Web, when appropriate. We attempt to provide accurate information; however, these resources are controlled by their respective owners and are therefore subject to change at any time.

---

# Introduction to iAuthenticator

The Oracle Knowledge iAuthenticator API (iAuthenticator) enables integration of Oracle Knowledge applications with existing enterprise security realms. You use the iAuthenticator API to separate user authentication processes from the Oracle Knowledge applications, and customize user authentication processes without changing the Oracle Knowledge application code.

You can use the iAuthenticator API to develop custom authentication solutions to meet various business requirements. For example, you can use iAuthenticator to implement authentication for employees using one security system, and authentication for customers using a separate system.

You implement the API to bridge two security models: your organization's and the Information Manager security model. Your organization makes the decisions for user role access. For example, the information repository configuration defines user roles and privileges, so that you could create agent roles with privileges that customer roles do not have.

With Information Manager, you can implement user authentication in two ways:

- Oracle Knowledge standard authentication
- Oracle Knowledge iAuthenticator API (iAuthenticator)

The standard authentication interface is configured by default as described in *Oracle Knowledge Information Manager Administration Guide*. It also integrates with external Lightweight Directory Access Protocol (LDAP) and Single Sign-On (SSO) systems as described in the “LDAP Authentication” and “SSO Authentication” sections, respectively.

## Standard Authentication

Oracle Knowledge has a built-in security realm that is stored in the Information Manager database. This realm uses the Oracle Knowledge standard authentication interface, and is entirely internal to Oracle Knowledge. You implement standard authentication by defining and authorizing application users by using the Management Console in the Information Manager repository. Oracle Knowledge authorizes users when they log in based only on the internally stored user information. Oracle Knowledge is not integrated with any external security systems.

## LDAP Authentication

You can configure the standard authentication interface to integrate Oracle Knowledge with an existing LDAP security realm. Information Manager has built-in LDAP authentication that you can configure to integrate with an external realm that is stored in an LDAP server and can be configured to map users to roles and views. LDAP Authentication is available for all Oracle Knowledge applications. The configuration is available through the Information Manager Management Console application.

When you configure built-in LDAP integration, the external (customer-supplied) LDAP directory stores the user authentication information, including the list of authorized views and roles assigned to each user. You configure the list of available roles and views within Information Manager.

When you use LDAP authentication, you perform all user-management tasks within the external LDAP system, not within Information Manager. Oracle Knowledge's built-in LDAP authentication requires that you store all user information within a single LDAP branch. You can store roles and views in a separate branch of the LDAP hierarchy, but, you must store all instances of each type of security object in a single branch. Information Manager authorizes users based on the security roles that the LDAP system assigns them.

For more information on configuring LDAP, see the "LDAP Configuration" section of the *Oracle Knowledge Information Manager Administrator's Guide*.

## SSO Authentication

To configure the standard authentication interface to integrate with an SSO provider, you configure the provider to work with an LDAP directory so that when the SSO provider authenticates each user, basic information, such as the userid, is stored in the HTTP Header. The information stored in the HTTP header validates the user against the LDAP directory, similar to the LDAP authentication process.

"Using iAuthenticator", in this guide, describes how to implement iAuthenticator with external security systems.



# Using iAuthenticator

This section describes how to use the iAuthenticator API for external authentication and lists the major interfaces and their methods and classes. It also includes a simple example of an implementation.

## Connecting Internal and External User Management Systems

The first time a user successfully signs in to iAuthenticator, the Information Manager creates a local copy of each user, and the external system stores the master copy of each user. The local copy ties an internal user management system with the iAuthenticator's external management system. The local copy is not the true user record. The true user record is stored in your organization's user management system.

You can modify the local profile copy by using the Management Console, which you can use to update fields that are not available in the Oracle Knowledge iAuthenticator module. These include such fields as locale, avatar, and subscription options. On each subsequent user login, iAuthenticator overwrites the fields that it manages, which typically include first name, last name, e-mail address, and roles. You can use this capability to store and control sensitive user data in an external realm without requiring you to secure the entire Information Manager database in the same external realm.

Every time a user logs in, the external user management system updates and overwrites the local copy, which is subordinate to the external management system.

## Coordinating the Authentication Process

The InquireaAuthenticator module coordinates the entire custom authentication process. The InquireaAuthenticator is the module that delegates to a custom iAuthenticator in the Information Manager IMTagLibrary and ClientLibrary applications by calling the custom authenticator to bridge authentication models.

This module calls the `iAuthenticator.authenticate` method. It is not part of the externally visible API.

If the custom authentication is successful, the InquireaAuthenticator sets the following fields for the local copy user in Information Manager.

Field	Description
<b>loginID</b>	Login set on the <code>RoleBasedUser</code> returned by the <code>authenticate</code> method.
<b>repository</b>	Known at runtime.
<b>active flag</b>	Field is always set to <code>true</code> .
<b>emailAddress</b>	E-mail as set on the <code>RoleBasedUser</code> returned by the <code>authenticate</code> method.
<b>admin flag</b>	Field is set to <code>true</code> if the user has one or more Information Manager Management Console roles.
<b>firstname</b>	The user's first name as set on the <code>RoleBasedUser</code> returned by the <code>authenticate</code> method.

