Oracle® Fusion Middleware

Securing Oracle Enterprise Data Quality

14c (14.1.2.0.0)
F85537-02
March 2025

Oracle Fusion Middleware Securing Oracle Enterprise Data Quality, 14c (14.1.2.0.0)

F85537-02

Primary Author: Oracle Corporation

Contents

A Configuring EDQ to support Windows Integrated Authentication (Kerberos)

B Configuring Single Sign On with Oracle Access Manager (OAM)

Preface

This guide explains the Oracle Enterprise Data Quality security features and administration.

Audience

The intended audience of this guide are experienced administrators, Java developers, deployers, and application managers who want to ensure that EDQ meets Oracle security standards.

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 customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documents

For more information, see the following documents in the Enterprise Data Quality documentation set.

EDQ Documentation Library

The following publications are provided to help you install and use EDQ:

- *Release Notes for Enterprise Data Quality*
- *Installing and Configuring Enterprise Data Quality*
- *Administering Enterprise Data Quality*
- *Understanding Enterprise Data Quality*
- *Integrating Enterprise Data Quality With External Systems*
- *Securing Oracle Enterprise Data Quality*
- *Installation and Upgrade Guide*
- *Release Notes*

Find the latest version of these guides and all of the Oracle product documentation at

https://docs.oracle.com

Online Help

Online help is provided for all Enterprise Data Quality user applications. It is accessed in each application by pressing the **F1** key or by clicking the Help icons. The main nodes in the Director project browser have integrated links to help pages. To access them, either select a node and then press **F1**, or right-click on an object in the Project Browser and then select **Help**. The EDQ processors in the Director Tool Palette have integrated help topics, as well. To access them, right-click on a processor on the canvas and then select **Processor Help**, or left-click on a processor on the canvas or tool palette and then press **F1**.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

About EDQ Security

This chapter explains the key security concepts used in Enterprise Data Quality (EDQ). It includes the following sections:

- Introducing EDQ Security
  Security within EDQ applies to accessing the application (ensuring that only authorized users can access it, and that data within the application is secured), and auditing of user actions to identify anomalies.
- EDQ User Groups
  All rights to EDQ are granted using a set of permissions that are granted to a user group (or 'internal' group). Where users are managed externally, for example in WebLogic, or in Active Directory, or in an alternative LDAP provider, permissions are granted by mapping External Groups from that directory to these internal groups.
- EDQ Permissions
  EDQ has four types of permission, as listed below. Permissions are granted to users by means of EDQ User Groups, which are in turn mapped to external user groups.
- Default EDQ Groups and Permissions
  The default groups and a summary of their permissions are listed below.
- Default Administrator User Accounts
  On an installation of EDQ on WebLogic server, the weblogic administrator user account (normally 'weblogic', with a password selected when the domain was created), is a member of the 'Administrators' group in WebLogic, which is mapped to the 'Administrators' group in EDQ by means of a default *login.properties* file in the EDQ Home configuration directory.
- Mapping External Groups to EDQ Groups
  External groups are mapped to internal EDQ Groups from the Administration pages of the EDQ Launchpad. An external group is granted all permissions from all EDQ groups that it is mapped to.
- Terms Used in this Guide
  This section provides information about the terms used in this guide.

Introducing EDQ Security

Security within EDQ applies to accessing the application (ensuring that only authorized users can access it, and that data within the application is secured), and auditing of user actions to identify anomalies.

This guide covers the following:

- Providing secure access control
- Securing data at rest and in transport.
- Providing appropriate security auditing capabilities to ensure user activity can be securely logged and traced.

For more information, see About Oracle Enterprise Data Quality in Understanding Oracle Enterprise Data Quality.

- Authentication
- Authorization
- Encryption
- Auditing

Authentication

Details of users and groups in EDQ can be stored within its own internal directory or taken from an external LDAP server, such as Microsoft Active Directory. Using external authentication sources enables EDQ to share user credentials with other systems, reducing the number of passwords that users need to remember and maintain, while eliminating overhead in management of users and groups.

Authorization

Authorization controls what users can do once they have authenticated successfully. Authorization of users is based on a model of users, and permissions associated with groups. A user is a member of one or more groups (either directly or by mapping an external group to an internal group), and is authorized according to the permissions that are associated with that group.

Encryption

Both the WebLogic and Tomcat servers support HTTPS and require that traffic between the client and EDQ is encrypted so that it cannot be read or modified in transit. For environments where HTTPS is not an option, EDQ encrypts passwords sent between the client and server.

Auditing

EDQ supports auditing of user actions using the Oracle Fusion Middleware Audit Framework. In addition, EDQ can be configured to write audit information to files.

EDQ User Groups

All rights to EDQ are granted using a set of permissions that are granted to a user group (or 'internal' group). Where users are managed externally, for example in WebLogic, or in Active Directory, or in an alternative LDAP provider, permissions are granted by mapping External Groups from that directory to these internal groups.

Users in the external group then have the permissions from whichever group or groups the external group is mapped to. Where users are managed in EDQ (in the 'internal' realm), these users are made direct members of one or more of the internal groups, and therefore have the associated permissions.

EDQ Permissions

EDQ has four types of permission, as listed below. Permissions are granted to users by means of EDQ User Groups, which are in turn mapped to external user groups.

- Application Permissions
- Functional Permissions

Application Permissions

These are simple permissions that grant (or deny) groups of users access to log in to EDQ user applications, such as Director, Server Console, and Case Management.

Functional Permissions

These are permissions to access certain functions of the system, for example, the ability to modify a Reference Data set in Director, or the ability to change the state of a case, or alert in Case Management.

Dynamic Permissions (in Case Management)

These permissions are dynamically created within a Case Management workflow as configured in Case Management Administration. Dynamic permissions are used to restrict access to specific cases or alerts or to case comments or attachments.

Project Permissions

By default, a project created in Director is accessible to all user groups. However, projects may be restricted to a smaller set of groups. Where this is the case, any user that is not in a group that has access to the project, will not be able to see or use the project in any way.

> **Note:**
>
> Any user that has the ability to add projects in Director automatically has access to all projects.

Default EDQ Groups and Permissions

The default groups and a summary of their permissions are listed below.

To see full details on the precise set of permissions granted to each group, select a group from the Administration - Groups option on the EDQ Launchpad (after logging in as an administrator), and click on the **Edit** button.

Table 1-1 Permissions for various user groups

Group	Summary	Functional Permissions	Application Permissions
Administrators	Power users with all functional and administrative privileges	All Dynamic Case Management permissions are not automatically granted to Administrators	All

Table 1-1 (Cont.) Permissions for various user groups

Group	Summary	Functional Permissions	Application Permissions
Data Stewards	Users with review access to Director and Dashboard, with permission to review all results, resolve issues, and make manual match and merged output decisions, but without permission to create or change any processing logic	Read-only permissions to Director Full permissions to Match Review	Dashboard Director Match Review Issue Manager Case Management Server Console
Executives	Users with access to Dashboard results only	Dashboard: View Dashboard	Dashboard
Match Reviewers	Users with access to the Match Review application, the Case Management application and Dashboard only	Basic Case Management permissions (no bulk updates and cannot view cases assigned to others) Dashboard: View Dashboard	Match Review Issue Manager Case Management Dashboard
Review Managers	Users with access to the Case Management application, with permission to configure case management and perform bulk edits on cases and alerts.	Full Case Management permissions except deletion and editing of audit trail activities	Case Management Dashboard
Data Analysts	Users with permission to create and modify processing logic in Director, but with no administration privileges	Full access to Director except the ability to modify System-level (cross-project) configuration and access the Users data store.	Director Server Console Match Review Case Management Issue Manager

Default Administrator User Accounts

On an installation of EDQ on WebLogic server, the weblogic administrator user account (normally 'weblogic', with a password selected when the domain was created), is a member of the 'Administrators' group in WebLogic, which is mapped to the 'Administrators' group in EDQ by means of a default *login.properties* file in the EDQ Home configuration directory.

This therefore grants the weblogic user, and any other users in the WebLogic Administrators group, administrative access to EDQ.

> **Note:**
>
> Ensure this mapping is fixed and intended only to grant initial access so that the actual required permissions can be set by mapping external groups to internal groups, see Mapping External Groups to EDQ Groups

On an installation of EDQ on Tomcat, a default internal administrator user account called '*dnadmin*' is created. The initial password for this user is also '*dnadmin*', but it must be changed to a more secure password after initial login. In such an installation, by default this is the only user account with administration rights, and it is important not to delete the user or forget the password, as otherwise it may not be possible to access the system.

Mapping External Groups to EDQ Groups

External groups are mapped to internal EDQ Groups from the Administration pages of the EDQ Launchpad. An external group is granted all permissions from all EDQ groups that it is mapped to.

To map external groups to internal EDQ groups, follow the steps below:

1. Log in to the EDQ Launchpad as an Administrator.
2. Click on the Administration dropdown link, and click on External Groups.
3. Expand the name of the external realm (this is 'ORACLE' on a default system) and click on a group to edit its mappings.
4. Select the EDQ Groups you want to grant to the external group, and click on Save to save the changes.

It is useful and desirable to create some external groups on the domain that can be mapped precisely to internal EDQ groups, to ensure optimal permission assignment.

> **Note:**
>
> For installations with a large number of external user groups, an optional filter can be specified in a local *login.properties* file to narrow down the list of groups that is available in this screen. See Filtering Groups
>
> for more information.

Terms Used in this Guide

This section provides information about the terms used in this guide.

The following terms are used in this guide:

- AD – Active Directory
- Certificate – Generally refers to an X.509 certificate
- Kerberos – Network authentication protocol
- LDAP – Lightweight Directory Access Protocol

- OID – Oracle Internet Directory
- OPSS - Oracle Platform Security Services
- SSL – Security Sockets Layer, a protocol for encrypted connections over which application traffic can be transported. Replaced by TLS, although SSL is still used as a generic term.
- TLS – Transport Layer Security, a successor to SSL.
- WLS – WebLogic Server
- X.509 Certificate – A certificate issued by a trusted authority (certificate authority) to certify that a specified entity (individual, organization, server, or other entity) holds the matching private key for a public key.

2

Applying Recommended Security Settings

This chapter provides tips for securing the EDQ environment.
The chapter includes the following sections:

- Configuring SSL with WebLogic
  This section describes how to configure SSL with WebLogic.
- Configuring SSL with Tomcat
  This section describes how to configure SSL with Tomcat.
- Processor Security
  This section provides information about processor security.
- Encrypting LDAP Connections
  Connections from EDQ to an LDAP directory can be encrypted using either an SSL/TLS connection layer or by negotiating encryption after a connection has been established (StartTLS).
- Encrypting Database Connections
  JDBC URL syntax for connections over TLS is dependent on the database driver being used.
- Limit Concurrent Logins
  A limit can be specified in *login.properties* for the number of concurrent logins by an individual user. The limit is concurrent logins per application. So if the sessionlimit is 1 a user can login to director, server console, case management at the same time but cannot login twice to any.
- Account with Minimal Permissions for Service Integration
  An EDQ account used by a remote system such as Siebel or Oracle Data Integrator should have the minimum set of permissions on an EDQ system.
- Protect JNDI Data Sources
  Unless specific steps are taken, a user in EDQ can set up a data store with a reference to the JNDI name of any existing data source and then access data in these schemas, which can contain very sensitive information.
- Blocking the Upload of Malicious Files
  To configure the server to block the upload of files according to the names or type of files, set the following parameters in `director.properties`.

Configuring SSL with WebLogic

This section describes how to configure SSL with WebLogic.

For instructions on configuring SSL with WebLogic Server, see the WebLogic documentation:

Overview of Configuring SSL in WebLogic Server

Configuring SSL with Tomcat

This section describes how to configure SSL with Tomcat.

For additional Tomcat information, see *Apache Tomcat Configuration Reference* at Apache Tomcat.

For additional `mod_ssl` information, see *Apache Module mod_ssl* at

http://httpd.apache.org/docs/2.2/mod/mod_ssl.html

Processor Security

This section provides information about processor security.

For more information, see Configuring Secure Script Execution in *Administering Oracle Enterprise Data Quality*.

Encrypting LDAP Connections

Connections from EDQ to an LDAP directory can be encrypted using either an SSL/TLS connection layer or by negotiating encryption after a connection has been established (StartTLS).

Encrypting Database Connections

JDBC URL syntax for connections over TLS is dependent on the database driver being used.

Connections to an Oracle database can be encrypted by adding the following property to the datasource connection pool configuration.:

```
oracle.net.encryption_client = REQUIRED
```

Limit Concurrent Logins

A limit can be specified in *login.properties* for the number of concurrent logins by an individual user. The limit is concurrent logins per application. So if the sessionlimit is 1 a user can login to director, server console, case management at the same time but cannot login twice to any.

This can be configured either globally or on a per realm basis. To set this globally for all realms, use the following line:

```
sessionlimit = 1
```

To use different settings for different realms, specify the realm name before the parameter - for example:

```
internal.sessionlimit = 1
```

> **Note:**
>
> Using the line given above, you can also limit the concurrent logins in an `internal' realm, meaning the users set up and administered in EDQ itself.

```
<external realm name>.sessionlimit = 1
```

Account with Minimal Permissions for Service Integration

An EDQ account used by a remote system such as Siebel or Oracle Data Integrator should have the minimum set of permissions on an EDQ system.

Specifically, the account should be in a custom group with the following permissions only, and no access to log into any user applications or perform any other functions:

- System / Connect to Messaging System - so that it is authorized to communicate with EDQ web services and JMS.
- System/ System Administration - so that it is authorized to connect to the EDQ Management (JMX) Port and can initiate jobs.
- Permissions to any projects containing any service interfaces (e.g. web services or jobs) that it needs to be able to call.

Protect JNDI Data Sources

Unless specific steps are taken, a user in EDQ can set up a data store with a reference to the JNDI name of any existing data source and then access data in these schemas, which can contain very sensitive information.

To protect JNDI data sources in EDQ, specify the names (or regular expressions matching the names) in `director.properties`:

```
protected.jndi.datasources = <space separated list of JNDI names>
```

For example:

```
protected.jndi.datasources = jdbc/edqconfig jdbc/edqresults
```

The property is a space-separated list of regexes so you could also use:

```
protected.jndi.datasources = jdbc/edq.*
```

> **Note:**
>
> When setting this value in the local *director.properties*, always include the default setting from the base properties file. This setting prevents access to the WebLogic internal data sources as well as the EDQ data sources.

Blocking the Upload of Malicious Files

To configure the server to block the upload of files according to the names or type of files, set the following parameters in `director.properties`.

Parameter Name	Type	Default Value
`upload.check.names`	Boolean	True. If the value is set to false, no checks are done for the parameter name.
`upload.valid.names`	Text	A comma or separated list of valid file names.

Parameter Name	Type	Default Value
`upload.valid.types`	Text	A comma or separated list of Java-recognised MIME file types, such as text/csv, image/bmp, text/plain etc For a list of MIME file types, see MIME types. **Note:** To disable the check (to allow files of any type), set an empty value to the parameter, `upload.valid.types=`
`upload.invalid.names`	Text	A comma or separated list of invalid file names.

Note:

The following properties are applicable for `upload.valid.names` and `upload.invalid.names` parameters.

1. Each name in this list is either a suffix (such as `.pdf`) or a regex prefixed with `~`. If a regex is used it is matched against the full name of the file.
2. A file name is rejected if:
   - `upload.valid.names` is specified and the file name does not match any of the items in the list.
   - `upload.invalid.names` is specified and the name matches any of the items in the list.
3. All checks are not case sensitive.

3
User Authentication

This chapter explains how users are authenticated in EDQ.
This chapter includes the following sections:

- The EDQ login.properties File
  User authentication in EDQ is configured using a file named *security/login.properties* in the EDQ configuration area. The file may exist in either the 'home' configuration directory or the 'local' configuration directory, or both.
- WebLogic Installations
  A standard installation of EDQ on WebLogic will use the domain's internal user store for authentication. This contains the domain administration user (usually 'weblogic') and a small set of other users.
- Enabling the internal realm
  To manage users internally, the internal realm needs to be enabled.
- Tomcat Installations
  An installation of EDQ on Tomcat will use the EDQ internal authentication mechanism. This contains the single user 'dnadmin' with full administration rights. Additional users can be added through the EDQ administration web pages.

