Oracle Commerce Guided Search

Security Guide

Version 11.1 • July 2014

ORACLE®

COMMERCE

Contents

Copyright and disclaimer

Preface

Oracle Commerce Guided Search is the most effective way for your customers to dynamically explore your storefront and find relevant and desired items quickly. An industry-leading faceted search and Guided Navigation solution, Guided Search enables businesses to influence customers in each step of their search experience. At the core of Guided Search is the MDEX Engine™, a hybrid search-analytical database specifically designed for high-performance exploration and discovery. The Oracle Commerce Content Acquisition System provides a set of extensible mechanisms to bring both structured data and unstructured content into the MDEX Engine from a variety of source systems. The Oracle Commerce Assembler dynamically assembles content from any resource and seamlessly combines it into results that can be rendered for display.

Oracle Commerce Experience Manager enables non-technical users to create, manage, and deliver targeted, relevant content to customers. With Experience Manager, you can combine unlimited variations of virtual product and customer data into personalized assortments of relevant products, promotions, and other content and display it to buyers in response to any search or facet refinement. Out-of-the-box templates and experience cartridges are provided for the most common use cases; technical teams can also use a software developer's kit to create custom cartridges.

About this guide

This guide describes the security features of Oracle Commerce Guided Search and explains how to use them to enhance the security of your Oracle Commerce Guide Search applications.

Who should use this guide

This guide is for anyone who is responsible for implementing security features in Oracle Commerce Guided Search applications.

Conventions used in this guide

This guide uses the following typographical conventions:

Code examples, inline references to code elements, file names, and user input are set in `monospace` font. In the case of long lines of code, or when inline monospace text occurs at the end of a line, the following symbol is used to show that the content continues on to the next line: ¬

When copying and pasting such examples, ensure that any occurrences of the symbol and the corresponding line break are deleted and any remaining space is closed up.

Contacting Oracle Support

Oracle Support provides registered users with answers to implementation questions, product and solution help, and important news and updates about Guided Search software.

You can contact Oracle Support through the My Oracle Support site at *https://support.oracle.com*.

Chapter 1

Introduction

Oracle Commerce Guide Search software is designed to provide a high level of security for your applications. This guide describes a number of recommended steps that you can take to enhance that security.

Enhancing the security of communication among Oracle Commerce Guided Search components

Oracle recommends that you enhance the security of your Oracle Commerce Guided Search application's communications by taking basic measures such as installing the application behind a firewall on a machine that hosts no other applications, and by making use of the security features that your network software provides.

In addition, Oracle recommends that you implement Secure Socket Layer (SSL) communication to guarantee the security of communication among your application's components. Following the procedures described in this guide, you can implement SSL communication by generating SSL certificates and credentials, storing the certificates and credentials securely, and enabling SSL communication to and from specific components of your application.

Chapter 2

Basic Security Measures

This section describes basic measures that you can follow to enhance the security of your Guided Search implementation.

Firewall Security

Your MDEX Engine and all other components of the Guided Search implementation – except for your web application – must be placed behind a secure firewall.

Ensure that ports are open to enable the web application to communicate through the firewall with the MDEX Engine and Workbench.

Installing Oracle Commerce Guided Search on its own machine

To protect your Guided Search implementation against viruses, malware, and other forms of malicious interference, Oracle recommends that you host your Guided Search software on a machine on which the only other software is the operating system. Running additional software on the same machine – for example, email applications – can compromise the security of your Guided Search implementation.

Using local network restrictions to protect application components

Be sure to protect your Guided Search components against unauthorized access by installing them in restricted parts of your local network.

In particular, be sure to place the following components in restricted parts of your network:

- Endeca Application Controller (EAC)
- Content Acquisition System (CAS)
- Log Server
- MDEX Engine
- Workbench

In addition, you can restrict access to Workbench by defining profiles of the users who will be authorized to access Workbench. See *Authenticating Workbench Users* on page 13.