<b>lastname</b>	The user's last name as set on the <code>RoleBasedUser</code> returned by the <code>authenticate</code> method.
<b>password</b>	<ul style="list-style-type: none"> <li>• If the <code>REMOTE_AUTHENTICATION_STORE_PASSWORD_LOCALLY</code> configuration parameter is set to <code>TRUE</code>, the supplied password is stored locally in the Information Manager database.</li> <li>• If the <code>REMOTE_AUTHENTICATION_STORE_PASSWORD_LOCALLY</code> configuration parameter is set to <code>FALSE</code> (the default), and the user being authenticated does not currently have a password stored in the Information Manager database, the <code>InqiraAuthenticator</code> will generate a random password for the user set to a random string (GUID) and store it in the Information Manager Database.</li> <li>• If the <code>ALLOW_PASSWORD_EDITING</code> configuration parameter is set to <code>TRUE</code>, the user's password can be set when creating a new user, and changed for existing users in the Information Manager Management Console.</li> <li>• If <code>ALLOW_PASSWORD_EDITING</code> is set to <code>FALSE</code>, the user's password will not be set when creating a new user, and existing users' passwords cannot be changed in the Information Manager Management Console.)</li> </ul>
<b>locale</b>	Defaults to the repository's default locale. It is only set on the initial user creation.
<b>reportingUserGroup</b>	Can be set from external LDAP realm. The <code>roles2PermissionsMap</code> also contains All <code>UserGroups</code> in repository.
<b>roles</b>	User roles as set on the <code>RoleBasedUser</code> returned by the <code>authenticate</code> method.
<b>views</b>	User views as set on the <code>RoleBasedUser</code> returned by the <code>authenticate</code> method.

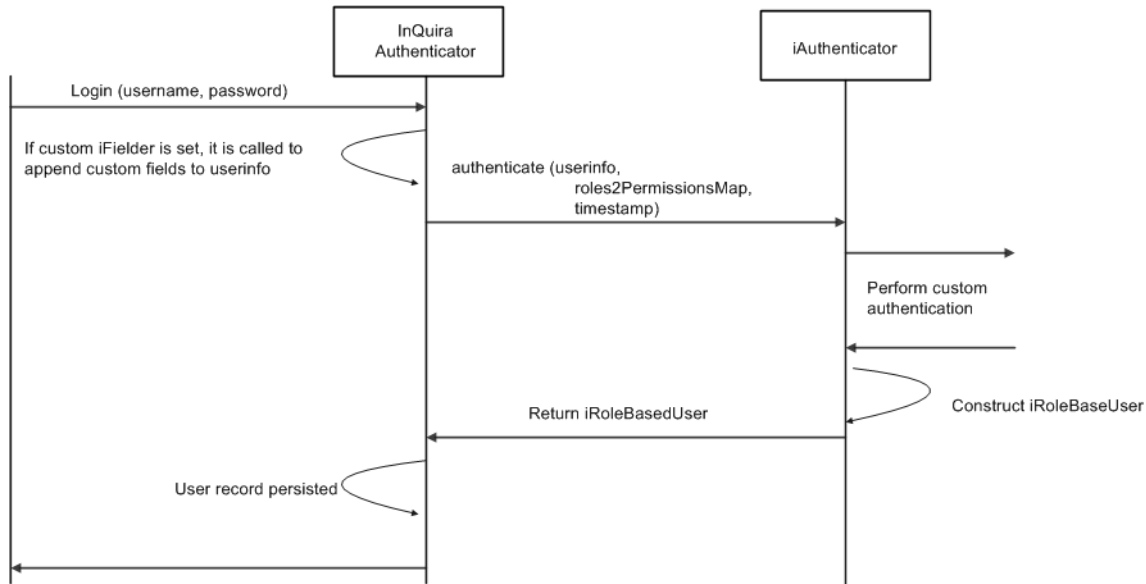
**Note:** `InqiraAuthenticator` uses no other fields from `RoleBasedUser`.

The `InqiraAuthenticator` stores the user's values in the Information Manager database. If the user does not exist in the database, the data in the fields is added; if the user exists, the fields are updated.

The Information Manager database is only updated when the user's values are returned from the `authenticate` method or are updated manually through the Information Manager Management Console. However, if the user data is modified using the Information Manager Management Console, the values in fields mentioned are overwritten the next time the user logs in.

## Implementation Overview

The `iAuthenticator` framework includes an API and an implementation process. Developers implement the `iAuthenticator` API and the implementation protocol in pluggable modules, and system administrators can then configure these modules for use within an application. The following use case diagram shows the interaction of components for a typical login.



- a The user name and password are passed to the InQuiraAuthenticator module.
- b The InQuiraAuthenticator delegates the user login information and passes it to the iAuthenticator.
- c iAuthenticator performs the custom authentication and constructs the iRolebasedUser interface.
- d iAuthenticator returns iRolebasedUser to the InQuiraAuthenticator.
- e The user record is added using a persist tool.

Because you must call it to authenticate the user, the most important method is `authenticate(FieldValue[] userInfo, Map roles2PermissionsMap, long timestamp)`. The return value is the `iUser` mentioned above. However, Information Manager expects the value to be the subclass `iRoleBasedUser` because Information Manager and Intelligent Search share the same authentication interface. Refer to *Oracle Knowledge Intelligent Search Application Development Guide* for information on `iAuthenticator` as it relates to the standalone Intelligent Search product and creating a basic custom authenticator built on the `iAuthenticator` interface.