The EDQ login.properties File

User authentication in EDQ is configured using a file named *security/login.properties* in the EDQ configuration area. The file may exist in either the 'home' configuration directory or the 'local' configuration directory, or both.

If present in both, the settings are merged with the values in the 'local' directory taking precedence. If you need to make changes to the file, always edit a version in the 'local' directory.

The EDQ 'home' configuration area contains a file *security/login.properties.template* which contains example settings for several types of LDAP integration.

The *login.properties* file defines a number of 'realms'. Each realm is an independent store of users. The file will start with global settings, followed by settings for each realm. Standard global settings are:

```
realms = realm1, realm2, …
gss    = false
```

The realms property defines the list of realms configured in this installation. The names are arbitrary, except that the special realm name 'internal' specifies the EDQ internal user store (user dnadmin etc).

The 'gss' setting turns off advanced Kerberos style authentication.

The global settings are followed by blocks of realm specific properties, each prefixed with the realm name from the global list. For example, a configuration which uses the internal realm and an LDAP realm could be:

```
realms                          = internal, corpldap
gss                  = false
corpldap.realm       = EXAMPLE.COM
corpldap.ldap.server = dc1.example.com
…
```

These settings are covered in more detail below.

If more than one realm is defined in *login.properties*, the EDQ login screens - web console and UIs - will contain a dropdown selector for the realm associated with the username and password.

- Static Groups Mapping in login.properties

Static Groups Mapping in login.properties

If you are using an external LDAP user store, and do not wish to use the EDQ internal user store, then you will face a bootstrapping problem when attempting to setup mappings from LDAP groups to EDQ groups. This is done using the EDQ web console and requires an EDQ administrator login. However a LDAP group mapping to the EDQ Administrators group is required before an LDAP user can login to EDQ.

To overcome this problem you can define static group mappings in *login.properties*. The syntax is:

```
realm.xgmap = exgroup1 -> edqgroup1, exgroup2 -> edqgroup2 …
```

'realm' is the realm name as listed in the global realms list. Each 'exgroup' is the name of an external LDAP group; each 'edqgroup' is the name of an EDQ group.

Static mappings should be used only to set the initial Administration mapping; other mappings should be configured in the EDQ web console External Groups page.

For example, to map an LDAP group named 'EDQ-ADMINS' to the EDQ Administrators group, add:

```
corpldap.xgmap = EDQ-ADMINS -> Administrators
```

WebLogic Installations

A standard installation of EDQ on WebLogic will use the domain's internal user store for authentication. This contains the domain administration user (usually 'weblogic') and a small set of other users.

The EDQ default security configuration maps any user in the WebLogic Administrators group to the EDQ Administrators group. No other group mapping are defined initially.

For more information about how to add users, see Users and Groups in *Oracle WebLogic Remote Console Online Help*

As an alternative to managing users in WebLogic you can integrate with an external source of users, normally an LDAP server, such as Active Directory. Integration can be configured in the WebLogic Remote Console or directly from EDQ by editing the security configuration file. The former approach is covered in Integrating External User Management (LDAP) using WebLogic and OPSS and the latter in Configuring External User Management (LDAP) directly with EDQ .

The default *login.properties* in the 'home' configuration directory contains:

```
realms     = opss
opss.realm = ORACLE
```

```
opss.label = Weblogic
opss.type  = opss
opss.xgmap = Administrators -> Administrators
```

This configuration indicates that the OPSS library supplied by WebLogic is used for authentication and that the WebLogic Administrators group is mapped to the EDQ Administrators group.

- Creating Users and Groups in Weblogic Installation

Creating Users and Groups in Weblogic Installation

You can create external users and groups in Weblogic Remote Console and map the users in EDQ Launchpad.

For more information, see Create a User and Create a Group in *Oracle WebLogic Remote Console Online Help*.

Enabling the internal realm

To manage users internally, the internal realm needs to be enabled.

Follow the procedure below to enable the internal realm for users:

1. Create a subdirectory called security in the local configuration directory (*oedq_local_home/security*).
2. Copy the *login.properties* file from the security directory of the base configuration directory (*oedq_home/security)* to *oedq_local_home/security.*
3. Add 'internal' to the comma-separated list of realms.

   Note that 'opss' enables the realm of WebLogic users. To only manage users internally, you can remove opss from the list of realms.
4. Restart the server.

This enables the internal realm of users, and allow internal user accounts to be managed in the Administration pages on the Launchpad. A single default Administrator user '**dnadmin**' will exist, with an initial password of '**dnadmin**' that will need to be changed on first login.

If more than one realm is defined, users will need to select which realm they want to use when they log in.

Tomcat Installations

An installation of EDQ on Tomcat will use the EDQ internal authentication mechanism. This contains the single user 'dnadmin' with full administration rights. Additional users can be added through the EDQ administration web pages.

You can integrate with an external LDAP server such as Active Directory by editing the EDQ security configuration file. This is covered in Configuring External User Management (LDAP) directly with EDQ .

No *login.properties* file is used by default in Tomcat installations because the default setting is to use internal authentication. If additional setting are required, or LDAP integration is required, create a new *security/login.properties* file in the 'local' configuration directory and make changes there.

- Creating Users and Groups in Tomcat Installation

Creating Users and Groups in Tomcat Installation

You can create internal users and groups in EDQ Launchpad.

To add users, follow the steps below:

1. In the EDQ Launchpad, click the Administration panel.
2. On the **Users** tab, click the add icon.
3. In the Create User window, do the following:
   a. Under **Account Details**, add the user details.
   b. Under **Group Selection**, assign the user to a group.

   The user is added and assigned to the selected group.

   Similarly, you can use the **Groups** tab in the Administration panel, to add a group and the group details.

4

Auditing User Activity

This chapter auditing in EDQ.
The chapter includes the following sections:

- Enabling Audit Logging Using Oracle Fusion Middleware Framework
  You can configure EDQ to log audit events using the Oracle Fusion Middleware Audit Framework when EDQ is installed with an Oracle WebLogic Server domain.
- Enabling Audit Logging to Files
  You can enable audit logs to be written to files on disk when EDQ is installed in Apache Tomcat, or if you do not want to use the Oracle Fusion Middleware Audit Framework.

Enabling Audit Logging Using Oracle Fusion Middleware Framework

You can configure EDQ to log audit events using the Oracle Fusion Middleware Audit Framework when EDQ is installed with an Oracle WebLogic Server domain.

For detailed information on the framework see Introduction to Oracle Fusion Middleware Audit Framework in *Oracle Fusion Middleware Securing Applications with Oracle Platform Security Services*.

To enable audit event logging, follow the steps below:

1. Open the Oracle Enterprise Manager Fusion Middleware Control14*c* application using the path:

   ```
   http://[servername]:[weblogic server admin port, e.g. 7001]/em
   ```

2. Navigate to the EDQ domain in the Target Navigation Tree on the left of the window.
3. Right-click the domain and select **Security > Audit Policy**.
4. Select `EDQ` in the **Audit Component Name** field.
5. Select `Custom` in the **Audit Level** field.
6. Select the categories to log, and the events within those categories.
7. Click **Apply**.

   To abandon the changes, you must click **Revert**.

- Configuring the EDQ Audit Events in Fusion Middleware Framework

Configuring the EDQ Audit Events in Fusion Middleware Framework

Set the directory property in the `audit.properties` file in any directory (that exists), relative to your local *config home*.

For example, add the following line to your new file:

```
directory = myAudits
```

where *myAudits* is a folder that exists at the same level as your new `audit.properties` file.

Table 4-1 EDQ Event Category and Types

Event Category	Event Types
Asset Transfer	Import Package
Case Management	Bulk Delete, Bulk Update, Bulk Assignment, Display Data edited, Export, Edit, Assignment updated, State changed, Comment added, Comment deleted, Comment edited, Attachment added, Attachment deleted
Case Management Admin	Case Source Added, Case Source Imported, Case Source Deleted, Permission Added, Permission Modified, Permission Deleted, Workflow Added, Workflow Imported, Workflow Deleted, Parameter Added, Parameter Modified, Parameter Deleted, Reception Action Added, Reception Action Modified, Reception Action Deleted, Reception Transition Added, Reception Transition Modified, Reception Transition Deleted, State Transition Added, State Transition Modified, State Transition Deleted, Workflow State Added, Workflow State Modified, Workflow State Deleted
Group Permission Management	Join group, Leave group, Leave all groups, Create group, Delete group, Change permissions.
Launchpad Management	Extension Add, Extension Delete, Front Page Update
Object Management	Create, Update, Delete.
User Management	Login, Logout, Password Change, Password Expire, User Blocked, User Blocked Temporarily, User Unblocked, User Created, User Updated, User Deleted, Security Configuration Updated.

Table 4-2 Event Attributes and Custom Attribute Slot

Event Attribute	Description	Custom Attribute Slot
Affected user	The name of the user for the logged event.	IAU_STRING_001
Login application	The name of the application that has been logged into.	IAU_STRING_002
Project Name	The name of the project containing the affected object. This attribute is left blank for system-level objects.	IAU_STRING_003
Item Type	The type of object created, modified or deleted.	IAU_STRING_004
Item Name	The name of the object created, modified or deleted.	IAU_STRING_005
Affected user	The name of the user affected by changes made by an administrator.	IAU_STRING_006
Affected group	The name of the group affected by changes made by an administrator.	IAU_STRING_007
Added Permissions	List of permissions added to a group.	IAU_LONGSTRING_001
Removed Permissions	List of permissions removed from a group.	IAU_LONGSTRING_002

The events that can be logged and their corresponding file-based auditing name are listed in the following table. Please note that this is not a complete list.

Table 4-3 Attributes Logged by User Management Event

Weblogic Display Name	File-Based Event Name
Login	login
Logout	logout
Password Change	pwchange
Password Expire	pwexpire

Table 4-3 (Cont.) Attributes Logged by User Management Event

Weblogic Display Name	File-Based Event Name
User Blocked	userblock
User Temporarily Blocked	usertempblock
User Unblocked	userunblock
User Created	usercreate
User Updated	userupdate
User Deleted	userdelete
Security Configuration Updated	secconfig

Table 4-4 Attributes Logged by Group Permission Management Event

Weblogic Display Name	File-Based Event Name
Join group	joingroup
Leave group	leavegroup
Leave all groups	leaveallgroups
Create group	creategroup
Delete group	deletegroup
Change permissions	changepermissions

Table 4-5 Attributes Logged by Object Management Event

Weblogic Display Name	File-Based Event Name
Create	create
Update	update
Delete	delete

Custom attributes are stored in the `iau_custom` table. For more information, see "Audit Reporting with the Dynamic Metadata Model" in *Oracle Fusion Middleware Securing Applications with Oracle Platform Security Services*. The generic attributes for the event are stored in the `iau_common` table. Both of these are in the IAU schema (`[RCUPREFIX]_IAU`).

After the audit logs are enabled, EDQ audits events by calling the central Oracle Fusion Middleware Audit Framework APIs. The audit events can then be stored either as files or in a database for compliance reporting purposes. For more information on how to store and report on the results of auditing, see *Oracle Fusion Middleware Securing Applications with Oracle Platform Security Services*.

Enabling Audit Logging to Files

You can enable audit logs to be written to files on disk when EDQ is installed in Apache Tomcat, or if you do not want to use the Oracle Fusion Middleware Audit Framework.

To enable audit logging, create a file named `audit.properties` in the local configuration directory and add the following line:

```
enabled = true
```

You can then either create a directory named audit in your local configuration directory, or specify a path to an existing directory using the directory property in audit.properties. This path is specified relative to the local configuration directory.

- Configuring the EDQ Audit Events on Disk

Configuring the EDQ Audit Events on Disk

For fine-grained control over specific categories and events that are audited, you can turn certain categories off by adding the following line to `audit.properties`:

```
category.<category name>. enabled = false
```

You can turn the individual events back on, for that category, or turn them off if the category has not been disabled, using the following command:

```
category.<category name>.<event name>.enabled = <true/false>
```

After the audit events are generated, they are placed in per-category files within the configured audit directory. These files contain entries as comma-separated values; the first line contains column headers.

5

Integrating EDQ with a Fusion Middleware Credential Store

This chapter describes how to use an Oracle Fusion Middleware credential store with EDQ running on WebLogic.

This chapter includes the following sections:

- Overview of the Credential Store
  EDQ supports the use of the Oracle Fusion Middleware credential store to hide user names and passwords that are used by EDQ to connect to protected resources, such as a JMS broker or LDAP server.
- Configuring the Credential Store for EDQ
  To configure a credential store, use Oracle Enterprise Manager Fusion Middleware Control.
- Specifying the EDQ Credential Key in Properties Files
  Once you have configured an EDQ credential map in Fusion Middleware Control, use the `.cred.key` property to specify the key name in place of the credential in properties files.
- Examples of Specifying a Key Name
  This section provides examples to specify a key name.

Overview of the Credential Store

EDQ supports the use of the Oracle Fusion Middleware credential store to hide user names and passwords that are used by EDQ to connect to protected resources, such as a JMS broker or LDAP server.

These credentials otherwise would be exposed as clear-text in the EDQ properties files. When a credential store is used, a user name and password are replaced by a key name that serves as an alias for the credential whenever a login is required.

Using a credential store with EDQ comprises the following steps:

- Configuring the Credential Store for EDQ
- Specifying the EDQ Credential Key in Properties Files

Configuring the Credential Store for EDQ

To configure a credential store, use Oracle Enterprise Manager Fusion Middleware Control.

For more information about using this browser-based console, see Administering Oracle Fusion Middleware.

In a credential store, a credential is identified by a *credential map*. The credential map consists of a *map* and one or more *keys*. In EDQ, the default map name is `edq`. The key name is specified by the person who is creating the credential map and serves as the ”alias” for the

credential in the properties files. The person who creates the credential map must be an Oracle Fusion Middleware administrator.

To Configure a Credential Store for EDQ

1. Log in to Oracle Enterprise Manager Fusion Middleware Control as an administrator.
2. Navigate to *Domain* > **Security** > **Credentials** to display the Credentials page.
3. Click **Create Map** to display the Create Map dialog. Once you create a map, you can create multiple keys for it at the same time, or you can add more keys at a later date.
4. Create a map named `edq`, and then click **OK**. The `edq` map name is displayed in the table.
5. Click **Create Key** to display the Create Key dialog.
6. Select the following in this dialog:
   - Select the `edq` map from the **Select Map** menu.
   - Enter a name for the key in the **Key** text box. This is the key name that will be entered in the properties files to replace the credential.
   - Select **Password** from the **Type** menu.
   - Enter the user name for the EDQ user in the **User Name** field and enter the password for that user in the **Password** field. Confirm the password in the **Confirm Password** field.
   - Optionally, you can add a description of this credential.
7. Click **OK** to return to the Credentials page. The new key is displayed under the `edq` map icon.

Specifying the EDQ Credential Key in Properties Files

Once you have configured an EDQ credential map in Fusion Middleware Control, use the `.cred.key` property to specify the key name in place of the credential in properties files.

The syntax is this:

```
prefix.cred.key = keyname
```

It replaces the standard, non-secured `username` and `password` entries:

```
prefix.username = username
prefix.password = password
```

The following shows an entry for a credential for user ”myuser”, followed by an entry for the same credential as represented by its key name.

Non-secured Credential in director.properties

This example shows the regular way of using the `username` and `password` properties to specify the actual user name and password.

```
sccs.vcs.username = myuser
sccs.vcs.password = mypassword1234
```

Secured Credential in director.properties

This example uses the `cred.key` property to specify a key name from the credential store in place of the login credential.

```
sccs.vcs.cred.key = mykey1
```

Secured Password-Only Entry

In cases where only a password is required, for example if creating a keystore for JMX over SSL, append the `.cred.key` property to the property name. The following is an example:

```
management.ssl.km.storepw.cred.key = mykey1
```

Examples of Specifying a Key Name

This section provides examples to specify a key name.