Chapter 3

Authenticating Workbench Users

This section describes how to authenticate Workbench users against credentials stored on an LDAP server or an Active Directory server, or to create user profiles for authenticating and managing Workbench users.

Integrating Workbench with an LDAP Server for User Authentication

A Workbench administrator can configure Workbench to authenticate users against user profiles stored on an LDAP server. The roles and permissions associated with each user profile are stored in the Workbench database.

The user profiles in the LDAP directory must be created independently of Workbench, which cannot write anything to LDAP directories.

For information about how to integrate LDAP with Oracle Commerce Workbench, refer to the *Oracle Commerce Guided Search Administrator's Guide*.

Managing Workbench Users Through Profiles

You can also control access to Workbench by creating Workbench user profiles.

Workbench users, roles, and permissions can be defined by a Workbench administrator. Workbench users log in to an application in Workbench with basic user name and password authentication. Before a business user can log in to an application in Workbench, a Workbench administrator or a user with the settings role must create a profile for the user that includes the following:

- user name
- password
- roles and permissions
- user identity information such as first name, last name, and email address

For information about how to create a Workbench user profile, refer to the *Workbench Administrator's Guide*.

Securing the Built-in Workbench Admin Account

Be sure to create a strong password for the Workbench Admin account.

Use a generated password whenever possible. If you create a password yourself, follow these standard guidelines to make the password as strong as possible:

- Use passwords that are at least 12 characters long.
- Do not use passwords that contain repeated elements, dictionary words, sequences of letters or numbers, user names, names of relatives, friends, or pets, or biographical information.
- Include numbers, and symbols if possible.
- Use a mixture of upper-case and lower-case letters, if your can system distinguish them from each other.
- Do not use the same password for different sites or for different purposes.
- Do not use passwords that refer to any of your personal preferences or dislikes.

For more information, consult a standard reference on the topic of password strength.

Chapter 4

Creating SSL Certificates

The Java utility `generateSSLCertificates` creates certificates that support secure two-way SSL communication between Oracle Commerce components on different machines in a distributed environment. For example, the certificates can support secure communication between a web application deployed on one server and an MDEX Engine (Dgraph) deployed on a different server, or between a Workbench deployed on one server and an EAC Central Server.

Installation

The `generateSSLCertificates` utility is installed by default in the following directories as part of the Tools and Frameworks installation.

Component	**Default Directory**
generateSSLCertificates.bat(Windows)/ generateSSLCertificates.sh(Unix)	deployment_template\ssl_Certs_Utility\bin
Output (certificates)	`deployment_template\ssl_certs_utility\bin\ssl`
`README.txt`	`deployment_template\ssl_certs_utility\bin\ssl`

The `README.txt` file provides details about the installation and output of `generateSSLCertificates`.

For information about how to use other installation directories for `generateSSLCertificates`, refer to the *Tools and Frameworks Installation Guide*.

How generateSSLCertificates Creates Two-Way Host Certificates

To create two-way host certificates using generateSSLCertificates, follow these steps:

1. If there is a CA root certificate in the `deployment_template\bin\ssl` directory, the `generateSSLCer¬tificates` utility loads the filename of this certificate. It will use the CA root certificate to sign the SSL host certificates that it generates.
2. If there is no CA root certificate in `deployment_template\ssl_certs_utility\bin\ssl`, the `gen¬erateSSLCertificates` utility:
   a. Generates a self-signed root certificate.

   b. Places the root certificate in the directory `deployment_template\ssl_certs_utility\bin\ssl`.

3. The `generateSSLCertificates` utility generates SSL host certificates in PEM, PKCS12, and JKS formats. To do this, it:
   a. Prompts the user to specify the hostname and domain name of the server.
   b. Generates a host certificate.
   c. Adds the hostname and domain name to the host certificate.
   d. Places the host certificate in the directory `deployment_template\ssl_certs_utility\bin\ssl`.
   e. Prompts the user to specify passphrases for the keystore and trust store files.
   f. Prompts the user to specify a keyname to add passphrases to OCS with the mapName `oracleCom¬merceSSLPassPhrase`.