`iRoleBasedUser` must contain all Roles and Views assigned to the user. During the `authenticate` method, if an exception condition occurs that cannot be corrected satisfactorily, you must throw an `InQuiraAuthenticationException` with a description of the problem so that it can be logged properly to the Information Manager runtime log at the `$INQUIRA_ROOT/instances/<company>/InQuira-InfoManager-infomanager.log`.

```

public interface IAuthenticator {

    public IUser authenticate(FieldValue[] userInfo,
                             Map roles2PermissionsMap,
                             longtime timestamp ) throws InQuiraAuthenticationException;
}
  
```

Method	Description
<b>FieldValue[] userInfo</b>	<p>The FieldValue is a name-value pair. This array contains the name-value pairs being used to authenticate the user. It also implements the iFieldNames interface. This interface defines all requisite as well as common field names.</p> <p>For example, this array always contains:</p> <ul style="list-style-type: none"> <li>• username FieldValue (name = IFieldNames.FIELD_USER_ID,)</li> <li>• password FieldValue (name = IFieldNames.FIELD_PASSWORD), and</li> <li>• domain/repository FieldValue (name = IFieldNames.FIELD_DOMAIN).</li> </ul> <p>This array can hold other FieldValues obtained from the request header or the application as well as from a custom iFieldBuilder. These are extra parameters that you may need to pass to your own implementation to perform authentication.</p>
<b>Map roles2PermissionsMap</b>	<p>Contains all the role, view, and usergroup reference keys in the repository. The iAuthenticator API passes this map into an implemented authenticate method, allowing implementers a reference in case they need to map from an external system to Information Manager's internal role, view, or usergroup.</p> <p>Implementers of the Oracle Knowledge iAuthenticator APIs authenticate method must instantiate and return an IRoleBasedUser as part of the API. If that user contains Information Manager roles, view, or reportingUsergroup, they must be a subset of what is in this map. The content of this map is configured in Information Manager.</p> <p>To distinguish these keys from each other in this multi-purpose map:</p> <ul style="list-style-type: none"> <li>• All role reference keys are prefixed with R.</li> <li>• All view reference keys are prefixed with V.</li> <li>• All usergroup reference keys are prefixed with G.</li> </ul>
<b>longtime timestamp</b>	<p>The time when the authenticate call is requested. This allows you to make authentication decisions based on time, or to track the time. In addition, many API calls require <code>longtime timestamp</code>, for example, when instantiating a RoleBasedUser to return.</p>

Perform any steps you need to authenticate the user and return an iRoleBasedUser. Typically, a custom authenticator communicates with one or more external realms to authenticate the user and then copies or maps views and roles to the user using the iRoleBasedUser. A null return value results in the user being denied access to the system.

The InquireAuthenticator takes the returned user information and adds or updates the values in the Information Manager database.

## iAuthenticator Interfaces and Components

The iUser and iRoleBasedUser interfaces define an interface hierarchy that provides the minimum amount of information a user needs to have in the Oracle Knowledge system for authentication and authorization purposes. This information typically includes the user name, user ID, e-mail address, and so on.

The IUpdatableUserAuthenticator interface provides the ability to either update the user information in the Information Manager database, or not update it, when authenticating the user with Information Manager. In addition, it provides the ability whether to enforce password life cycle that you configure in Information Manager.

## iUser Interface Methods

The following table lists iUser interface methods and whether or not Information Manager iAuthenticator uses them.

Method	Description	Used by Information Manager iAuthenticator?
getUserId	Returns the user's ID, for example, jdoe.	Yes
getDisplayName	Returns the user's full name, for example, John Doe.	Yes
getEmailAddress	Returns the user's e-mail address.	Yes
getLoginId	Returns the combination of the domain and user ID.	Yes
setAttribute(UserAttribute id, String value)	Sets the value for the given UserAttribute. <b>Note:</b> The UserAttribute class provided by Oracle Knowledge pre-defines only three user attributes: FIRST_NAME, MIDDLE_INITIAL, and LAST_NAME. The InquireAuthenticator uses these three attributes when constructing the local copy user.	Yes
getAttribute(UserAttribute id)	Returns the value for the given UserAttribute. <b>Note:</b> The InquireAuthenticator uses the FIRST_NAME, MIDDLE_INITIAL, and LAST_NAME pre-defined user attributes when constructing the local copy user.	Yes
getActive	Returns the boolean indicating whether the user is active or inactive.	Yes
setActive(Boolean)	Sets the user active or inactive. Default is active.	Yes
getAlias	Returns the user's alias name.	Yes
setAlias(String value)	Sets the user's alias name.	Yes
getReceiveAssigned	Returns the boolean indicating whether email notification is enabled for tasks.	Yes
setReceiveAssigned(Boolean receiveAssigned)	Sets the email notification for tasks. Default is disabled.	Yes
getReceivePerform	Returns the boolean value indicating whether email notification is enabled for tasks assigned to the user.	Yes
setReceivePerform(Boolean receivePerform)	Sets the email notification for tasks assigned to the user. Default is disabled.	Yes
getSubscribeOnTopicCreation	Returns the boolean value indicating whether the user has enabled subscription to topics that user has created.	Yes
setSubscribeOnTopicCreation(Boolean subscribeOnTopicCreation)	Set the boolean value to enable or disable subscription to topics that user has created. Default is disabled.	Yes
getSubscribeOnTopicReply	Returns the boolean value indicating whether the user has enabled subscription to topics that user has replied to.	Yes
setSubscribeOnTopicReply(Boolean subscribeOnTopicReply)	Set the boolean value to enable or disable the subscription to topics that user has replied to. Default is disabled.	Yes
getSubscriptionSchedule	Returns the value for subscription notifications schedule.	Yes