These examples show additional ways to specify credentials by means of a key name.

Connection to a JMS Broker

This example shows a realtime bucket definition in which a credential is required to connect to a JMS broker.

The following is the unsecured way of specifying the credential:

```
<messengerconfig>
  …
  username = myuser
  password = mypassword1234
 …
</messengerconfig>
…
```

The following is the secure specification using the key name:

```
…
<messengerconfig>
  …
  cred.key = mykey1
 …
</messengerconfig>
…
```

Connection to a JNDI Store

This example uses a credential to connect to a JNDI store.

The following is the unsecured way of specifying the credential:

```
…
<messengerconfig>
  …
```

```
  java.naming.security.principal   = myuser
  java.naming.security.credentials = mypassword1234
 …
</messengerconfig>
…
```

The following is the secure specification using the key name. In this case, the `jndi` prefix is required, so the `.cred.key` is appended to it.

```
…
<messengerconfig>
  …
  jndi.cred.key = mykey1
 …
</messengerconfig>
…
```

Connecting to an LDAP Server

This example shows the correct syntax for specifying a connection to an LDAP server in the `login.properties` file.

Non-secured entry:

```
myrealm.ldap.user = myuser
myrealm.ldap.pw = mypassword
```

Secured entry with credential store key:

```
myrealm.ldap.cred.key = mykey1
```

6

Integrating External User Management (LDAP) using WebLogic and OPSS

This chapter provides information on integration to LDAP using Weblogic and OPSS. It includes the following sections:

- Understanding Security Realms, Providers and Control Flags
  A WebLogic domain can contain one or more Security Realms. Each realm defines a security configuration, users and groups. Only one realm can be active at any one time, and generally there is no benefit in creating additional realms. The default realm in a WebLogic domain is named '**myrealm**'.
- Configuring WebLogic to use LDAP
  This section provides information about prerequisites and the procedure to configure WebLogic.

Understanding Security Realms, Providers and Control Flags

A WebLogic domain can contain one or more Security Realms. Each realm defines a security configuration, users and groups. Only one realm can be active at any one time, and generally there is no benefit in creating additional realms. The default realm in a WebLogic domain is named '**myrealm**'.

Each realm contains a number of security 'providers'. Providers are 'authenticators', which handle user authentication and user stores, or 'identity asserters', which can determine user identity from data embedded in a request, such as X.509 certificates or SAML tokens. The Oracle Access Manager Identity Asserter, used for OAM integration, is explained in WebLogic Configuration .

The default security realm has one authenticator and two identity asserters:

The **DefaultAuthenticator** handles users in the internal WebLogic store.

Each Authenticator provider has a control flag setting. The flag determines the part that the provider plays in authenticating a user. There are four possible flag values, but the two of importance here are:

REQUIRED: authentication in this provider must succeed for the overall process to complete.

SUFFICIENT: if authentication in this provider succeeds, the overall process is successful, as long as no other authenticator has the **REQUIRED** control flag.

The **DefaultAuthenticator** is pre-configured with the control flag set to **REQUIRED** so that it is required to succeed even if an LDAP authentication provider is configured. Part of the process of configuring LDAP support is to change this value to **SUFFICIENT**.

The order of providers is also important. When more than one authenticator is enabled and configured with the correct control flags, a user supported by any of the authenticators is able to login to the WebLogic Remote Console. However EDQ has access to the first authenticator in the list only; users from authenticators lower in the list will not be recognized. Therefore it is important that when an LDAP authenticator is added, it is moved to the top of the list. This is also covered below.

Configuring WebLogic to use LDAP

This section provides information about prerequisites and the procedure to configure WebLogic.

Prerequisites

Before beginning the configuration process, the following information must be gathered:

- The type of LDAP server in use. WebLogic provides authenticators for several types of LDAP server, including Microsoft Active Directory, OpenLDAP and Oracle Internet Directory. There is also a generic LDAP authenticator which can be used with servers not supported directly (such as AD-LDS).
- The host name and port of the LDAP server. If your environment contains multiple servers for high availability, you can use more than one host in the configuration. The default LDAP port is 389.
- The identity and password of an LDAP user which can connect and perform searches. The user identity is normally a full Distinguished Name (DN) but Active Directory also allows shorter forms.
- The locations in the LDAP tree (base DNs) where users and groups can be found.
- The LDAP attribute on a user record which identifies the user on login. Whilst most LDAP user schemas have standard user name attributes, organizations can choose to use others.
- The LDAP attribute on a group record that identifies the group. In most installations, this can be the group 'Common Name' (cn).
- The name of an LDAP group containing all the users that will be working with EDQ. The group is used to filter the list of users presented for EDQ issue assignment, etc. Without this filter, every user in the LDAP server would be presented, and this is generally not recommended.

Integrating with Active Directory

The rest of this section will cover the detailed steps for an integration with Active Directory. The steps are broadly similar for other types of LDAP server. Initial configuration will not use SSL.

In the walkthrough, these settings are used:

- **Active Directory Domain:** EXAMPLE.COM
- **Domain Controller (LDAP server)**: dc1.example.com. The default, non-SSL, port 389 will be used.
- **LDAP user**: *cn=netuser,cn=users,dc=example,dc=com.* Assuming that the AD username for this user is 'netuser' then you can also use netuser@example.com or example\netuser.

- **User base DN**: *dc=example,dc=com*. Here the base is the root of the full LDAP tree. If all the users are contained in a subtree, you could use something like: *cn=users,dc=example,dc=com*.
- **Group base DN:** *dc=example,dc=com*. Again a subtree could be used if suitable.
- **User name attribute:** *sAMAccountName*. This is the standard AD attribute which stores the short login name. In the Active Directory Users and Computers application this is displayed as the pre-Windows 2000 login name.
- **Group name attribute**: *cn*
- **All EDQ users group**: *edqusers*

EDQ Configuration

This section provides the procedure to configure EDQ.

User Group

To define the group containing EDQ users, follow the steps below:

1. Logon to the server hosting the EDQ domain and navigate to the EDQ 'local' configuration directory.

   This is located at: *DOMAIN_HOME/config/fmwconfig/edq/oedq.local.home*.

2. Create a local security directory and copy the default login configuration to this directory:

   ```
   $ mkdir security
   $ cp ../oedq.home/security/login.properties security/
   ```

3. Edit the new local configuration at *security/login.properties* and add the line:

   ```
   opss.prof.defaultusergroup = edqusers
   ```

Permissions

The pre-defined group mapping in *login.properties* maps the external 'Administrators' group to the EDQ 'Administrators' group. This is a valid mapping when using the internal WebLogic user store because the internal Administrators group contains the standard 'weblogic' administration console user.

However the Administrators group in Active Directory contains domain level administrators and it is unlikely that users in this group will use or administer EDQ. To add additional group mappings from external EDQ relevant groups using the EDQ web console, it is necessary to log in to EDQ as an administrator. This will be possible with the default pre-defined group mapping only an Active Directory administrator is available. To allow other users to become EDQ administrators, there are two approaches:

- Edit the pre-defined mapping:

  Edit the local *login.properties* and change the xgmap line to map a different LDAP group to the EDQ Administrators group. For example if a group named 'edqadmins' has been created in Active Directory to contain EDQ administration users, edit the line to read:

  ```
  opss.xgmap = edqadmins -> Administrators
  ```

- Enable the 'internal' realm:

Edit the local *login.properties* and change the realms line to read:

```
realms = internal, opss
```

This will enable the EDQ internal user store and you can log in as user 'dnadmin' and setup the necessary external group mappings using the web console. Once the mappings are complete you can disable the internal realm again.

Once these changes are complete you can start the EDQ managed server and login in as an Active Directory user. Note that external group mappings must be complete to grant the necessary permissions before application logins can succeed.

Filtering Groups

LDAP stores in large organizations may include thousands of distinct groups and with no additional filtering, all of these will appear in the external group mapping list on the EDQ web console. To filter the group list to a manageable size, you can add an LDAP group filter to *login.properties*.

Ensure to make a copy of this in the local configuration area before adding the filter. To add the filter, edit and add the line:

```
opss.prof.groupfilter = LDAPGROUPFILTER
```

where *LDAPGROUPFILTER* is an LDAP-style filter to select groups. The filter uses generic OPSS attribute names instead of LDAP-specific attributes. Use 'name' to refer to the name of the group. The filter must return any groups used in pre-defined mappings.

In the EXAMPLE.COM setup all the groups used with EDQ start with the prefix 'edq' so the filter would be:

```
opss.prof.groupfilter = (name=edq*)
```

Using SSL to connect to LDAP

If you wish to secure connections to the LDAP server by using SSL, tick the SSL Enabled check box on the Provider Specific tab for the LDAP provider, and enter the SSL port (normally 636).

> **Note:**
>
> In current versions of WebLogic, if you make changes to the Provider Specific page after initial configuration, you will need to enter the LDAP password again. The original password is obfuscated but unfortunately this version is then saved.

Then to enable successful connections from WebLogic to the LDAP server, so that the list of users and groups can be displayed, and you can login to WebLogic as an LDAP user, you will need to add the LDAP server certificate or root CA to the trust store of the JRE used to run WebLogic. Use the Java keytool command to update the JRE's cacerts file. Documentation for keytool can be found at:

https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html

Unfortunately the OPSS library used by EDQ for WebLogic authentication does not use the cacerts store. Instead a new keystore must be created. Stop the EDQ managed server but

leave the Administration server running. Login to the server running WebLogic and set these environment variables:

```
JAVA_HOME=java install location
WL_HOME=WLSHOME/wlserver
ORACLE_HOME=WLSHOME
```

and then execute:

```
$ WLSHOME/oracle_common/bin/libovdconfig.sh -host HOST -port PORT -userName weblogic -
domainPath DOMAIN_HOME -createKeystore
```

Here WLSHOME is the WebLogic installation directory; HOST is the name of the host running the Administration Server, PORT is the Administration port (typically 7001) and DOMAIN_HOME is the location of the WebLogic domain.

The command will prompt for the password for the weblogic user and then for a password for the new keystore. This can by anything.

On successful completion, a new keystore is created at:

```
DOMAIN_HOME/config/fmwconfig/ovd/default/keystores/adapters.jks
```

Use the Java keytool to add the server or CA certificate to the newly created keystore:

```
$ keytool -import -keystore DOMAIN_HOME/config/fmwconfig/ovd/default/keystores/
adapters.jks -alias ALIAS -file CERTFILE
```

Replace **ALIAS** with the keystore identifying the certificate and replace **CERTFILE** with the location of the certificate file.

The command will prompt for the keystore password; enter the password you specified when the keystore was created.

Restart the EDQ managed server and LDAP users should be able to login. Traffic between WebLogic and LDAP will be protected by SSL.

7

Configuring External User Management (LDAP) directly with EDQ

This chapter provides information on integration to LDAP with EDQ. It includes the following sections:

- Integrate EDQ with LDAP
  This section provides information about prerequisites and the procedure to integrate EDQ with LDAP.

Integrate EDQ with LDAP

This section provides information about prerequisites and the procedure to integrate EDQ with LDAP.

If you are not using WebLogic, or wish to use more than one LDAP store, you can integrate EDQ directly with LDAP using settings in *login.properties*. This may also be convenient if WebLogic does not provide a pre-defined Authenticator for your type of LDAP server. For example, AD-LDS, the lightweight LDAP server provided on Windows, is not compatible with Active Directory and you cannot use the WebLogic **ActiveDirectoryAuthenticator**.

This section covers the settings required in *login.properties* for simple LDAP integration. Kerberos support, used for Windows integrated authentication, is covered in Configuring EDQ to support Windows Integrated Authentication (Kerberos) .

- Prerequisites
- Integrating with Active Directory

Prerequisites

Before beginning the configuration process, the following information must be gathered:

- The type of LDAP server in use. EDQ has built-in profiles for Active Directory, Oracle Internet Directory, OpenLDAP (using the **inetOrgPerson** user schema) and servers using the RFC 2307 (Posix) style user schema.
- The host name and port of the LDAP server. If your environment contains multiple servers for high availability, you can use more than one host in the configuration. The default LDAP port is 389.
- The identity and password of an LDAP user which can connect and perform searches. The user identity is normally a full Distinguished Name (DN) but Active Directory also allows shorter forms.
- The base DN of the LDAP tree. Where possible this is determined automatically from the LDAP RootDSE **defaultNamingContext** attribute, but if this is not available, it must be provided in *login.properties*.
- The name of an LDAP group containing all the users that will be working with EDQ. The group is used to filter the list of users presented for EDQ issue assignment, etc. Without

this filter, every user in the LDAP server would be presented, and this is generally not recommended.

Integrating with Active Directory

The rest of this section will cover the detailed steps for an integration with Active Directory. The steps are broadly similar for other types of LDAP server. Initial configuration will not use SSL.

In the walkthrough, these settings are used:

- **Active Directory Domain:** EXAMPLE.COM
- **Domain Controller (LDAP server)**: dc1.example.com. The default, non-SSL, port 389 will be used.
- **LDAP user**: *cn=netuser,cn=users,dc=example,dc=com.* Assuming that the AD username for this user is 'netuser' then you can also use netuser@example.com or example\netuser.
- **Base DN**: *dc=example,dc=com*. When using Active Directory this is determined automatically from the RootDSE.
- **All EDQ users group**: *edqusers*

> **Note:**
>
> Ensure that you have a *login.properties* in the security directory in the EDQ 'local' configuration area. Tomcat installs do not have a default *login.properties* so just create a new empty file.

- **Global Settings:**

  In this example the Active Directory realm will be named 'ad'.

  ```
  realms                    = internal, ad
  gss                       = false
  ldap.prof.useprimarygroup = false
  ```

  Define the realm list. The EDQ internal realm is used so that the built-in administrator user 'dnadmin' can be used to set up external group mappings in the EDQ web console. The final setting indicates that the 'primary group' of the user should not be considered in determining group membership. For Active Directory this is always the 'Domain Users' group, containing every user and would not be used for EDQ.

- **Realm name:**

  ```
  ad.realm = EXAMPLE.COM
  ```

  Set the name of the Active Directory domain. If more than one realm is defined in *login.properties*, a dropdown selector for the realm is present on the EDQ login screens. If you wish this to contain more explanatory text, add a 'label' property:

  ```
  ad.label = Example, Inc. Active directory
  ```

- **LDAP server settings:**

  ```
  ad.ldap.server = dc1.example.com
  ad.ldap.auth   = simple
  ad.ldap.user   = cn=netuser,cn=users,dc=example,dc=com
  ad.ldap.pw     = <password for the user>
  ```

Set the server name and user information. If the server is omitted EDQ attempts to determine it automatically by a DNS lookup for ldap service records. If the server running EDQ is configured to use the Active Directory domain controllers for DNS, then these records should be setup automatically. Otherwise specify the server in *login.properties*. Multiple servers can be specified using a comma or space separated list:

```
ad.ldap.server = dc1.example.com, dc2.example.com:2342
```

To configure encryption for connections between EDQ and LDAP, add:

```
ad.ldap.security = ssl or tls or tlsverify
```

If you specify ssl as the value then a connection is made to the SSL port of the LDAP server (normally 636). In this case the certificate used by the LDAP server must be trusted by the JRE running EDQ. If the certificate was not created by a recognized provider, add the server or Root CA certificate to the JRE cacerts trust store using the keytool command. See: https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html

If you specify tls as the property value then a connection is made to the normal non-SSL LDAP port and a StartTLS command is then issued to switch the connection to use TLS encryption. All sensitive information such as passwords is sent after the switch to TLS. In this case, EDQ applies relaxed checks to the server certificate - it need not be trusted by the JRE. If you are concerned about spoofing on the internal network, use SSL for extra checks; otherwise using TLS simplifies the configuration. Alternatively, use tlsverify which uses StartTLS and does check the certificate.