   Note: Each of the host certificates is digitally signed by the CA root certificate.

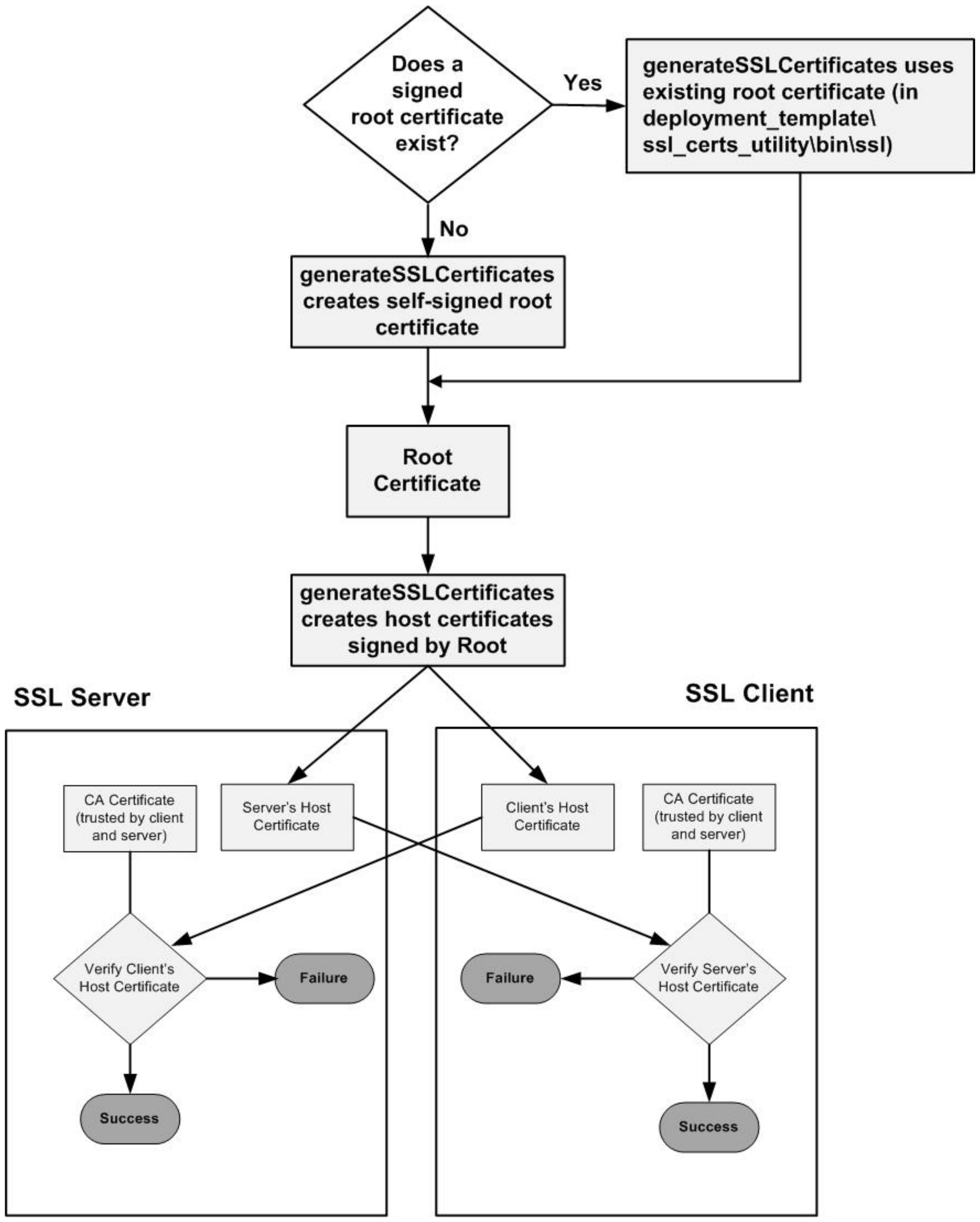
4. It prompts users to indicate whether they want to generate a certificate for another host.
   - If the user answers yes, it repeats steps 3a through 3f.
   - If the user answers no, the `generateSSLCertificates` utility exits.