Method	Description	Used by Information Manager iAuthenticator?
setSubscriptionSchedule(Num ber subscriptionSchedule)	Sets the value for subscription notifications schedule. The valid values are: <ul style="list-style-type: none"> <li>• 0 - Don't send emails</li> <li>• 1 - Immediately (Default)</li> <li>• 2 - Once per day</li> <li>• 3 - Every other day</li> <li>• 4 - Once per week</li> </ul>	Yes
getAssignedCategories	Returns the set of Category Reference Key strings assigned to the user.	Yes
setAssignedCategories(Set<St ring> assignedCategories)	Sets the Category Reference Key strings assigned to the user. <b>Note:</b> The method ignores the invalid reference key strings that you pass and retains the previous category strings.	Yes
getDefaultSubsite	Returns the default view of the user.	Yes
setDefaultSubsite(String defaultSubsite)	Sets the default view of the user. <b>Note:</b> The method ignores the invalid view you specify and retains the previous default view.	Yes
getKeyValues	Returns the set of User Key Values for the user.	Yes
setKeyValues(Set<com.inquir .infra.UserKeyValue> keyValues)	Sets the User Key Values for the user.	Yes
getDefaultLocale	Returns the default locale of the user.	Yes
setDefaultLocale(java.util.Loc ale locale)	Sets the default locale for the user. You must create an instance of locale by passing the language and country code. For example, new java.util.Locale("en","US"). <b>Note:</b> The method ignores the invalid locale you specify and retains the previous default locale.	Yes
getContentLocales	Returns the set of Content Locales of the user.	Yes
setContentLocales(Set<java.ut il.Locale> locales)	Sets the set of Content Locales for the user. You must create an instance of locale by passing the language and country code. For example, java.util.Locale("en","US"). <b>Note:</b> The method ignores the invalid content locale you specify and retains the previous default locale.	Yes
getSubject	Returns the new Subject object with the user's Principal added as the only known Principal.	No
getPrincipal	Returns the user's Principal.	No
getDomain	Returns the user's domain.	No
getTimestamp	Returns the time this user was created or logged-in.	No
isAnonymousUser	Checks whether the given user object represents the anonymous user.	No
hasPasswordExpired	Returns whether the user's password has expired.	No

Method	Description	Used by Information Manager iAuthenticator?
getDaysTillPasswordExpires	Gets the number of days until the user's password expires.	No
setDaysTillPasswordExpires (int daysTillPasswordExpires)	Sets the number of days until the user's password expires.	No
hasAccess(Permission perm)	Returns whether the user has access rights to the given permission.	No
getSecurityPermissions	Returns the set of all security permissions this user can access.	No
getSecurityKeys	Returns the set of keys for all security permissions this user can access.	No
dump	Dumps the state of the user object to stdout for debugging.	No

## iRoleBasedUser Interface Method

The iRoleBasedUser interface consists of a Subject (for example, the user), the Subject's credentials (for example, the roles and views), and the Subject's principals (for example, the UserID). This interface extends the iUser interface. The following table describes the method for iRoleBasedUser.

Method for iRoleBasedUser	Description	Used by Information Manager iAuthenticator?
getActiveRoles	Returns the user's active roles as stored on the remote system.	Yes

## RoleBasedUser Subclass

An instance of iRoleBasedUser must be the return value of the iAuthenticator.authenticate() method described in the following example. If you need to support custom fields, you can implement the iRoleBasedUser interface, subclass RoleBasedUser or use RoleBasedUser directly.

Use the following method and parameters to construct a RoleBasedUser:

```
public class RoleBasedUser extends AbstractUser implements IRoleBasedUser,
IAASLogConstants {

    public RoleBasedUser(String userId,
                        String domain,
                        String displayName,
                        Principal principal,
                        Set roles,
                        Permissions permissions,
                        long timestamp);
}
```

Method	Description
<b>String userId</b>	The subject's User Name. It must be unique in the Information Manager repository.
<b>String domain</b>	The domain of the user. It is also called the repository reference key.
<b>String displayName</b>	The name of the user, formatted as <i>firstname lastname</i> .

Method	Description
<b>Principal principal</b>	Although the Information Manager implementation of Oracle Knowledge Authenticator does not use this field, it is required to instantiate a RoleBasedUser object which is the expected return type of the authenticate method. To satisfy this requirement, you can use the SimplePrincipal object provided by Oracle Knowledge as shown in the sample code in "iAuthentication Sample".
<b>Set roles</b>	<p>A Set&lt;String&gt; that contains all Roles and Views for a user. Any values you assign must match those found in iAuthenticator roles2PermissionsMap.</p> <p>Implementers can add the reportingUserGroup field to this set. If more than one, the first one is used. The reportingUserGroup must be one of the UserGroups in roles2PermissionMap prefixed with G.</p> <p>This set must contain at least one valid role and view for the user.</p>
<b>Permissions permission</b>	<p>Although this value is not used in Information Managed-based implementations, it cannot be a null value. To satisfy this requirement, create a Permission Set and add Oracle Knowledge's StandardPermission object to it.</p> <p><b>Note:</b> Oracle Knowledge StandardPermission is a concrete implementation of the Java abstract BasicPermission class.</p>
<b>Long timestamp</b>	The user value passed into iAuthenticator constructor.