- **Configure User Authentication**:

  This set of properties defines how a username and password presented to EDQ are verified in LDAP. This is a two-stage process. First a search is done to locate the user record identified by the username. For Active Directory this is done by searching for a user with the **sAMAccountName** matching the username or the **userPrincipalName** matching username@DOMAIN. These Active Directory attributes represent the 'short' and 'long' Windows login names.

  If the user record is not found, authentication does not succeed. Then there are three different methods in which the password can be verified:

  – **bind**: Attempt to connect to LDAP using the discovered user identity and the password. If this does not succeed, authentication fails.

  – **password**: read the hashed password from the user record and compare it with the supplied password. Most LDAP servers do not allow user password values to be read and this method is rarely applicable. It is sometimes used with Posix style user schemas.

  – **compare**: Use an LDAP compare operation to check the password. This method can be used with Oracle Internet Directory and can be preferable to 'bind' because it does not involve creating a new session on the LDAP server.

  The default method is 'bind' and this is always used with Active Directory.

```
ad.auth            = ldap
ad.auth.bindmethod = simple
ad.auth.binddn     = search: dn
```

  The 'auth' property sets LDAP-based authentication; other settings are possible if using advanced methods such as Kerberos. The 'bindmethod' property sets the connection method - 'simple' sends the user name and password in clear. The 'binddn' property sets the identity used in the connection attempt; here the DN (Distinguished Name) found from the initial user search is used.

An alternative bind method is digest-md5 which sends the password in hashed form rather than clear. To use this method, replace the bind properties with:

```
ad.auth.bindmethod = digest-md5
ad.auth.binddn     = search: sAMAccountName
```

The Windows user name is used for the connection instead of the Distinguished Name - this is required for digest-md5 connections.

To use LDAP compare for password verification, with Oracle Internet Directory, use these settings:

```
oid.auth        = ldap
oid.auth.method = compare
```

- **LDAP profile**:

  These settings define the LDAP schema in use and customise user lookup:

  ```
  ad.ldap.profile               = adsldap
  ad.ldap.prof.defaultusergroup = edqusers
  ```

  The 'adsldap' profile is used with Active Directory. Other pre-defined profiles are:

  – inetorgoidldap: Oracle Internet Directory, with **inetOrgPerson** users

  – inetorgopenldap: OpenLDAP, with **inetOrgPerson** users

  – rfc2307ldap: RFC 2307 (Posix) style user records

  The 'defaultusergroup' property defines the group containing all EDQ users.

  LDAP stores in large organizations may include thousands of distinct groups and with no additional filtering, all of these will appear in the external group mapping list on the EDQ web console. To filter the group list to a manageable size, you can add an LDAP group filter:

  ```
  ad.ldap.prof.groupsearchfilter = LDAPGROUPFILTER
  ```

  where **LDAPGROUPFILTER** is an LDAP-style filter to select groups. The filter must return any groups used in pre-defined mappings.

  In the EXAMPLE.COM setup all the groups used with EDQ start with the prefix ‘edq' so the filter would be:

  ```
  ad.ldap.prof.groupsearchfilter = (cn=edq*)
  ```

- **Putting it all together**:

  This is the complete *login.properties* used for the Active Directory integration example:

  ```
  # EXAMPLE.COM LDAP integration
  # ---------------------------

  realms                        = internal, ad
  gss                           = false
  ldap.prof.useprimarygroup     = false

  ad.realm                      = EXAMPLE.COM
  ad.ldap.server                = dc1.example.com
  ad.ldap.auth                  = simple
  ad.ldap.user                  = cn=netuser,cn=users,dc=example,dc=com
  ad.ldap.pw                    = <password for the user>
  ad.auth                       = ldap
  ad.auth.bindmethod            = simple
  ad.auth.binddn                = search: dn
  ```

```
ad.ldap.profile               = adsldap
ad.ldap.prof.defaultusergroup = edqusers
```

- **Integrating with more than one LDAP store**:

  To integrate with multiple LDAP stores, simply define multiple realms and include a block of settings for each:

  ```
  realms = internal, ad1, ad2, oid
  ad1.realm = …
  …
  ad2.realm = …
  …
  oid.realm = …
  …
  ```

  If you are using an Active Directory Forest with multiple domains, each domain must be defined as a separate realm in *login.properties* - EDQ cannot make cross-domain references. For example:

  ```
  realms = internal, aduk, asus
  aduk.realm       = UK.EXAMPLE.COM
  aduk.ldap.server = DC1.UK.EXAMPLE.COM
  …
  adus.realm       = US.EXAMPLE.COM
  asus.ldap.server = DC1.US.EXAMPLE.COM
  …
  ```

8

Integrating EDQ with Azure Active Directory

This chapter describes how to integrate Azure Active Directory (AD) with Oracle Enterprise Data Quality (EDQ).

This chapter includes the following sections:

- Registering the EDQ Application in Azure AD
  EDQ can integrate with Azure AD as an identity store. You need to register the EDQ app in Azure AD so that user and group queries can use the Microsoft Graph v1.0 REST APIs to access data in Azure AD.
- Enabling OpenID Connect SSO in the Application
  You need to configure the redirect URIs to the EDQ servers to enable SSO using OpenID Connect. You can add the redirect URIs of any additional EDQ servers so that the single app registration can support multiple servers.
- Enabling Multiple URI Redirects for OpenID Authentication
  The OpenID integration framework is enhanced in EDQ 12.2.1.4.3 to allow configuration of multiple redirect URIs based on the incoming host name. This allows login through the load balancer or to a specific host behind the load balancer.
- Editing the EDQ login.properties File
  User authentication in EDQ is configured using a file named *security/login.properties* in the EDQ configuration area. The file should be created in the "local" configuration. If the directory "security" does not exist in this location you must create it manually. You need to edit *login.properties* and define a realm for Azure AD.
- Enabling OAuth2 Bearer Authentication for Web Services
  Authentication in Azure AD is based on SSO mechanisms such as OpenID Connect and SAML. There is no support for programmatic authentication with a username and password.

Registering the EDQ Application in Azure AD

EDQ can integrate with Azure AD as an identity store. You need to register the EDQ app in Azure AD so that user and group queries can use the Microsoft Graph v1.0 REST APIs to access data in Azure AD.

To register the EDQ app in Azure AD,

1. In the Azure Portal, navigate to the Azure AD instance (named **Default Directory** by default).
2. In the Manage menu on the left, click **App Registrations**.
3. Click **New Registration** and do the following:
   - Enter a name. For example, `edqgraph`.
   - Select **Accounts in this organizational directory only (Default Directory only - Single tenant)**
4. Click **Register** to create the new app registration.

The Overview screen is displayed.

5. Copy and save the Application (client) ID. This ID is used as the Client ID for OAuth2 client credentials authentication.
6. Click **Client credentials** on the top left of the screen.
7. Click **New client secret** and do the following:
   - Enter a description.
   - Select the expiry date for the client secret.
   - Click **Add**.
   - Copy and save the new secret value.

   > **Note:**
   >
   > Ensure that you update the EDQ configuration before the secret expires.

8. Click **API permissions** and then click **Add a permission**.
9. Under Microsoft APIs, select **Microsoft Graph**.
10. The application runs as a background service and not as a real signed-in user. Do the following to set the permissions:
    - Select **Application Permissions**.
    - Expand **Application** and select **Application.Read.All**.
    - Expand **Group** and select **Group.Read.All**.
    - Expand **User** and select **User.Read.All**.

    > **Note:**
    >
    > The single permission **Directory.Read.All** covers user, group, and application access.

11. On the API Permissions page, click **Grant admin consent for Default Directory**.

    This allows the application to use the permissions without further consent prompts.

You can now now use the application to request OAuth2 client credentials for use with Microsoft Graph REST API calls.

Enabling OpenID Connect SSO in the Application

You need to configure the redirect URIs to the EDQ servers to enable SSO using OpenID Connect. You can add the redirect URIs of any additional EDQ servers so that the single app registration can support multiple servers.

To enable SSO using OpenID Connect, follow the steps below:

1. In the Azure Portal, navigate to the Azure AD instance (named **Default Directory** by default).
2. In the Manage menu on the left, click **App Registrations** and select the app you newly registered in Registering the EDQ Application in Azure AD.

3. On the Overview screen, click **Add an Application URI** in the top right of the screen.
4. On the next screen click **Add a platform**.
5. Select **Web** as the platform type.
6. Enter the call back URI for the EDQ instance in the following format:

   `https://server/edq/oidc/callback`
   where `server` is the host name of your EDQ installation. This URI is used to support the login flow. If a web server such as Apache HTTPD or Nginx is being used as a front end or load balancer for EDQ, enter the name of that server.
7. Click **Configure** to save the URI. You can leave the other settings as is.
8. Click **Add URI**.
9. Enter the following URI for the EDQ instance logout flow:

   `https://server/edq/oidc/loggedout`
10. Click **Save** at the top to save the changes.

The app registration is now ready for SSO using OpenID connect.

Enabling Multiple URI Redirects for OpenID Authentication

The OpenID integration framework is enhanced in EDQ 12.2.1.4.3 to allow configuration of multiple redirect URIs based on the incoming host name. This allows login through the load balancer or to a specific host behind the load balancer.

You can set up logins to EDQ instances in the following ways:

- **Using automatic URI:** Set the redirect URL in *login.properties* to **auto**. In this case, the URL is derived from the HTTP request as `https://HOST/edq/oidc/callback`.
  For example, if the user refers to `https://lb.example.com/edq/` the redirect URI is constructed as `https://lb.example.com/edq/oidc/callback`. If the user refers to `https://instance1.example.com/edq/` the redirect URI is constructed as `https://instance1.example.com/edq/oidc/callback`. The host name is derived from reading the **X-Forwarded-Host** HTTP header. If **X-Forwarded-Host** is not present, then the **Host** header is used.
- **Using host to URI mappings:** Define a map from host name to URI in *login.properties*. In this case, the host name is compared against each mapping and the first match is used. In addition to the special value **auto**, you can use **none** to indicate that no redirection should take place. This allows normal internal logins to the EDQ Launchpad and other applications. For more control, the host name can be a regex, prefixed with ~. If there is no match, the default redirect URI is used.
  For example,

  ```
  azure.extra.oidc.redirect_map = localhost -> none, \
                                          testhost -> none, \
                                          myalias -> https://
  lb.example.com/edq/oidc/callback,
                                          ~host\\d+\.example\.com -> auto
  ```

  In this example:

  – References to `https://localhost/edq/` or `https://testhost/edq/` do not redirect and give direct login access.

  - References to `https://myalias/edq/` use the load balancer redirect.
  - References to `https://host23.example.com/edq/` redirect to `https://host23.example.com/edq/oidc/callback`.

Editing the EDQ login.properties File

User authentication in EDQ is configured using a file named *security/login.properties* in the EDQ configuration area. The file should be created in the "local" configuration. If the directory "security" does not exist in this location you must create it manually. You need to edit *login.properties* and define a realm for Azure AD.

Before you edit *login.properties*, note the following:

- Ensure that EDQ supports HTTPS connections.
- The Marketplace images implement SSL in the Apache HTTPD front end, so no further actions are necessary.
- If you are connecting to EDQ without a web front end, HTTPS must be configured in the application server.

Refer to the WebLogic or Tomcat documentation for more details.

Set *login.properties* as follows:

```
# Realms
realms = azure
# yourdomain.com users and groups at Azure AD
azure.realm                    = yourdomain.com
azure.label                    = Azure AD
azure.type                     = azure
azure.clientid                 = clientid
azure.clientsecret             = clientsecret
azure.tenant                   = tenant ID
azure.proxy                    = <proxy server>:port
azure.prof.groupfilter         = startsWith(displayName, 'edq-')
azure.prof.userdisplayname     = userName
azure.prof.defaultusergroup    = edq-users
azure.extra.oidc               = true
azure.extra.oidc.redirect_uri = https://server/edq/oidc/callback
azure.xgmap                    = edq-admins -> Administrators
```

Where,

- **realm** is the custom domain for your Azure AD instance.
  - **tenant** is your Azure tenant ID.
  - **clientid** and **clientsecret** are the values for your Azure AD application.
  - **proxy** is the host and port of the proxy server required to access the internet from your EDQ server. If a proxy is not required, omit this setting.
  - **groupfilter** is a Microsoft Graph *$filter* expression to select the groups used with EDQ. The value shown in the example returns all groups whose name starts with `edq-`. Refer to the Microsoft documentation for details about filtering. Omit this setting if you want all groups to be visible.

- **defaultusergroup** is the name of the group containing the users who work with EDQ. Here the group `edq-users` has been used as an example.
- **extra.oidc.redirect_uri** is the redirect URI entered in the Azure AD application. The URIs must match exactly.
- **xgmap** is the bootstrap group mapping required for admin login. Here as an example, the Azure group `edq-admins` is mapped to the EDQ Administrators group. You can set other group mappings in the EDQ console.

After the server is restarted, references to the EDQ launchpad will redirect to the Microsoft login page.

Enabling OAuth2 Bearer Authentication for Web Services

Authentication in Azure AD is based on SSO mechanisms such as OpenID Connect and SAML. There is no support for programmatic authentication with a username and password.

If Azure AD is configured as the sole identity store for EDQ, the following features are not available:

- Basic authentication for calls to EDQ web services.
- Authentication for JMX connections.
- Explicit login to the EDQ launchpad or Java applications.
- Connections to the built-in SSH (SFTP/SCP) server

To continue to use JMX (JConsole, JMXTools) connections to EDQ, the recommended approach is to enable the "internal" realm in *login.properties*:

```
realms = internal, azure
```

and use internal users such as "dnadmin" for JMX authentication.

For web service authentication with Azure AD, EDQ supports authentication using OAuth2 Bearer access tokens. A caller will use the client credentials or authorization code flows to acquire an access token and then pass this to EDQ in an **Authorization** header. For example,

```
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Im5PbzNa ......
```

User credentials obtained from the authorization code flow are mapped using the user identity in the same way as normal logins.

See the Azure documentation for more information about access tokens and token validation.

Client credentials are mapped using application roles, as described in the following sections:

- Configuring the Azure Application to Support Bearer Authentication with EDQ
- Creating Client Applications in Azure AD
- Editing the EDQ login.properties File to Map Roles
- Configuring Application Display in EDQ

Configuring the Azure Application to Support Bearer Authentication with EDQ

To configure the Azure application to support Bearer authentication with EDQ, follow the steps below:

1. In the Azure Portal, navigate to the Azure AD instance (named **Default Directory** by default).
2. In the Manage menu on the left, click **App Registrations**.
3. On the Overview screen, click **Add an Application URI** in the top right of the screen and click **Set**.

   A default value such as *api://1b843028-71bd-47e7-b62e-7859c0a96627* is provided, but you can use any value after `api://` as long as it is unique in the Azure Directory instance.

   > **Note:**
   >
   > For ease of reference, we will assume the value as *api://edq* for the rest of this topic.

4. Click **Save** to store the URI. You can leave the other settings as is.

   OAuth2 calls to request client credentials tokens must use the scope *api://edq/.default*
5. Applications that call EDQ web services are authorized using application roles or group membership. To define roles for the existing Azure application, click **App Roles** and then click **Create app role**.
6. Enter a display name that is used in the portal, a value which is used for configuration, and a description.
7. Select **Applications** as the **Allowed member types**. For example, roles might be defined for simple web service calls and for system administration.

Creating Client Applications in Azure AD

Calls to EDQ using OAuth2 client credentials are made on behalf of Azure Applications.
To create an application, follow these steps:

1. Create an app registration as described in Registering the EDQ Application in Azure AD.
2. Copy and save the Application (client) ID and client secret.
3. If you are using App Roles for authorization in EDQ, click **API permissions** and then click **Add a permission**.
4. Select **My APIs** and click the graph+SSO application.
5. Select the required roles and then Grant admin consent.

   The new client application is now associated with the selected roles in the graph+SSO application.
6. If you are using Group membership for application authorization in EDQ, select the required Azure group and add the new application as a member.