The hosts in the distributed system use the certificates created by `generateSSLCertificates` as follows:

1. The server and each client load one of the SSL host certificates created by `generateSSLCertificates`.
2. When a client asks to verify the server, the server sends its SSL host certificate to the client.
3. The client verifies the SSL host certificate by comparing it to its own CA Certificate, which was signed by the Certificate Authority that signed the root certificate.
4. When a server asks to verify a client, the client sends its SSL client certificate to the server.
5. The server verifies the SSL client certificate by comparing it to its own CA Certificate, which was signed by the Certificate Authority that signed the root certificate.

Because the server and the clients have certificates signed by a Certificate Authority whom they all trust, they can recognize each other as trustworthy.

The following figure illustrates how `generateSSLCertificates` establishes SSL connections between Oracle Commerce components:


Does a signed root certificate exist?
Yes
generateSSLCertificates uses existing root certificate (in deployment_template\ssl_certs_utility\bin\ssl)
No
generateSSLCertificates creates self-signed root certificate
Root Certificate
generateSSLCertificates creates host certificates signed by Root
SSL Server
SSL Client
CA Certificate (trusted by client and server)
Server's Host Certificate
Client's Host Certificate
CA Certificate (trusted by client and server)
Verify Client's Host Certificate
Failure
Failure
Verify Server's Host Certificate
Success
Success

Storing generateSSLCertificates Credentials

The certificates and keys created by the `generateSSLCertificates` utility for each host must be stored in the following files on that host:

- The *truststore*, which stores certificates from the trusted Certificate authorities (CA).
- The *keystore*, which stores the private key and host certificate of the host.

For information about the file names of the truststore and keystore, refer to the README.txt file provided with the `generateSSLCertificates` utility.

Access to the truststores and keystores must be limited to the application meant to use the certificates. To limit access to the truststores and keystores, `generateSSLCertificates` prompts you to specify passphrases for the truststores and keystores. You can specify the same passphrase or different passphrases for a truststore and keystore.

For an illustration of how to specify the passphrases, see *Example: Using generateSSLCertificates* on page 20.

Storing Passphrases in the Oracle Credential Store

Passphrases themselves must be placed in the Oracle Credential Store, to prevent unauthorized access to them.

To do this, follow these steps:

1. On each host in the Oracle Commerce distributed application, use the Oracle Wallet Manager to create an Oracle Wallet to hold the passphrases that you specify for the truststores and keystores on that host. For information about how to do this, refer to the Oracle documentation.
2. On each host, `run manage_credentials.bat` (Windows) or `manage_credentials.sh` (UNIX) to upload the passphrases for that host to the Oracle Wallet.

Because there is only one truststore and one keystore on each host, `manage_credentials` must be run only once on each host.

Note: The Oracle Credential Store is shipped with no passphrases or other credentials in it.

The following sections describe the syntax of `manage-credentials` for adding passphrases to and deleting them from the Oracle Credential Store.