## iFieldBuilder for Oracle CRM OnDemand

The `iAuthenticator.authenticate` method parameter `userInfo` array always contains the username, password, and domain FieldValues.

iFieldBuilder is a mechanism to pass additional attributes to the remote authenticator. The iFieldBuilder array contains any request header or other application-specific values that you decide how to use. If you want to add additional custom FieldValues to the userInfo, you must implement an iFieldBuilder. The InquireAuthenticator uses the configured iFieldBuilder by calling the `buildFieldValue` method. You can add any FieldValues. The parameters for this interface method are:

```
public interface IFieldBuilder {

    public void buildFieldValue(Map<String, String> requestHeaderMap,
                               HttpServletRequest httpRequest);

}
```

<b>Map&lt;String, String&gt; requestHeaderMap:</b>	Map of available request headers, including any fields available from the HTTP header, such as <i>User-Agent</i> , <i>Accept</i> , <i>Referrer</i> , or custom fields. For more information, enter this URL into a browser: <a href="http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html">http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html</a> .
<b>HttpServletRequest request:</b>	HttpServletRequest object created automatically by the Java Servlet API when a request comes in to a web application written in Java. The HttpServletRequest object encapsulates all data needed to respond to the request.

## IUpdatableUserAuthenticator Interface Methods

The following table lists IUpdatableUserAuthenticator interface methods:



Method	Description
<b>boolean isUpdateRequired(IRoleBasedUser iUser, boolean newUser)</b>	Returns the boolean value indicating whether to update the user information in the Information Manager database. <b>Note:</b> This method has two input parameters: IRoleBasedUser and Boolean flag. IRoleBasedUser is a user object and Boolean flag indicates whether the user is new or already exists in the database.
<b>boolean enforcePasswordLifecycle()</b>	Returns the boolean value indicating whether the password life cycle must be enforced.

# Implementing iAuthenticator

An implementation of iAuthenticator consists of one or more installations across physical or virtual machines.

Each installation is designated by `$INQUIRA_ROOT` and may have:

- One Information Manager instance hosting one or more Information Manager-based applications, such as Management Console, Information Manager Tag Library Custom Applications, Information Manager Web Services Custom Applications, Client Library Custom Applications and InfoCenter)
- One Search Scheduler (Indexer) instance
- One or more Search Runtime instances
- One or more Search Workclient instances
- One or more Search Query Coordinator instances
- One or more Search QueryWorker instances

## Configuring Information Manager

After you create a custom iAuthenticator, you must configure each Information Manager Instance in the implementation to use the custom iAuthenticator.

**Important!** You must perform this procedure for each instance of Information Manager.

- 1 Copy the jar files that contain your custom iAuthenticator (and IFieldBuilder if applicable) classes to each of the following locations:
  - Deployed Information Manager Management Console  
`$INQUIRA_ROOT/instances/<company>/appserverim/webapps/InfoManager/WEB-INF/lib/`
  - Deployed Information Manager Web Services  
`$INQUIRA_ROOT/instances/<company>/appserverim/webapps/imws/WEB-INF/lib/`
  - Deployed Information Manager Tag Library based Applications  
`$INQUIRA_ROOT/instances/<company>/appserverim/webapps/<CONTEXT>/WEB-INF/lib/`
- 2 Configure Information Manager to use the custom iAuthenticator (and iFieldBuilder if applicable):
  - a Log in to the Management Console and switch to your repository (if you logged into the SYSTEM repository).
  - b Navigate to **Tools, System Configure**, and then **Expert Mode**.
  - c Set the `REMOTE_AUTHENTICATION_ENABLED` configuration parameter to `true`.
  - d Set the `REMOTE_AUTHENTICATION_CLASS` configuration parameter to your fully qualified custom authenticator class, for example `com.company.package.CustomAuthenticator`.
  - e If you are using custom iFieldBuilder, set the `REMOTE_FIELD_BUILDER_CLASS` configuration parameter to your fully qualified custom iFieldBuilder class.

If the Management Console Application is not implemented on a particular Information Manager instance, you can manually modify the parameters in this procedure by editing the repository configuration file located at:

```
$IM_HOME/config/[REPOSITORY]/config.properties
```

The following example shows how to manually modify the parameters:

```
REMOTE_AUTHENTICATION_STORE_PASSWORD_LOCALLY=10;false
```

```
REMOTE_AUTHENTICATION_CLASS=10;oracle.km.test.authenticator.OKTestAuthenticator
```

# Error Conditions

During the implementation of a custom authenticator, you may encounter one or more of the following errors.

## Error Condition Solutions

**Note:** Each condition logs a detailed error message to the Information Manager runtime log.

### Error Conditions

ClassNotFoundException exception

The custom iAuthenticator is not used during authentication

InstantiationException or ClassNotFoundException thrown

The custom iFieldValues is not being added to FieldValues array of iAuthenticator.authenticate method call

ClassCastException thrown

"Authentication denied" is logged to the Information Manager runtime log at the following location:  
\$INQUIRA\_ROOT/instances/<company>/Inquirea-InfoManager-infomanager.log

An IllegalStateException is thrown and the message *More than one User was found* is logged to the Information Manager runtime log at this location:  
\$INQUIRA\_ROOT/instances/<company>/Inquirea-InfoManager-infomanager.log

The user is not authenticated and the following message *No valid roles and/or valid views* is logged to the Information Manager runtime log at this location:  
\$INQUIRA\_ROOT/instances/<company>/Inquirea-InfoManager-infomanager.log

### Possible Solutions

Verify that the custom iAuthenticator is in the Information Manager-based application's class path.