Editing the EDQ login.properties File to Map Roles

Add settings in *login.properties* to verify the audience and issuer claims in access tokens, and to add *rolemap* settings to map application roles to EDQ groups. Add the following to *login.properties*:

```
azure.extra.oauth2.token.aud = api://edq
azure.extra.oauth2.token.iss = https://sts.windows.net/TENANTID/
azure.extra.oauth2.rolemap   = administration -> Administrators, callers ->
Data Stewards
```

The `rolemap` example setting maps the administration role to the EDQ Administrators group and the callers role to the EDQ Data Stewards group. If you are using groups for application authorization in EDQ, the rolemap setting is not required.

Configuring Application Display in EDQ

A client application that makes a call to EDQ services using OAuth2 is allocated an internal "user" ID in the same way as for standard users. In applications, such as Case Management, that display historical information regarding user updates, an application is displayed as *App: NAME* where *NAME* is the display name of the application in Azure AD. The display format may be configured using the `appusername` profile property in *login.properties*:

```
azure.prof.appusername = OAuth2 application: {0}
```

In the property value {0} is replaced by the application name.

To enable application name display, the list of users retrieved from Azure AD is augmented with a query for applications, which uses the list servicePrincipals API. By default, applications that have never made a client call to EDQ are not included. When an application makes an initial call to EDQ, the name is not immediately available for display in Case Management. If this is a problem, the profile `showallapps` property can be set such that all applications are retrieved:

```
azure.prof.showallapps = true
```

The `appfilter` profile property can used to filter applications by name:

```
azure.prof.appfilter   = startsWith(displayName, 'Studio')
```

If OAuth2 client calls are not used for an installation, application listing can be disabled by setting the `appfilter` property to the special value 'none':

```
azure.prof.appfilter = none
```

In this case, the Graph **Application.Read.All** permission is not required.

9

Integrating EDQ with Oracle Identity Cloud Service

This chapter describes how to integrate Oracle Identity Cloud Service (IDCS) with Oracle Enterprise Data Quality (EDQ).

> **Note:**
>
> This feature is applicable for EDQ 12.2.1.4.2 and later releases.

This chapter includes the following sections:

- Creating an IDCS Application
  EDQ uses the IDCS REST APIs to authenticate credentials and retrieve lists of users and groups. These APIs are authenticated with the OAuth2 client credentials flow using tokens obtained from an IDCS application.
- Configuring the EDQ login.properties File
  You need to edit *login.properties* and define a realm for IDCS.
- Enabling SSO using OpenID Connect
  You can configure EDQ for SSO with IDCS using OpenID Connect with OAuth2. When a user visits the EDQ Launchpad they are redirected to the IDCS login page. After successful login the user is redirected back to EDQ. Applications are also logged in automatically using the IDCS identity.
- Enabling OAuth2 Bearer Authentication for Web Services
  If OpenID connect SSO is configured, EDQ supports web service authentication using OAuth2 Bearer access tokens. A caller will use the client credentials or authorization code flows to acquire an access token and then pass this to EDQ in an **Authorization** header.

Creating an IDCS Application

EDQ uses the IDCS REST APIs to authenticate credentials and retrieve lists of users and groups. These APIs are authenticated with the OAuth2 client credentials flow using tokens obtained from an IDCS application.

To create an application to use with EDQ, follow these steps.

1. Login to the IDCS console as an administrator.

   For logins from the main OCI console, you can find the link to the IDCS console by navigating to **Identity** > **Federation** > **OracleIdentityCloudService**. The console link will be similar to the following:
   `https://idcs-xxxxxxxxxxxxxxxxxxxxxx.identity.oraclecloud.com/ui/v1/adminconsole`

The portion of the URL before `.identity.oraclecloud.com` is the IDCS instance identifier and is required when configuring EDQ. In this example it is `idcs-xxxxxxxxxxxxxxxxxxxx`

2. On the **Applications and Services** page click **+ Add** to create a new Application.
3. Select **Confidential Application**.
4. Enter a name and, optionally, a description. No other information is required on the first page.
5. Click **Next** .
6. Select **Configure this application as a client now**.
7. In the Authorization section, select **Client Credentials** from the Allowed Grant Types.
8. Under **Grant the client access to Identity Cloud Service Admin APIs** click **Add**.
9. Select **Application Administrator** and **Authenticator Client**.
10. Click **Next** through the remaining steps, and then click **Finish**.
11. Copy and save the Client ID and Client Secret values that are displayed. You can also retrieve these values from the Application Configuration in the General Information section.
12. Click **Activate** to enable the application for use.

Configuring the EDQ login.properties File

You need to edit *login.properties* and define a realm for IDCS.

Set *login.properties* as follows:

```
# Realms
realms            = internal, idcs

idcs.realm        = IDCS
idcs.label        = IDCS
idcs.type         = idcs

idcs.clientid     = 61a155a32c39486c95a18ed1de7cc934
idcs.clientsecret = 53d09389-3645-4963-b0aa-152dd7505e7f
idcs.instance     = idcs-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
# Add this if a proxy is required to reach https://idcs-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.identity.oraclecloud.com
#idcs.proxy = host:port
```

Enter the Client ID and Client Secret that was generated when the application was created, and the IDCS instance identifier portion of the URL.

Restart the EDQ server. To verify the integration check the External Groups page on the EDQ Launchpad.

- Additional Configuration

Additional Configuration

Many IDCS instances are configured to use the user's email address as the user name. By default external users in EDQ are identified by `username@REALM`. If IDCS is configured with

email addresses as user names, for a user named John Sheridan who works with Interstellar Alliance, for example, this would appear as `john.sheridan@interstellaralliance.org@IDCS`.

To remove the `@IDCS` portion from the user name, add the following to *login.properties*:

```
idcs.prof.userdisplayname = userName
```

To setup some bootstrap group mappings, use the **xgmap** property:

```
idcs.xgmap = EDQ admins -> Adminstrators
```

Members of the IDCS group EDQ admins will login to EDQ as members of the Administrators group. You can define more group mappings using the EDQ console.

To limit the set of users displayed in EDQ, add a default group assignment, for example:

```
idcs.prof.defaultusergroup = EDQ users
```

Enabling SSO using OpenID Connect

You can configure EDQ for SSO with IDCS using OpenID Connect with OAuth2. When a user visits the EDQ Launchpad they are redirected to the IDCS login page. After successful login the user is redirected back to EDQ. Applications are also logged in automatically using the IDCS identity.

To enable SSO using OpenID Connect, see the following sections:

- Configure the EDQ Application in IDCS
- Configuring EDQ for OpenID Connect SSO

Configure the EDQ Application in IDCS

You need to configure the redirect URIs to the EDQ servers to enable SSO using OpenID Connect.

> **Note:**
>
> IDCS supports multiple values for the redirect URLs. You can use the same application to support several EDQ instances. However, this section describes setting the redirect URIs using the IDCS console, which does not support the configuration of multiple values. You need to use the IDCS REST API to set the URLs for such cases.

1. Login to the IDCS console and navigate to the application you created in Creating an IDCS Application.
2. On the Configuration tab, open the **Client Configuration** section and select **Authorization Code** as an allowed grant type.
3. If your server does not have HTTPS enabled, select the **Allow non-HTTPS URLs** checkbox.

4. Set the **Redirect URL** to `https://yourserver/edq/oidc/callback` where `yourserver` is the full host name of your EDQ server. Include the port, if required.
5. Set the **Post Logout Redirect URL** to `https://yourserver/edq/oidc/loggedout`.
6. Save the changes made to the application.
7. Select the Users or Groups tabs to add users to the application.

   If you have a `defaultusergroup` defined in *login.properties*, the simplest approach would be to add the group so that all EDQ users can use the application.

Configuring EDQ for OpenID Connect SSO

Add these settings to *login.properties*:

```
idcs.extra.oidc    = true
idcs.extra.oidc.redirect_uri    = https://yourserver/edq/oidc/callback
```

The redirect_url value must match the Redirect URL entered for the application in **Configure the EDQ Application in IDCS**.

Restart EDQ. When you browse to the Launchpad you will be redirected to the IDCS login page.

EDQ 12.2.1.4.3 onwards you can configure *login.properties* to map the host name to multiple redirect URIs. See Enabling Multiple URI Redirects for OpenID Authentication for more information.

Enabling OAuth2 Bearer Authentication for Web Services

If OpenID connect SSO is configured, EDQ supports web service authentication using OAuth2 Bearer access tokens. A caller will use the client credentials or authorization code flows to acquire an access token and then pass this to EDQ in an **Authorization** header.

For example,

```
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Im5PbzNa ......
```

User credentials obtained from the authorization code flow are mapped using the user identity in the same way as normal logins.

See the IDCS documentation for more information about access tokens and token validation.

Client credentials are mapped using request scopes, as described in the following sections:

- Configuring Application Scopes
- Configuring EDQ

Configuring Application Scopes

To configure the IDCS application to support Bearer authentication with EDQ, define a protected resource and associated scopes. The scopes can be mapped to EDQ groups, allowing different client applications to request different permissions.
To define a resource, follow these steps:

1. Navigate to the IDCS application.
2. Click the Configuration tab and open the resources area.
3. Select **Register Resources**.
4. Enter a value in the **Primary Audience** field. The value can be any string ending in a colon. For example, *urn:edq:*
5. In the Scopes section, click Add to add scopes corresponding to the required EDQ access. For example, *admininstration* and *callers*.
6. Configure the client application to request access tokens using the configured scope. For example, **urn:edq:administration** or **urn:edq:callers**.

Configuring EDQ

In *login.properties* add settings to verify the Audience and map scopes to groups:

```
idcs.extra.oauth2.token.aud    = urn:edq:
idcs.extra.oauth2.scopemap    = administration -> Administrators, callers ->
Data Stewards
```

Where,

- **token.aud** enables verification of the `aud` claim in access tokens.
- **scopemap** maps the administration scope to the EDQ Administrators group and the callers scope to the EDQ Data Stewards group.

10

Integrating EDQ with Cognito User Pools

This chapter describes how to integrate Cognito user pools with Oracle Enterprise Data Quality (EDQ).

> **Note:**
>
> This feature is applicable for EDQ 12.2.1.4.3 and later releases.

This chapter includes the following sections:

- Creating the User Pool
  EDQ can integrate with Amazon Cognito user pool as an identity store. User and group logins happen using Amazon Cognito REST APIs. To use EDQ with Amazon Cognito, you need to first create a user pool using the Amazon Cognito console.
- Creating an IAM User for REST API Calls
  Calls to the Cognito REST APIs are authenticated with standard AWS request signatures. We recommend that you create an IAM user with only the permissions needed by EDQ. First create an IAM policy with the required permissions. Create an IAM user and attach the new policy, either directly or via a group. Create credentials for the user and note the key and associated secret.
- Editing the EDQ login.properties File
  User authentication in EDQ is configured using a file named *security/login.properties* in the EDQ configuration area. The file should be created in the "local" configuration. If the directory "security" does not exist in this location you must create it manually. You need to edit *login.properties* and define a realm for Cognito.
- Enabling OAuth2 Bearer Authentication for Web Services
  EDQ integration with Cognito user pools is based on SSO mechanisms such as OpenID Connect SSO authentication. There is no support for programmatic authentication with a username and password.

Creating the User Pool

EDQ can integrate with Amazon Cognito user pool as an identity store. User and group logins happen using Amazon Cognito REST APIs. To use EDQ with Amazon Cognito, you need to first create a user pool using the Amazon Cognito console.

To create an Amazon Incognito user pool, follow these steps.

1. Login to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. In the top-right corner of the page, choose **Create a user pool** to open the wizard. You can also use an existing user pool.
4. Choose your password policy, multi-factor authentication (MFA) requirements, and user account recovery options.

5. Configure user sign-up experience to determine how new users will verify their identities when signing up. Ensure that email and name are selected as required attributes. Cognito does not provide a default "Organization" attribute. If this is required for EDQ, create a custom attribute.
6. Configure user message delivery as required to send email and SMS messages to your users for sign-up, account confirmation, MFA, and account recovery. There is no need to configure message delivery if users will be created by administrators with preset initial passwords.
7. In **Integrate your app**, name your user pool, configure the hosted UI, and create an app client. For more information, see Add an App to Enable the Hosted Web UI
8. In **Integrate your app**, select **Confidential client** as the type, and ensure that **Generate a client secret** is selected.
9. In **Allowed callback URLs** enter `https://server/edq/oidc/callback`

   If you wish to use an existing user pool, you can add a new App client supporting the Authorization code grant flow and configure the callback URLs. You can add additional callback URLs after the pool is created to allow multiple EDQ instances to be supported.
10. In **Advanced app client settings**, add `https://server/edq/oidc/loggedout` as an allowed sign-out URL.

    If you are using WebLogic, it may be necessary to include the port number in the sign-out URL. Add *https://server:443/edq/oidc/loggedout* as an additional sign-out URL.
11. Review your choices in the summary screen and click **Create user pool** to proceed.

Creating an IAM User for REST API Calls

Calls to the Cognito REST APIs are authenticated with standard AWS request signatures. We recommend that you create an IAM user with only the permissions needed by EDQ. First create an IAM policy with the required permissions. Create an IAM user and attach the new policy, either directly or via a group. Create credentials for the user and note the key and associated secret.

Editing the EDQ login.properties File

User authentication in EDQ is configured using a file named *security/login.properties* in the EDQ configuration area. The file should be created in the "local" configuration. If the directory "security" does not exist in this location you must create it manually. You need to edit *login.properties* and define a realm for Cognito.

Before you edit *login.properties*, click App client information in the user pool summary and copy the Client ID and Client secret.

Note the following:

- Ensure that EDQ supports HTTPS connections.
- The Marketplace images implement SSL in the Apache HTTPD front end, so no further actions are necessary.
- If you are connecting to EDQ without a web front end, HTTPS must be configured in the application server.

Refer to the WebLogic or Tomcat documentation for more details.

Set *login.properties* as follows:

```
# Realms

realms = cognito

cognito.realm                  = yourdomain.com
cognito.label                  = Cognito user pool
cognito.type                   = cognito

cognito.region                 = eu-west-1
cognito.pool                   = eu-west-1_z9wkvtsPx
cognito.proxy                  = proxyserver:port
cognito.access_key             = AKIAVWUKAYEFKZM7GFVU
cognito.access_secret          = lapirJ3l8n4+lD43TCDReFXexS....
cognito.prof.defaultusergroup  = edq-users
cognito.xgmap                  = edq-admins -> Administrators

cognito.extra.oidc             = true
cognito.extra.oidc.clientid    = 2p4st7jlu6ne49rchiump9ulqc
cognito.extra.oidc.clientsecret = ...
cognito.extra.oidc.redirect_uri = https://server/edq/oidc/callback
```

Where,

- **realm** is the domain you wish to use with your Amazon Cognito user pool.
- **region** is the user pool region.
- **pool** is the user pool ID.
- **proxy** is the host and port of the proxy server required to access the internet from your EDQ server. If a proxy is not required, omit this setting.
- **access_key** and **access_secret** to the access key and secret for the IAM user.
- **defaultusergroup** is the name of the group containing the users who work with EDQ. Here the group `edq-users` has been used as an example.
- **extra.oidc.clientid** and **extra.oidc.clientsecret** are the values for your user pool app client.
- **extra.oidc.redirect_uri** is the redirect URI entered in the Cognito application. The URIs must match exactly.
- **xgmap** is the bootstrap group mapping required for admin login. Here as an example, the Cognito group `edq-admins` is mapped to the EDQ Administrators group. You can set other group mappings in the EDQ console.

After the server is restarted, references to the EDQ launchpad will redirect to the Cognito login page. You can provide your own login page or customize the standard page using CSS.

- User Attribute Selection

User Attribute Selection

Internal and external users in EDQ are represented by a username, email address, full name (such as John Smith), telephone number, and organization name. Full name, telephone, and

organization are stored as "vcard" attributes, to allow for future expansion. When integrating with Cognito user pools these values are derived by reading attributes from user objects:

Value	Cognito user attribute	Overriding property in login.properties
username	Username	cognito.prof.usernameattr
email address	email	cognito.prof.vcard.email.pref
full name	name	cognito.prof.vcard.fn
telephone number	phone_number	cognito.prof.vcard.tel.work
organization		cognito.prof.vcard.org

Cognito does not have a standard Organization attribute. If you defined a custom attribute for this when the user pool was created, you can refer to it in *login.properties*. For example:

```
cognito.prof.vcard.org = custom:organization
```

vcard. properties can specify multiple attributes. The first attribute with a non-empty value is used as the result. For example, if there are custom organization and department attributes, you can set the following:

```
cognito.prof.vcard.org = custom:department custom:organization
```

You can also configure a **cognito.prof.userdisplayname** property. The value is an expression constructed from user attributes. The resulting value is used whenever a user is displayed in EDQ. If not set, **username@realm** is used. For example `john.smith@yourdomain.com`. If the user name is an email address, or you have only a single realm, you can override **cognito.prof.userdisplayname** to use a different attribute. For example, set the following to use the user name as the display name:

```
cognito.prof.userdisplayname = Username
```

The value is an expression so that you can specify more exotic values. For example, set the following to create values such as `john.smith (John Smith)`:

```
cognito.prof.userdisplayname = Username || ' (' || name || ')'
```

Enabling OAuth2 Bearer Authentication for Web Services

EDQ integration with Cognito user pools is based on SSO mechanisms such as OpenID Connect SSO authentication. There is no support for programmatic authentication with a username and password.

If Cognito is configured as the sole identity store for EDQ, the following features are not available:

- Basic authentication for calls to EDQ web services.
- Authentication for JMX connections.
- Explicit login to the EDQ launchpad or Java applications.
- Connections to the built-in SSH server

To continue to use JMX (JConsole, JMXTools) or SSH connections to EDQ, the recommended approach is to enable the "internal" realm in *login.properties*:

```
realms = internal, cognito
```

and use internal users such as "dnadmin" for authentication.

For web service authentication with Cognito, EDQ supports authentication using OAuth2 Bearer access tokens. A caller will use the client credentials or authorization code flows to acquire an access token and then pass this to EDQ in an **Authorization** header. For example,

```
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Im5PbzNaRHJPRFhFSzFqS1doWHNs......
```

User credentials obtained from the authorization code flow are mapped using the user identity in the same way as normal logins.

See the Cognito documentation for more information about access tokens and token validation.

Client credentials are mapped using custom roles, as described in the following sections:

- Configuring Custom Scopes in the User Pool
- Configuring Client Credentials Application
- Configuring EDQ
- Configuring Application Display in EDQ

Configuring Custom Scopes in the User Pool

Authorization for client credentials calls is done by mapping custom OAuth2 scopes to EDQ groups. The first step is to create the required custom scopes in the user pool.

To configure custom scopes in the user pool:

1. In the **Resource servers** section of the user pool properties click **Create resource server**.
2. Enter a Resource server name and Resource server identifier. The URI does not refer to a real server instance so you can use a simple URI such as `urn:edq`.
3. Click **Add custom scope**.
4. Enter a Scope name and description.
5. Click **Create**.

Configuring Client Credentials Application

After you have added the custom scopes in the user pool, you need to map them in the user pool properties.

To configure client credentials:

1. In the user pool properties, create a new client.
2. Select **Confidential client** and ensure that **Generate a client secret** is enabled.
3. Remove the callback URL and select **Client credentials** as the OAuth 2.0 grant type.

4. In the Custom scopes section, select the scope you defined with the resource server in Configuring Custom Scopes in the User Pool.

Configuring EDQ

In *login.properties* add **scopemap** settings to map scopes to EDQ groups:

```
cognito.extra.oauth2.scopemap = urn:edq/callws -> Data Stewards
```

Here **scopemap** maps the *urn:edq/callws* custom scope to the EDQ Data Stewards group.

Configuring Application Display in EDQ

A client application that makes a call to EDQ services using OAuth2 is allocated an internal "user" ID in the same way as for standard users. In applications, such as Case Management, which display historical information regarding user updates, an application is displayed as *App: NAME* where *NAME* is the display name of the application client. The display format may be configured using the `appusername` profile property in *login.properties*:

```
cognito.prof.appusername = OAuth2 application: {0}
```

Here the property value {0} is replaced by the application name.

To enable application name display, the list of users retrieved from Cognito is augmented with a query for application clients, which uses the `ListUserPoolClients` API. By default, application clients that have never made a client call to EDQ are not included. When an application makes an initial call to EDQ, the name is not immediately available for display in Case Management. If this is a problem, the profile `showapps` property can be set to "all" so that all applications are retrieved.

```
cognito.prof.showapps = all
```

To disable application listing, set `showapps` to "none". In this case, the IAM user does not need to have the `ListUserPoolClients` permission.

```
cognito.prof.showapps = none
```

11

Configuring EDQ Encryption

The encryption framework for EDQ now supports a pluggable module with implementations for OCI vaults and scripts as described in this chapter.

- EDQ Enhanced Encryption Overview
  To provide additional security, certain information stored in the EDQ configuration is encrypted.
- Configuring Pluggable Credentials Stores
  The user name and password credentials used in EDQ configuration are normally defined in properties files.
- Creating Custom Java Encryption and Credentials Store Modules
  The 12.2.1.4.4 and 14.1.2.0.0 versions of EDQ supports pluggable modules for encryption and credentials lookup. The following section describes how to implement modules using custom Java classes which is necessary if it is not possible to write a script to interact with some security store because native code is required.

EDQ Enhanced Encryption Overview

To provide additional security, certain information stored in the EDQ configuration is encrypted.

- Database passwords in `director.properties` (optional, Tomcat only)
- Data store passwords
- Stored credentials
- Web push notification Vapid private keys

In EDQ versions up to 12.2.1.4.3, encryption is performed using the AES cipher in ECB mode with PKCS5Padding. At the first startup, a new random key is generated and stored. In Tomcat, the key is stored in the file `localhome/kfile` and in WebLogic the key is stored in the FMW key store.

The following limitations are seen in this implementation:

- The default AES keysize (128 bits) is used. For additional security, 256 bits should be used.
- The ECB cipher mode does not use an initialization vector (IV) so encrypting the same string always produces the same result. This can make it easier for attackers to guess passwords if they can see the encrypted values.
- In Tomcat, the key file can be read by any user of the system.
- If a new install is created using the existing schemas, a new key file is generated and existing encrypted data in the database cannot be recovered (this problem is common with the OCI marketplace images).

In EDQ 12.2.2.1.4.4 and later versions, encryption and decryption is handled by a pluggable security module. The module is configured by the new properties file `localhome/security/module.properties`. This must contain a type property defining the module type. The following types are supported:

- legacy - The current mechanism used by default if `module.properties` does not exist.
- default - A refactoring of the legacy module with improved security.
- ocivault - Uses a key stored in an OCI vault. Encryption and decryption is performed by calls to the vault cryptographic endpoint. The key never leaves the vault so HSM or software protection modes can be used. Keys can be rotated for additional security - decrypt operations using the key version when the data was encrypted. It can be configured to act as a credentials store using vault secrets.
- script - A JavaScript script can be configured to perform the encryption and decryption operations. This can be used to support additional key frameworks such as AWS Key Management Service (KMS) or GCP Cloud Key Management Service. The module can also be configured to act a as a credentials store using external services such as AWS and GCP Secrets Managers.
- custom - Custom Java code. For more information, see Creating Custom Java Encryption and Credentials Store Modules.

The default module is used by new installations. The installations that are migrated from earlier versions continue to use the legacy module.

Default Security Module

The new default security module is similar to the legacy implementation but with these enhancements:

- AES is used in GCM mode with a random initialization vector so encrypting the same data returns different results each time.
- A 256-bit key is used if possible.
- In Tomcat, the key is stored in the file `localhome/masterkey` which is set to mode 600 (-rw-------).

Ocivault Security Module

The ocivault security module type uses a key stored in an OCI vault. Authentication for calls to the OCI APIs uses either platform credentials or credentials configured with external properties. Stored credentials cannot be used here because the module can be used to decrypt passwords for the EDQ database schemas at startup time.

The following additional properties are used with the type in `module.properties`:

Table 11-1 Ocivault Security Module Properties

Property	Description
vault	Vault name or OCID (required).
key	Key name or OCID (required).
region	OCI region name. If omitted, the region of the EDQ instance is used.
compartmentid	OCID of vault compartment. Used if vault or key are not specified by OCID. If omitted the compartment of the EDQ instance is used.
secrets.vault	Vault name or OCID for secrets queries. If omitted the key vault is used.

Table 11-1 (Cont.) Ocivault Security Module Properties

Property	Description
secrets.encrypt	Set to true if secret values are encrypted using the vault key. Secret values can be seen in the OCI console so they must be encrypted to hide them from the console.
secrets	Set to true to support credentials retrieval using vault secrets.
auth.X	Additional properties passed to the authenticator for OCI API calls. For example, set `auth.profile` to name a profile from *~/.oci/config*.

The primary use case for the OCI module is with compute instances which can use platform credentials for authentication for OCI API calls. If the OCI dynamic groups imply the required permissions, no additional configuration is required.

Secrets Storage

If the module is enabled for secrets use, credentials for external integrations can be stored in a vault rather than in the EDQ configuration files. The examples include JMS broker connections, SMTP authentication, and LDAP credentials.

To specify credentials using a secret, replace the username and password properties with the following:

```
cred.secret = secretname
```

The secret vault in the value should be a JSON string containing **username** and **password** attributes, as shown below:

```
{ "username": "user1", "password": "mysecret2" }
```

To override the default secret encryption setting for the module, use the following:

```
cred.secret.encrypt = true | false
```

To specify the username in plain and retrieve the password from the vault, use the following:

```
cred.secret.username = username
```

The secret value in the vault is then just the password.

If no username is required, add the following:

```
cred.secret.passwordonly = true
```

The secret value in the vault is then just the password.

Sample OCI Vault module.properties Configurations

Using platform credentials with local compartment and region:

```
type            = ocivault
vault           = edqvault
key             = key1
```

```
secrets         = true
secrets.encrypt = true
```

Using a remote vault with credentials from *~/.oci/config*:

```
type            = ocivault
auth.profile    = default
region          = us-phoenix-1
compartmentid   = compartmentid
vault           = edqvault
key             = key1
secrets         = true
```

Script Security Module

The module definition for the script security module must include a script property which is the name of the script file. If the file is not absolute it is found relative to the EDQ local configuration directory. The script defines functions as described in the following table:

Table 11-2 Script Security Module Functions

Function	Required	Description
init(props)	No	Initialize the module. The argument is an object containing properties from `module.properties`.
encrypt(plain)	Yes	Encrypt some plain text. The argument and result are ArrayBuffers.
decrypt(plain)	Yes	Decrypt some ciphertext. The argument and result are ArrayBuffers.
getCredentials(props)	No	Retrieve credentials from a secrets store. The properties argument contains the cred.* settings from the EDQ configuration.

The properties passed to the **init** function include a **type** value set to "encryption". This allows the same script to be used as the security module and a credentials store.

base64 stuff

External encryption APIs require base64 encoding for plain and cipher text. EDQ scripts can use these base64 support functions:

```
var encoder = BASE64.getEncoder()
var enc     = encoder.encode(bytes)                 // Encode an ArrayBuffer
to base64 string
var enc2    = encoder.encodeFromString(str)         // Encode a string to
base 64 string
```

```
var urlenc  = BASE64.getUrlEncoder()                // Encoder using URL-safe
alphabet
var mimeenc = BASE64.getMimeEncoder()               // Encoder with MIME line
splitting
vat nopad   = BASE64.getEncoder().withoutPadding()  // Encoder which does not
use = padding

var decoder = BASE64.getDecoder()
var dec     = decoder.decode(string)                // Decode a base64 string
to ArrayBuffer
var dec2    = encoder.decodeToString(str)           // Decode a base 64
string to a string

var urldec  = BASE64.getUrlEncoder()                // Decoder using URL-safe
alphabet
var mimdec  = BASE64.getMimeEncoder()               // Decoder with MIME line
splitting
```

Sample Scripts

Using the AWS Key and Secrets Management Frameworks

module.properties

```
type         = script
keyid        = alias/rde1
region       = eu-west-1
script       = awsenc.js
secrets      = true
auth.profile = myprofile
```

awsenc.js

```
// Script security module sample for AWS KMS and secrets manager
// ============================================================
//
// Required properties:
//
//      region          AWS region
//      keyid           KMS keyid or alias/name
//
// Optional:
//
//      secrets         Set to true for secrets support
//
// Configure proxy server with https.proxyHost and https.proxyPort properties

addLibrary("http")
addLibrary("logging")

var client
var secclient
var encoder = BASE64.getEncoder()
```

```
var decoder = BASE64.getDecoder()
var kmsurl
var keyid
var secrets   = false
var secretenc = false

function init(props) {

  // Authentication properties

  var aprops = Object.keys(props).filter(k =>
k.startsWith("auth.")).reduce((o, k) => (o[k.substring(5)] = props[k], o), {})

  // Client for KMS

  client = HTTP.builder().withAWSAuthentication(aprops).build();

  // Properties

  var region   = props.region

  keyid = props.keyid

  if (!region || !keyid) {
    throw "awsenc: region or key not specified"
  }

  kmsurl  = "https://kms." + region + ".amazonaws.com"

  // Secrets handling

  secrets = props.secrets == "true"

  if (secrets) {

    // URL and client for secret queries

    securl    = "https://secretsmanager." + region + ".amazonaws.com"
    secclient = HTTP.builder().withAWSAuthentication(aprops).build();
  }
}

function encrypt(plain) {
  var req = client.requestbuilder().header("X-Amz-Target",
"TrentService.Encrypt").build();
  var res = req.post(kmsurl, JSON.stringify({KeyId: keyid, Plaintext:
encoder.encode(plain)}), "application/x-amz-json-1.1")

  return decoder.decode(JSON.parse(res.data).CiphertextBlob)
}

function decrypt(ctext) {
  var req = client.requestbuilder().header("X-Amz-Target",
"TrentService.Decrypt").build();
  var res  = req.post(kmsurl, JSON.stringify({keyId: keyid, CiphertextBlob:
encoder.encode(ctext)}), "application/x-amz-json-1.1")
```

```
  return decoder.decode(JSON.parse(res.data).Plaintext)
}

function getCredentials(props) {
  if (secrets) {
    var s = props.secret

    if (s) {
      var req = secclient.requestbuilder().failonerror(false).header("X-Amz-
Target", "secretsmanager.GetSecretValue").build()
      var res = req.post(securl, JSON.stringify({SecretId: s}),
"application/x-amz-json-1.1")

      if (res.code != 200) {
        logger.log(Level.WARNING, "AWS secrets manager call failed with code
{0}", res.code)
      } else {
        var obj = JSON.parse(res.data)
        var str = obj.SecretString || decoder.decodeToString(obj.SecretBinary)

        if (props["secret.passwordonly"] == "true") {
          return {username: "", password: str}
        } else {
          var val = JSON.parse(str)
          return {username: val.username, password: val.password}
        }
      }
    }
  }

  return null
}
```