Verify that REMOTE\_AUTHENTICATION\_ENABLED is set to `true` in `config.properties`.

Verify that REMOTE\_AUTHENTICATION\_CLASS is set to the full and correct path to the custom iAuthenticator.

Verify that the REMOTE\_FIELD\_BUILDER\_CLASS is set to the full and correct path of the custom IFieldBuilder that is being used.

Verify that your custom authenticator returns an instance of iRoleBasedUser.

Verify that the user who is being authenticated is authorized to log in and that the passed-in username and password match the username and password stored in the remote system.

Verify that your Information Manager database does not contain duplicate usernames.

Verify that the user who is being authenticated has valid roles and views in the remote system.

# iAuthentication Sample

This section shows a simple example of `iAuthenticator` that validates whether or not a user ID and password match the values in the database. After the user validation, `iAuthenticator` assigns a set of roles and views to the user.

## Implementation Sample Code

**Note:** The following code is an example of an implementation. An actual implementation requires contact with an external realm to authenticate and create the `RoleBasedUser`.

```
public class CEGTestAuthenticator implements IAuthenticator, IFieldNames {

    private static final Logger logger =
        Logger.getLogger(CEGTestAuthenticator.class.getName());

    /* {@inheritDoc} */
    IUser authenticate(FieldValue[] userInfo, Map roles2PermissionsMap, long timestamp)
        throws InquiraAuthenticationException {

        IUser        user = null;
        String        userId = null;
        String        password = null;
        String        repository = null;

        logger.info("CEGTestAuthenticator.authenticate() called...");

        //loop thru the userInfo values to get username, password, and domain
        for (int i=0 ; i<userInfo.length ; i++) {
            FieldValue fieldValue = userInfo[i];
            if(userId==null && fieldValue.getName().equals(FIELD_USER_ID)) {
                userId = fieldValue.getValue();
            }
            if(password==null && fieldValue.getName().equals(FIELD_PASSWORD)) {
                password = fieldValue.getValue();
            }
            if(repository==null && fieldValue.getName().equals(FIELD_DOMAIN)) {
                repository = fieldValue.getValue();
            }
        }

        //userid and password must exist
        if(userId!=null && password!=null) {
            if(repository!=null && repository.equals(getDomain())) {

                //only allow jdoe to authenticate the text, "Users roles, views, and
                //reportingUserGroup should exist in roles2PermissionsMap (sans prefix)."
                if(userId.equals("jdoe") && password.equals("password")) {
                    HashSet<String> roles = new HashSet<String>();
                    roles.add("R_DEFAULT_ADMINISTRATION_ROLE");
                }
            }
        }
    }
}
```

```

roles.add("V_TEST");
roles.add("G_AN_REPORTING_USERGROUP");
Permissions permissions = new Permissions( );
permissions.add(new StandardPermission("none"));

try {
    user = new RoleBasedUser(userId,
                            repository,
                            "John Doe",
                            new SimplePrincipal(userId),
                            roles,
                            permissions,
                            timestamp,
                            null);
} catch(InquiraSecurityException ise) {
    throw new InquiraAuthenticationException(ise);
}

((RoleBasedUser)user).setEmailAddress("jdoe@inquira.com");

//Set the below properties if it needs be up updated in InfoManager
//Set the Boolean value to make user active or inactive. If value is not
set user will be active by default
((RoleBasedUser)user).setActive(true);

((RoleBasedUser)user).setAlias("AliasName");
HashSet cat = new HashSet( );
//Add the category reference keys.
cat.add( "CATEGORY_1" );
cat.add( "CATEGORY_2" );
cat.add( "PARNETCATEGORY1" );
((RoleBasedUser)user).setAssignedCategories(cat);

//Set default view reference key
((RoleBasedUser)user).setDefaultSubsite("VIEW11");

//Set the default locale of the user. Instance of Locale should be created
by passing language and country code like new //Locale("en","US") and this
should be available in InfoManager.

((RoleBasedUser)user).setDefaultLocale(new Locale("en","US"));
HashSet loca = new HashSet( );
loca.add(new Locale("en","US"));
loca.add( new Locale("fr", "FR"));
loca.add(new Locale("it","IT"));
((RoleBasedUser)user).setContentLocales(loca);
((RoleBasedUser)user).setReceiveAssigned(true);
((RoleBasedUser)user).setReceivePerform(true);
((RoleBasedUser)user).setSubscribeOnTopicCreation(true);
((RoleBasedUser)user).setSubscribeOnTopicReply(false);
((RoleBasedUser)user).setSubscriptionSchedule(2);

//user key values are used to store some user level customizations/
preferences like below

UserKeyValue uk1 = new UserKeyValue();
uk1.setKeyID("search_prefs_languages");
uk1.setValue("en-US,fr-FR");
UserKeyValue uk2 = new UserKeyValue();
uk2.setKeyID("lastquestion");
uk2.setValue("newQ111");