Using GCP Key and Secrets Frameworks

module.properties

```
type       = script
script     = gcpenc.js
project    = green-wombat-4252311
keyfile    = /opt/edq/gcp-encrypt.json
location   = europe-west2
keyring    = rde
key        = rde1
secrets    = true
seckeyfile = /opt/edq/gcp-secrets.json
```

gcpenc.js

```
// Script security module sample for GCP KMS and secrets manager
// ============================================================
//
```

```
// Required properties:
//
//      project         GCP project name
//      location        Key location (such as europe-west2)
//      keyring         Keyring name
//      key             Key name
//      keyfile         Service account key file location
//
// Optional:
//
//      version         Key version name
//
// Configure proxy server with https.proxyHost and https.proxyPort properties

addLibrary("http")
addLibrary("logging")

var client
var encoder = BASE64.getEncoder()
var decoder = BASE64.getDecoder()
var decurl
var encurl
var secrets   = false

function init(props) {

  // Need:
  //
  // project
  // location
  // keyring
  // key
  // keyfile
  //
  // Optional:
  //
  // version

  if (!props.project || !props.location || !props.keyring || !props.key || !
props.keyfile) {
     throw "gcpenc: missing properties"
  }

  var project = props.project
  var location = props.location

  var base = "https://cloudkms.googleapis.com/v1"
        + "/projects/"   + props.project
        + "/locations/"  + props.location
        + "/keyRings/"   + props.keyring
        + "/cryptoKeys/" + props.key

  decurl = base + ":decrypt"
  encurl = base

  if (props.version) {
```

```
    encurl += "/cryptoKeyVersions/" + props.version
  }

  encurl += ":encrypt"

  // Client for KMS

  client = HTTP.builder().withAuthentication("GCP", {keyfile: props.keyfile,
claim_scope: "https://www.googleapis.com/auth/cloud-platform"}).build();

  secrets = props.secrets == "true"

  if (secrets) {

    // URL and client for secret queries

    securl = "https://secretmanager.googleapis.com/v1/projects/" +
props.project + "/secrets/"
  }
}

function encrypt(plain) {
  var req = client.requestbuilder().build();
  var res = req.post(encurl, JSON.stringify({"plaintext":
encoder.encode(plain)}))

  return decoder.decode(JSON.parse(res.data).ciphertext)
}

function decrypt(ctext) {
  var req = client.requestbuilder().build();
  var res  = req.post(decurl, JSON.stringify({ciphertext:
encoder.encode(ctext)}))

  return decoder.decode(JSON.parse(res.data).plaintext)
}

function getCredentials(props) {
  if (secrets) {
    var s = props.secret

    if (s) {
      var url = securl + s + "/versions/latest:access"
      var req = client.requestbuilder().failonerror(false).build()
      var res = req.get(url)

      if (res.code != 200) {
        logger.log(Level.WARNING, "GCP secrets manager call failed with code
{0}", res.code)
      } else {
        var obj = JSON.parse(res.data)
        var str = decoder.decodeToString(obj.payload.data)

        if (props["secret.passwordonly"] == "true") {
          return {username: "", password: str}
        } else {
```

```
            var val = JSON.parse(str)
            return {username: val.username, password: val.password}
          }
        }
      }
    }

    return null
}
```

Encryption Migration

If the security module is replaced or reconfigured in an existing system, then encrypted data is no longer usable. If the system is a new install and encryption is not used in the `director.properties`, then this is not an issue. If there is an existing encrypted data, the new security module can be defined using a migration REST API.

There are two new system administration REST endpoints.

POST to /edq/admin/security/migrateencryption

The payload is a JSON object with the following structure:

```
{ "type" : "moduletype",
  "properties": {
     ... other settings for the module ...
  }
}
```

The migration process includes the following steps:

1. A new security is created and initialized using the definition in the payload.
2. Existing encrypted data is decrypted using the *current* module and encrypted using the *new* module.
3. If the previous steps succeed, the new module replaces the current module and *module.properties* is written with the new definition.

The result of the migration call is a report summarizing the items which were updated. Any items which could not be decrypted are listed.

If the URL contains the query parameter *?dryrun=true* then the new module is created and decryption of existing data tested, but no updates are made.

Example Payload for Migration to a Remote OCI Vault

```
{ "type" : "ocivault",
  "properties": {
    "auth.profile"    : "default",
    "vault"           : "rde",
    "key"             : "rtest",
    "compartmentid"   : "compartmentid",
    "region"          : "us-phoenix-1",
    "secrets"         : true,
    "secrets.encrypt" : true
```

```
    }
}
```

POST to /etc/admin/security/rotateencryption

The payload is an empty JSON object. Existing encrypted data is decrypted using the current security module and then encrypted using the same module. This call can be used with a vault key after a new version is created so that the old version can be deleted.

The result of the call is a summary for the migrate call.

If the URL contains the query parameter `?dryrun=true`, then decryption of existing data is tested but no updates are made.

Using Existing Schemas for New OCI Instance Without Losing Encrypted Data

If you create a new OCI instance but select the **Use Existing Schemas** option, the new instance starts with the legacy (or default) security module and creates a new secret key, rendering the existing encrypted data inaccessible. These are the steps to retain the data:

1. Before shutting down the old system, ensure it is using some external encryption mechanism (OCI vaults or AWS).
2. Create and setup the new system.
3. Run encryption migration to setup the external module on the new system. Decryption fails for existing data because the current module is different but after migration is complete. The encrypted data is usable again.

Configuring Pluggable Credentials Stores

The user name and password credentials used in EDQ configuration are normally defined in properties files.

Examples include:

- Database schema passwords in `director.properties`
- JMS broker authentication in "bucket" files
- SMTP authentication in `mail.properties`
- LDAP authentication in `login.properties`

When running an instance in WebLogic the credentials are stored in the FMW credentials store framework. The encryption module mechanism in EDQ 12.2.1.4.4 and later versions allows the security module to function as a credentials store but this does not support the definition of additional stores. For example, it is not possible to use a local encryption mechanism and retrieve credentials from external stores such as OCI Vaults or AWS Secrets Manager.

Configuring Additional Credential Stores

EDQ 12.2.1.4.4 and later versions extend the encryption module to allow additional credentials stores to be defined. The stores are configured by adding properties files to the `localhome/security/credstores` directory. Each properties file must contain a **type** property defining the store type along with additional settings which are specific to the store type. The following types are supported:

- ocivault - The credentials are stored as secrets in an OCI vault and can be encrypted using a value key.

- script - A JavaScript script can be configured to call out to external services such as AWS and GCP Secrets Managers.
- custom - A custom Java class.

Credentials stored are queried in the alphabetical order of the associated file names.

Ocivault Credentials Store

The ocivault security module type queries secrets stored in an OCI vault. Authentication for calls to the OCI APIs uses either platform credentials or credentials configured with external properties. Stored credentials cannot be used here because the module can be used to query passwords for the EDQ database schemas at startup time.

The following additional properties are used with the type.

Table 11-3 Ocivault Credentials Store

Property	Description
vault	Vault name or OCID (required).
secrets.encrypt	Set to true if secret values are encrypted using a key from the vault. The secret values can be seen in the OCI console so they must be encrypted to hide them from the console.
key	Key name or OCID (required if *secrets.encrypt* is true).
region	OCI region name. If omitted the region of the EDQ instance is used.
compartmentid	OCID of vault compartment. It is used if vault or key are not specified by OCID. If omitted the compartment of the EDQ instance is used.
auth.X	Additional properties passed to the authenticator for OCI API calls. For example, set `auth.profile` to name a profile from *~/.oci/config*.

The primary use case for the OCI store is with compute instances which can use platform credentials for authentication for OCI API calls. As long as the OCI dynamic groups imply the require permissions no additional configuration is required.

Using Secrets

To specify credentials using a vault secret, replace the username and password properties with the following:

```
cred.secret = secretname
```

The secret vault in the value should be a JSON string containing **username** and **password**attributes as shown below:

```
{ "username": "user1", "password": "mysecret2" }
```

To override the default secret encryption setting for the module, use the following:

```
cred.secret.encrypt = true | false
```

To specify the username in plain and retrieve the password from the vault, use the following:

```
cred.secret.username = username
```

The secret value in the vault is then just the password.

If no username is required, add the following:

```
cred.secret.passwordonly = true
```

The secret value in the vault is then just the password.

Example

oci.properties

```
type            = ocivault
secrets.encrypt = true
vault           = vault1
key             = rtest
compartmentid   = ocid1.compartment.oc1..aaaaaaaaeq2s...
region          = us-phoenix-1
```

Script Credentials Store

The properties for a script credentials store must include a **script** property which is the name of the script file. If the file is not absolute, it is found relative to the EDQ local configuration directory. The script defines the following functions:

Table 11-4 Script Credentials Store

Function	Required	Description
init(props)	No	Initialize the module. The argument is an object containing properties from the properties file.
getCredentials(props)	Yes	Retrieve credentials from a secrets store. The properties argument contains the cred.* settings from the EDQ configuration.

The properties passed to the **init** function include a **type** value set to "credentials". This allows the same script to be used as the security module and a credentials store.

Example Using the AWS Secrets Management Framework

For more information about base64 support functions, see EDQ Enhanced Encryption Overview.

aws.properties

```
type         = script
region       = eu-west-1
script       = awscreds.js
auth.profile = myprofile
```

awscreds.js

```
// Script credentials module sample for AWS secrets manager
// =========================================================
```

```
//
// Required properties:
//
//      region          AWS region
//
// Configure proxy server with https.proxyHost and https.proxyPort properties

addLibrary("http")
addLibrary("logging")

var secclient
var encoder = BASE64.getEncoder()
var decoder = BASE64.getDecoder()

function init(props) {

  // Authentication properties

  var aprops = Object.keys(props).filter(k =>
k.startsWith("auth.")).reduce((o, k) => (o[k.substring(5)] = props[k], o), {})

  // Properties

  var region   = props.region

  if (!region) {
    throw "awscreds: region not specified"
  }

  // URL and client for secret queries

  securl    = "https://secretsmanager." + region + ".amazonaws.com"
  secclient = HTTP.builder().withAWSAuthentication(aprops).build();
}

function getCredentials(props) {
  var s = props.secret

  if (s) {
    var req = secclient.requestbuilder().failonerror(false).header("X-Amz-
Target", "secretsmanager.GetSecretValue").build()
    var res = req.post(securl, JSON.stringify({SecretId: s}), "application/x-
amz-json-1.1")

    // Error 400 can mean secret not found, so don't report it

    if (res.code != 200) {
      if (res.code != 400) {
        logger.log(Level.WARNING, "AWS secrets manager call failed with code
{0}", res.code)
      }
    } else {
      var obj = JSON.parse(res.data)
      var str = obj.SecretString || decoder.decodeToString(obj.SecretBinary)

      if (props["secret.passwordonly"] == "true" || props["secret.username"])
```

```
{
        return {username: props["secret.username"] || "", password: str}
      } else {
        var val = JSON.parse(str)
        return {username: val.username, password: val.password}
      }
    }
  }

  return null
}
```

Creating Custom Java Encryption and Credentials Store Modules

The 12.2.1.4.4 and 14.1.2.0.0 versions of EDQ supports pluggable modules for encryption and credentials lookup. The following section describes how to implement modules using custom Java classes which is necessary if it is not possible to write a script to interact with some security store because native code is required.

Custom modules are defined as shown below:

```
type  = custom
class = java class name
```

Perform the following steps to create a custom module:

1. Create a Java class which implements the required interface.
2. Compile the class using `installdir/buildjars/customsecuritymodules.jar` in the classpath.
3. Package the classes into a jar file in `localhome/security/jars`.

Brief javadocs for the interfaces that are included in the `docs` subfolder in `installdir/buildjars/customsecuritymodules.jar`.

Custom Encryption Module

A custom encryption module must implement the interface **oracle.edq.security.module.custom.interfaces.CustomEncryptionModule** as shown below:

CustomEncryptionModule

```
/*
 * Copyright (C) 2023, Oracle and/or its affiliates. All rights reserved.
 */
package oracle.edq.security.module.custom.interfaces;

import java.nio.file.Path;
import java.util.Properties;

/**
 * Interface implemented by custom encryption modules.
 */
```

```
public interface CustomEncryptionModule {

  /**
   * Initialize the encryption module.
   *
   * @param localconfig The local configuration area.  This can be used to
locate or store extra files
   * required by the module.
   * @param props The module properties
   * @param persist If <code>true</code> the module data can be persisted for
future use.
   * This flag will be <code>false</code> if the module is being initalized
for a "dryrun" encryption migration.
   * In this case no permanent changes should be made in the file system.
   *
   * @throws Exception If initialization failed.
   */

  void initEncryption(Path localconfig, Properties props, boolean persist)
throws Exception;

  /**
   * Encrypt some data.
   *
   * @param in The plain text
   *
   * @return The cipher text
   *
   * @throws Exception If encryption failed
   */

  byte [] encrypt(byte [] in) throws Exception;

  /**
   * Decrypt some data.
   *
   * @param in The cipher text
   *
   * @return The plain text
   *
   * @throws Exception If decryption failed
   */

  byte [] decrypt(byte [] in) throws Exception;
}
```

If the encryption module implements **oracle.edq.security.module.custom.interfaces.CustomCredentialsModule** also then it acts as a credentials store. In this case the **initCredentials** method is not called.

Custom Credentials Store

A custom encryption module must implement the interface **oracle.edq.security.module.custom.interfaces.CustomCredentialsModule** as shown below:

CustomCredentialsModule

```
/*
 * Copyright (C) 2023, Oracle and/or its affiliates. All rights reserved.
 */
package oracle.edq.security.module.custom.interfaces;

import java.nio.file.Path;
import java.util.Properties;

/**
 * Interface implemented by custom credentials modules.
 */

public interface CustomCredentialsModule {

  /**
   * Get the prefix used to extract properties recognized by the module.  The
default value is "cred".
   *
   * @return The prefix
   */

  default String credsPrefix() {
    return "cred";
  }

  /**
   * Initialize the credentials provider for credentials-only use.
   *
   * @param localconfig The local configuration area
   * @param props The module properties
   *
   * @throws Exception If initialization failed.
   */

  void initCredentials(Path localconfig, Properties props) throws Exception;

  /**
   * Attempt to get credentials.
   *
   * @param props The filtered property set.
   *
   * @return The credentials, or <code>null</code> if not found
   *
   * @throws Exception If an error ocurred
   */

  Credentials getCredentials(Properties props) throws Exception;

  /**
   * Credentials result class.
   */

  final class Credentials {
```

```
    private String  username;
    private String  password;

    /**
     * Constructor.
     *
     * @param username The user name
     * @param password The password
     */

    public Credentials(String username, String password) {
      this.username = username;
      this.password = password;
    }

    /**
     * Get the user name.
     *
     * @return The user name
     */

    public String getUsername() {
      return username;
    }

    /**
     * Get the password.
     *
     * @return The password
     */

    public String getPassword() {
      return password;
    }
  }
}
```

A

Configuring EDQ to support Windows Integrated Authentication (Kerberos)

The appendix contains information on configuring EDQ to work with Kerberos and Active Directory.

An integration of EDQ with Active Directory using *login.properties* (see Configuring External User Management (LDAP) directly with EDQ) can be configured to support single sign on - SSO - in which the user logs into Windows and then does not need to log again into EDQ. This is supported on both WebLogic and Tomcat.

To enable SSO, the EDQ server must be set up to enable Kerberos authentication from the client PC. This authentication is achieved using the standard GSSAPI token exchange mechanism (RFC 4121). The client contacts the domain controller (DC) to request access to a service provided by the server application. The response from the DC is encoded into a token sent to the server by the client. The server validates this token and generates another token to send to the client. The token exchange can continue until client and server have established a secure context. In practice this exchange never requires more than one token in either direction.

At start up time the server application sets up 'accept' credentials which it uses to initialize its half of the security context.

- EDQ running as Windows service using local system account
  If the EDQ application server is running on a Windows server in the domain, using the local system account, then the configuration is very simple. EDQ will use the system account for the accept credentials and also to contact AD for user lookups.
- EDQ running on Unix
  If the server is running on Unix, it must use an account in AD to set up the accept credentials. It validates the request using the encrypted account password read from a Kerberos key table (keytab). Setting up a valid key table is an essential step in configuring SSO on Unix.
- Kerberos Shared Libraries
  The shared libraries (*wingss.dll* and *libunuxgss.so*) required for Kerberos integration are shipped inside the `edq.war` file. For most installations this is sufficient since EDQ can determine the location of the shared libraries and load the right version automatically.

EDQ running as Windows service using local system account

If the EDQ application server is running on a Windows server in the domain, using the local system account, then the configuration is very simple. EDQ will use the system account for the accept credentials and also to contact AD for user lookups.

The EXAMPLE.COM *login.properties* for this configuration would be:

```
# EXAMPLE.COM LDAP integration
# ----------------------------

realms                          = internal, ad
ldap.prof.useprimarygroup       = false
```

```
clientcreds                    = true

ad.realm                       = EXAMPLE.COM
ad.auth                        = ldap
ad.auth.bindmethod             = simple
ad.auth.binddn                 = search: dn

ad.ldap.profile                = adsldap
ad.ldap.prof.defaultusergroup  = edqusers
ad.ldap.prof.groupsearchfilter = (cn=edq*)
```

The 'clientcreds' setting indicates that Kerberos credentials should be obtained from the current user's cache (in this case the local system account). These credentials are used to connect to Active Directory and to set up the 'accept' GSSAPI context.

A server which is a member of the Active Directory domain will generally use the domain controller for DNS lookups. If this is the case, EDQ can determine the LDAP server addresses automatically. If you wish to fix the address, perhaps because some of the domain controllers are at a remote location, use the ldap.server property:

```
ad.ldap.server                 = dc1.example.com
```

EDQ running on Unix

If the server is running on Unix, it must use an account in AD to set up the accept credentials. It validates the request using the encrypted account password read from a Kerberos key table (keytab). Setting up a valid key table is an essential step in configuring SSO on Unix.

What is in the keytab?

A Kerberos key table contains encrypted passwords for one or more Kerberos principals. The DC normally supports a number of different encryption algorithms (DES3, AES, RC4 etc) and the entry for a principal will include keys for each of these algorithms. The client will pick the best algorithm available for communication with the DC.

The service requested using GSSAPI is identified by a service principal name (SPN).Normally this will be a reference to a particular service type at a machine hostname. Examples of service types are **HOST** (for general access such as ssh), **HTTP** (for SSO from browsers) and **LDAP** (for LDAP servers such as AD domain controllers). An SPN is usually displayed as:

```
service/hostname
```

For example:

```
HOST/testserver.example.com
```

Each entry in a keytab also includes a 'key version number' (KVNO). This is a version number which is incremented whenever the password for the principal is changed in the DC. The keytab must contain the correct KVNO for authentication to succeed.

On most Unix systems, the default location of the system Kerberos keytab is:

```
/etc/krb5.keytab
```

The Java Kerberos implementation does not have a default keytab location and this must be set in *loign.properties*:

```
keytab = path to keytab
```

If the path is not absolute, it is relative to the security folder containing *login.properties*.

The *klist* command can be used to list the contents of a keytab:

```
klist -k [file]
klist -ke [file]
klist -keK [file]
```

A file name can be provided if the keytab is not in the default location. The first form just lists the principals; the second also includes the encryption algorithms and the third also includes the key values in hexadecimal.

Here's some example output from *klist -ke*:

```
Keytab name: FILE:/etc/krb5.keytab
KVNO Principal
---- --------------------------------------------------------------------------
   5 ALTAIR$@EXAMPLE.COM (des-cbc-crc)
   5 ALTAIR$@EXAMPLE.COM (des-cbc-md5)
   5 ALTAIR$@EXAMPLE.COM (des3-cbc-sha1)
   5 ALTAIR$@EXAMPLE.COM (aes128-cts-hmac-sha1-96)
   5 ALTAIR$@EXAMPLE.COM (aes256-cts-hmac-sha1-96)
   5 ALTAIR$@EXAMPLE.COM (arcfour-hmac)
   5 HOST/altair@EXAMPLE.COM (des-cbc-crc)
   5 HOST/altair@EXAMPLE.COM (des-cbc-md5)
   5 HOST/altair@EXAMPLE.COM (des3-cbc-sha1)
   5 HOST/altair@EXAMPLE.COM (aes128-cts-hmac-sha1-96)
   5 HOST/altair@EXAMPLE.COM (aes256-cts-hmac-sha1-96)
   5 HOST/altair@EXAMPLE.COM (arcfour-hmac)
   5 HOST/altair.EXAMPLE.COM@EXAMPLE.COM (des-cbc-crc)
   5 HOST/altair.EXAMPLE.COM@EXAMPLE.COM (des-cbc-md5)
   5 HOST/altair.EXAMPLE.COM@EXAMPLE.COM (des3-cbc-sha1)
   5 HOST/altair.EXAMPLE.COM@EXAMPLE.COM (aes128-cts-hmac-sha1-96)
   5 HOST/altair.EXAMPLE.COM@EXAMPLE.COM (aes256-cts-hmac-sha1-96)
   5 HOST/altair.EXAMPLE.COM@EXAMPLE.COM (arcfour-hmac)
```

In a normal Kerberos system using a standard Kerberos Domain Controller (KDC) each SPN is a separate principal with a different password. In Active Directory, SPNs are essentially 'aliases' of a single account, stored as values of the AD **servicePrincipalName** LDAP attribute. When a computer account is created in AD, SPNs for the **HOST** service are created automatically. If additional services such as IIS or SQLserver are installed on the server, additional SPNs will be added to the account.

The Windows setspn command can be run on an AD server to manage the SPNs for an account. For example:

```
setspn -A HTTP/altair.example.com altair$
```

The first command adds an HTTP SPN to the machine account for altair. (The Windows account name for a computer always ends with $).

Creating keytabs using existing tools

In a normal Kerberos system, keytab entries are created using the ktadd subcommand of the Kerberos administration tool *kadmin*. AD does not provide a Kerberos administration server so other approaches are required.

The keytab contains the encrypted password for the account so for each method either the password for the account must be known in advance, or it must be run with privileges to change the account password.

The method to use depends on the system configuration. Some existing options are:

- **Samba**: If the system has been registered with AD using the Samba suite, the net ads keytab command can be used to create and update the keytab. This works because Samba has set the password for the account and stored it in a secret location.
- **ktpass**: The Windows ktpass command can run by an AD administrator to generate keytab entries. Unless there is no other alternative, do not use this command. It is complex and very difficult to use reliably. It will update the password of the account, thus rendering any previous keytab useless.
- **msktutil**: This is an open source application for Unix which can be used to manage keytabs.

Creating keytabs using winktab

winktab is a Java application which can be used to create a keytab for an account in AD. It contacts AD to determine the KVNO for the account and to determine the SPNs associated with the account. If run with an administrator account, it can reset the account password to a random value; alternatively the account password can be supplied to the command.

winktab can be run on any system which can connect to the AD server.

> **Note:**
>
> You must type all commands manually as copying and pasting the command can result in issues with the type of dash used in the command.

To create a keytab for a machine account and reset the password to a random value:

```
java -jar dbstuff.jar winktab -o ktfile -domain DOMAIN -server adserver
                        -user aduser -pw adpw -tls machinename
To create a keytab for a machine account using a known password:
java -jar dbstuff.jar winktab -o ktfile -accpw accpw -domain DOMAIN -server adserver
                        -user aduser -pw adpw [-tls] machinename
```

The parameters are:

- ktfile: the keytab is written to this file
- accpw: the password for the machine or user account; if a single dash (-) is used the command will prompt for the password without echo
- DOMAIN: the AD domain name
- adserver: the host name or IP address of an AD server for the domain

- aduser, adpw: the user name and password of an AD account which is used to connect to the server for queries. The user name must be:

  user@DOMAIN

  or SHORTDOMAIN\user

  If a single dash (-) is given as the password the command will prompt for it without echo.

  If the account password is set to a fixed (**-setpw** used) or random (**-setpw** and **-accpw** omitted), the user must have administrator privileges.

- **machinename**: the name of a computer account. The internal AD account name will be **machinename**$. The internal AD account name is stored in the **sAMAccountName** LDAP attribute.

  Use the **-tls** switch if the AD server requires authenticated connections. Windows requires an encrypted connection if an account password is being reset so **-tls** must be used if the password is being reset to a random or known value.

  Example for the domain EXAMPLE.COM, split over several lines for clarity:

  ```
  java -cp dbstuff.jar rde.security.tools.win.winktab -o altair.keytab -setpw altair \
                              -domain EXAMPLE.COM -server dc1.example.com \
                              -user "example\Admin" -tls altair
  ```

  Create a keytab for a machine account, setting the password to a known value and prompting for the administrator password.

Check the Unix Kerberos configuration

The Kerberos configuration used by commands such as *kinit* and the Java runtime is read from a global configuration file, normally stored at */etc/krb5.conf*. This contains references to the domain controllers and mappings between DNS and Kerberos domains. Here's an example for the domain EXAMPLE.COM:

```
[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log

[libdefaults]
 default_realm = EXAMPLE.COM
 dns_lookup_realm = false
 dns_lookup_kdc = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = yes

[realms]
 EXAMPLE.COM = {
  kdc = dc1.example.com:88
}

 [domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

The [*realms*] section lists the KDCs by host or IP for each domain; the [*domain_realm*] section maps DNS host names to Kerberos domains.

krb5.conf must be checked and adjusted for the configuration of the target domain. If it is not possible to update a file in /etc, store the file elsewhere and inform the Java runtime of the

location via a system property. To do this, edit or create the file jvm.properties in the EDQ local configuration directory and add the line:

```
java.security.krb5.conf  = absolute path to modified krb5.conf
```

Java Encryption

Java runtime environments do not ship with support for high-strength ciphers by default. For example AES with 256-bit keys is not supported. Active Directory and Kerberos support a full set of ciphers and it may be that your key table contains entries not supported by Java. In this case you should install the JCE Unlimited Strength Jurisdiction Policy Files, if permitted in your location.

Changes to login.properties

In this example, the server running EDQ is assumed to be **altair.example.com**.

- **Global Settings**:

  ```
  realms                    = internal, ad
  spn                       = HOST/altair.example.com@EXAMPLE.COM
  keytab                    = /etc/krb5.keytab
  ldap.prof.useprimarygroup = false
  ```

  The 'spn' property sets the Service Principal to use for the GSSAPI accept context. The value shown here is the default and can be omitted if the Unix server's DNS domain the same as the AD domain. Sometimes Unix servers are in a separate DNS domain and the 'spn' property is then required.

  The 'keytab' property sets the location of the Kerberos key table. Java does not default this to the standard location and this property is always required.

- **LDAP server settings**:

  ```
  ad.ldap.server = dc1.example.com
  ad.ldap.spn    = "ALTAIR$@EXAMPLE.COM"
  ```

  Set the server name and Kerberos principal used to connect to Active Directory. The principal should be the key table entry containing the actual machine account name. As before the server can be omitted if it can be determined from DNS.

- Putting it all together:

  This is the complete *login.properties* used for the Active Directory + Kerberos integration example:

  ```
  # EXAMPLE.COM LDAP integration with
  Kerberos




  realms                          = internal,
  ad

  spn                             = HOST/
  altair.example.com@EXAMPLE.COM

  ldap.prof.useprimarygroup       =
  false
  ```

```
keytab                          = /etc/
krb5.keytab



ad.realm                        =
EXAMPLE.COM


ad.label                        = Example, Inc. Active
directory

ad.ldap.server                  =
dc1.example.com


ad.ldap.spn                     =
"ALTAIR$@EXAMPLE.COM"


ad.ldap.security                =
tls


ad.auth                         =
ldap


ad.auth.bindmethod              =
simple


ad.auth.binddn                  = search:
dn



ad.ldap.profile                 =
adsldap


ad.ldap.prof.defaultusergroup   =
edqusers



ad.ldap.prof.groupsearchfilter = (cn=edq*)
```

With this *login.properties* in place, a Windows user in the EXAMPLE.COM domain should be able to login to an EDQ application without supplying additional credentials. The user must of course have the permission in EDQ to use the application.

Kerberos Shared Libraries

The shared libraries (*wingss.dll* and *libunuxgss.so*) required for Kerberos integration are shipped inside the `edq.war` file. For most installations this is sufficient since EDQ can determine the location of the shared libraries and load the right version automatically.

However, this automatic loading does not work with all Java Runtime Environments (JREs), and notably it does not work with the IBM JRE. For these installations the libraries need to be extracted from the provided `kerberos-gss.zip` file and copied to a known location on disk. The location must then be added to the following environment variables such that the JRE can find it:

- `LD_LIBRARY_PATH (Linux, Solaris)`
- `LIBPATH (AIX)`
- `PATH (Windows)`

Examine the `native/Kerberos-gss.zip` archived provided with the EDQ install and verify it contains the following files:

- `aix/ppc/libunixgss.a`
- `aix/ppc64/libunixgss.a`
- `linux/amd64/libunixgss.so`
- `linux/i386/libunixgss.so`
- `win32/amd64/wingss.dll`
- `win32/x86/wingss.dll`

Extract the relevant library for the OS the EDQ server runs on, and copy it to a location on a disk accessible by the user EDQ runs as. This location needs to be added to the environment variables mentioned above, so the JRE can find the library.

B

Configuring Single Sign On with Oracle Access Manager (OAM)

When EDQ is integrated with Oracle Access Manager, a user can login on a common access page and have automatic access to EDQ applications and the web console without additional Logins (assuming or course that the user has the required EDQ permissions). If there are multiple EDQ installations using the same OAM configuration, the login will work for each. For more information, see *Oracle Access Management*.

This section covers the configuration steps to integrate EDQ with OAM. It does not cover installation and basic configuration or OAM or installation of the Web Tier front end (OHS). This appendix contains the following sections:

- Prerequisites
  This section provides information about the prerequisites required for installing OAM.
- OAM configuration
  This section describes how to configure OAM.
- WebLogic plugin configuration
  This section describes how to configure WebLogic plugin.
- WebLogic Configuration
  This section describes how to configure WebLogic.

Prerequisites

This section provides information about the prerequisites required for installing OAM.

The following are the prerequisites for installing OAM:

- OAM must be configured with an Authentication Scheme using an identity store supported by WebLogic (typically LDAP - Active Directory or Oracle Internet Directory).
- WebLogic must be configured to authenticate EDQ using the same identity store. See Integrating External User Management (LDAP) using WebLogic and OPSS . This should be configured and tested with EDQ before proceeding with the OAM integration steps.
- A web server front end (OHS or Apache) must be installed and configured with Webgate software and the WebLogic plugin (mod_wl_ohs). These are bundled with OHS 14 releases.

OAM configuration

This section describes how to configure OAM.

To configure OAM, follow the steps below:

1. Create a Webgate in OAM using the authentication schema which refers to the identity store configured in WebLogic.

   Create these HTTP resources in the Webgate:

Table B-1 Creating HTTPS resources in the Webgate

RESOURCE	POLICY
/edq/ui/**	Protected Resource Policy
/edq/**	Public Resource Policy (or Excluded)

2. Copy the Webgate artefacts to your OHS installation and place in the *webgate/config* directory.

WebLogic plugin configuration

This section describes how to configure WebLogic plugin.

Ensure that the WebLogic plugin (mod_wl_ohs) is configured in the web server front end. Add this entry to the plugin configuration file (normally mod_wl_ohs.conf):

```
<Location /edq>
  SetHandler weblogic-handler
  WebLogicPort managed server port
  WebLogicHost hostname
</Location>
```

If you are using a WebLogic cluster, replace the host and port settings with a cluster definition:

```
WebLogicCluster host1:port1, host2:port2, ...
```

Ensure that the WebLogic Plug-In enabled option is set for the EDQ servers. This can be done at the domain, cluster, server template or server level. For the domain the option is present in the Configuration/Web Applications tab. For the other items the option is present in the Advanced area of the General Configuration tab.

WebLogic Configuration

This section describes how to configure WebLogic.

To configure WebLogic, follow the steps below:

1. Login to the WebLogic Remote Console.
2. In the **Edit Tree**, go to **Security** > **Realms** > **Authentication Providers**.
3. Click **New** to add a new provider.
4. In the **Name** field, enter a name. For example, OAM.
5. Click the **Type** drop-down and select **Oracle Access Manager Identity Asserter**.

   You must set the following values:

   - **Active Types** - `OAM_REMOTE_USER`
   - **SSO Header Name** - `OAM_REMOTE_USER`
6. Click the **Control Flag** and set the value of all other providers as **SUFFICIENT**.
7. Click the move up button to reorder the providers so that the **OAM Identity Asserter** is displayed as the first provider.
8. Click **shopping cart** on top right corner and select **commit changes**.