```

```
        HashSet uks = new HashSet();
        uks.add(uk1);
        uks.add(uk2);

        ((RoleBasedUser)user).setKeyValues(uks);
    }
}
else {
    new InquiraAuthenticationException("Invalid authentication domain");
}
}
else {
    throw new InquiraAuthenticationException("Username and password required to
    authenticate");
}
}

return user;
}

/** {@inheritDoc} */
public Field[] getAuthenticationFields() throws InquiraAuthenticationException {
    //this authenticator requires a username, password, and repository to authenticate.
    return new Field[] {new InputField(FIELD_USER_ID),
        new InputField(FIELD_PASSWORD, true),
        new InputField(FIELD_DOMAIN)};
}

/** {@inheritDoc} */
public String getDomain() {
    return "TESTREPOSITORY";
}
}
```

# Implementing IUpdatableUserAuthenticator

This section describes how to use the `IUpdatableUserAuthenticator` API. The `IUpdatableUserAuthenticator` interface extends the `iAuthenticator` interface.

The `IUpdatableUserAuthenticator` API contains two abstract methods: `isUpdateRequired(IRoleBasedUser iUser, boolean newUser)` and `enforcePasswordLifecycle()`.

You can implement the `isUpdateRequired` abstract method to update the user information in the Information Manager database and the `enforcePasswordLifecycle` method enables you to enforce password life cycle that you configure in Information Manager.

## Configuring IUpdatableUserAuthenticator

After you create a custom `IUpdatableUserAuthenticator`, you must configure each Information Manager Instance in the implementation to use the custom `IUpdatableUserAuthenticator`.

**Important!** You must perform this procedure for each instance of Information Manager.

- 1 Copy the jar files that contain your custom `IUpdatableUserAuthenticator` (and `IFieldBuilder` if applicable) classes to each of the following locations:
  - Deployed Information Manager Management Console  
`$INQUIRA_ROOT/instances/<company>/appserverim/webapps/InfoManager/WEB-INF/lib/`
  - Deployed Information Manager Web Services  
`$INQUIRA_ROOT/instances/<company>/appserverim/webapps/imws/WEB-INF/lib/`
  - Deployed Information Manager Tag Library based Applications  
`$INQUIRA_ROOT/instances/<company>/appserverim/webapps/<CONTEXT>/WEB-INF/lib/`
- 2 Configure Information Manager to use the custom `IUpdatableUserAuthenticator` (and `IFieldBuilder` if applicable):
  - a Log in to the Management Console and switch to your repository (if you logged into the SYSTEM repository).
  - b Navigate to **Tools, System Configure**, and then **Expert Mode**.
  - c Set the `REMOTE_AUTHENTICATION_ENABLED` configuration parameter to `true`.
  - d Set the `REMOTE_AUTHENTICATION_CLASS` configuration parameter to your fully qualified custom `IUpdatableUserAuthenticator` interface, for example `com.company.package.CustomIUpdatableUserAuthenticator`.



- e If you are using custom `iFieldBuilder`, set the `REMOTE_FIELD_BUILDER_CLASS` configuration parameter to your fully qualified custom `iFieldBuilder` class.

If the Management Console Application is not implemented on a particular Information Manager instance, you can manually modify the parameters in this procedure by editing the repository configuration file located at:

```
$IM_HOME/config/[REPOSITORY]/config.properties
```

The following example shows how to manually modify the parameters:

```
REMOTE_AUTHENTICATION_STORE_PASSWORD_LOCALLY=10;false
REMOTE_AUTHENTICATION_CLASS=
10;oracle.km.test.authenticator.UpdatableUserTestAuthenticator
```

## IUpdatableUserAuthenticator Sample Code

This section shows a simple example of `IUpdatableUserAuthenticator` that implements the `isUpdateRequired()` and `enforcePasswordLifecycle()` abstract methods, and also validates whether the user ID and password match the values in the Information Manager database.

**Note:** The following code is an example of interface implementation. An actual implementation will need a contact with an external realm to authenticate and create the `RoleBasedUser`.

```
public class UpdatableUserTestAuthenticator implements IUpdatableUserAuthenticator,
IFieldNames {

    private static final Logger logger =
    Logger.getLogger(UpdatableUserTestAuthenticator.class.getName());

    /* {@inheritDoc} */
    IUser authenticate (FieldValue [] userInfo, Map roles2PermissionsMap, long timestamp)
        throws InquireaAuthenticationException {

        IUser        user = null;
        String        userId = null;
        String        password = null;
        String        repository = null;

        logger.info("UpdatableUserTestAuthenticator.authenticate() called...");

        //loop thru the userInfo values to get username, password, and domain
        for (int i=0 ; i<userInfo.length ; i++) {
            FieldValue fieldValue = userInfo[i];
            if(userId==null && fieldValue.getName().equals(FIELD_USER_ID)) {
                userId = fieldValue.getValue();
            }
            if(password==null && fieldValue.getName().equals(FIELD_PASSWORD)) {
                password = fieldValue.getValue();
            }
            if(repository==null && fieldValue.getName().equals(FIELD_DOMAIN)) {
                repository = fieldValue.getValue();
            }
        }

        //userid and password must exist
        if(userId!=null && password!=null) {
            if(repository!=null && repository.equals(getDomain())) {

                //only allow jdoe to authenticate the text, "Users roles, views, and
```

```

reportingUserGroup should exist in roles2PermissionsMap (sans prefix)."
if(userId.equals("jdoe") && password.equals("password")) {
    HashSet<String> roles = new HashSet<String>();
    roles.add("R_DEFAULT_ADMINISTRATION_ROLE");
    roles.add("V_TEST");
    roles.add("G_AN_REPORTING_USERGROUP");
    Permissions permissions = new Permissions();
    permissions.add(new StandardPermission("none"));

    try {
        user = new RoleBasedUser(userId,
                                repository,
                                "John Doe",
                                new SimplePrincipal(userId),
                                roles,
                                permissions,
                                timestamp,
                                null);

    } catch(InquireaSecurityException ise) {
        throw new InquireaAuthenticationException(ise);
    }

    ((RoleBasedUser)user).setEmailAddress("jdoe@inquirea.com");

    //Set the below properties if it needs be up updated in InfoManager
    //Set the Boolean value to make user active or inactive. If value is not
    set user will be active by default
    ((RoleBasedUser)user).setActive(true);

    ((RoleBasedUser)user).setAlias("AliasName");
    HashSet cat = new HashSet();
    //Add the category reference keys.
    cat.add("CATEGORY_1");
    cat.add("CATEGORY_2");
    cat.add("PARNETCATEGORY1");
    ((RoleBasedUser)user).setAssignedCategories(cat);

    //Set default view reference key
    ((RoleBasedUser)user).setDefaultSubsite("VIEW11");

    //Set the default locale of the user. Instance of Locale should be created
    by passing language and country code like new //Locale("en","US") and this
    should be available in InfoManager.

    ((RoleBasedUser)user).setDefaultLocale(new Locale("en","US"));
    HashSet loca = new HashSet();
    loca.add(new Locale("en","US"));
    loca.add(new Locale("fr","FR"));
    loca.add(new Locale("it","IT"));
    ((RoleBasedUser)user).setContentLocales(loca);
    ((RoleBasedUser)user).setReceiveAssigned(true);
    ((RoleBasedUser)user).setReceivePerform(true);
    ((RoleBasedUser)user).setSubscribeOnTopicCreation(true);
    ((RoleBasedUser)user).setSubscribeOnTopicReply(false);
    ((RoleBasedUser)user).setSubscriptionSchedule(2);

    //user key values are used to store some user level customizations/
    preferences like below

    UserKeyValue uk1 = new UserKeyValue();
    uk1.setKeyID("search_prefs_languages");

```

```

        uk1.setValue("en-US,fr-FR");
        UserKeyValue uk2 = new UserKeyValue();
        uk2.setKeyID("lastquestion");
        uk2.setValue("newQ111");

        HashSet uks = new HashSet();
        uks.add(uk1);
        uks.add(uk2);

        ((RoleBasedUser)user).setKeyValues(uks);
    }
}
else {
    new InquireaAuthenticationException("Invalid authentication domain");
}
}
else {
    throw new InquireaAuthenticationException("Username and password required to
    authenticate");
}

return user;
}

/** {@inheritDoc} */
public Field[] getAuthenticationFields() throws InquireaAuthenticationException {
    //this authenticator requires a username, password, and repository to authenticate.
    return new Field[] {new InputField(FIELD_USER_ID),
        new InputField(FIELD_PASSWORD, true),
        new InputField(FIELD_DOMAIN)};
}

/** {@inheritDoc} */
public String getDomain() {
    return "TESTREPOSITORY";
}

/**
 * Does the caller wants the password restrictions or not
 * @return true enforce rules : false do not enforce
 */
public boolean enforcePasswordLifecycle() {
    boolean enforcePasswCycle;
    // Write your business logic to determine if password life cycle to be enforced.
    //Or just set enforcePasswCycle = true if password life cycle to be enforced
    // otherwise set it to false.
    return enforcePasswCycle;
}

/**
 * Adds an additional method on top of the {@link IAuthenticator} so that *the user will
 * be modified if it returns true
 * @return
 */
public boolean isUpdateRequired(IRoleBasedUser arg0, boolean newUser) {
    boolean isUpdateRequired;
    // Write your business logic to determine if user information needs to be updated
    in
    //InfoManager database.
    return isUpdateRequired;
}
}

```



# API Reference

The following is a list of the full package names of classes mentioned in this guide.

## Package Names of Classes

- `com.inquiria.infra.IRoleBasedUser`
- `com.inquiria.infra.IUser`
- `com.inquiria.infra.security.AbstractUser`
- `com.inquiria.infra.security.Field`
- `com.inquiria.infra.security.FieldValue`
- `com.inquiria.infra.security.InputField`
- `com.inquiria.infra.security.InquiriaAuthenticationException`
- `com.inquiria.infra.security.IAuthenticator`
- `com.inquiria.infra.security.IFieldNames`
- `com.inquiria.infra.security.StandardPermission`
- `com.inquiria.infra.security.impl.RoleBasedUser`
- `com.inquiria.services.Idapservices.IFieldBuilder`
- `com.inquiria.util.security.SimplePrincipal`
- `java.security.Permissions`
- `java.security.BasicPermission`
- `java.security.Principal`
- `javax.security.auth.Subject`
- `import com.inquiria.infra.UserKeyValue`

## Public API Classes and Methods

API classes and methods are available as HTML files. In a default installation, the files are placed in the `InfoManager/docs/iAuthentication` directory.

**Tip:** You can drill down on each Java package by first opening the `index.html` file.