Adding Passphrases to the Oracle Credential Store

```
manage_credentials.bat[.sh] add [--config path_to_jps-config.xml] [--mapName
map_name] [--user user_id] [--key key_name] [--type (password|generic)]
```

where:

`add`	Required, to identify the operation to be performed.
`path_to_jps-config.xml`	The path to the Oracle Wallet configuration file `jps_config.xml`. The default path is `../../server/workspace/credential_store/jps-config.xml`
`map_name`	Omit to accept the required default (`endecaToolsAndFrameworks`).
`user_id`	The ID of the user who invokes the `manage_credentials` script. The default is "admin" . Note that `user_id` is used only when you specify `password` for `--type`.

key_name	Any arbitrary value by which the passphrase can be accessed. If you do not specify a `key_name` value with the --key option, you are prompted to enter one.
`password|gener¬ ic`	Specify `generic`.

Deleting Passphrases from the Oracle Credential Store

`manage_credentials.bat[.sh] delete [--config` *`<path to jps-config.xml>`* `] [--mapName` *`map_name`*`] [--key` *`key_name`*`]`

where:

`delete`	Required, to identify the operation to be performed.
path_to_jps-config.xml	The path to the Oracle Wallet configuration file `jps_config.xml`. The default path is `../../server/workspace/credential_store/jps-config.xml`
`map_name`	Omit to accept the required default (`endecaToolsAndFrameworks`).
key_name	Required to identify the passphrase to be deleted from the Oracle Credential Store.

Enabling the JDBC or ODBC Adapter to be used with the Oracle Credential Store

If you intend to use the JDBC or ODBC adapter with the Oracle Credential Store, you must edit `DataIngest.xml` to specify the credentials map, jpsconfig file path and opss directory path.

To do this, modify the following tags under "Forge" component definition in `DataIngest.xml`:

- `credentials-map` - Specify the map name specified while adding the credentials information to Oracle Credential Store. The associated credentials key name has to be specified while creating the pipeline.
- `jps-config-path` - Specify the absolute path name to the JPS Config xml file.
- `opss-jars-dir` - Specify the absolute path name to the directory where OPSS jar files are located.

Example

```
 <forge id="Forge" host-id="ITLHost">
    . . .
    <credentials-map>mymap</credentials-map>

    <jps-config-path>C:\ToolsAndFrameworks\11.1.0\server\workspace\creden¬
tial_store\jps-config.xml</jps-config-path>

    <opss-jars-dir>C:\PlatformServices\11.1.0\lib\java\opss</opss-jars-dir>
    . . .
 </forge>
```

In addition, while configuring ODBC adapter, you need to specify an additional argument tag to place `$ENDE¬ CA_ROOT\lib\java\adapter.jar` in the java classpath.

`To do this, add the following elements to the Forge Component definition in DataIngest.xml:`

Example:

```
 <forge id="Forge" host-id="ITLHost">
    . . .
   <args>
      . . .
      <arg>--javaClasspath</arg>
      <arg>C:\PlatformServices\11.1.0\lib\java\adapter.jar</arg>
      . . .
    </args>
    ...
  </forge>
```

Example: Using generateSSLCertificates

The section illustrates how to use `generateSSLCertificates` to generate certificates for two hosts in a distributed environment.

To create a certificate using the `generateSSLCertificates` utility, follow these steps (Prompts are shaded and user input is in boldface):

1. Execute the `generateSSLCertificates` utility as follows:

   ```
   On Windows: generateSSLCertificates.bat
   On Unix:    generateSSLCertificates.sh
   ```

2. Specify the location of the OCS configuration file that will contain keystore and truststore passphrases.

   ```
   Enter the location of OCS JPS_CONFIG file:
    /localdisk/endeca/ToolsAndFrameworks/11.1.0/server/workspace/credential_store
   ```

3.
   ```
   Please specify the location of an existing openssl executable. If you do not
   have one available,
   please consult the Oracle Documentation on how it might be obtained...

   Specify the complete path of directory containing openssl executable:
    :/localdisk/endeca/PlatformServices/11.1.0/bin/openssl
   ```

4. `The following message appears if no CA root certificate is available:`

   ```
   There is no CA root certificate available in
   deployment_template\ssl_certs_utility\bin\ssl directory; generating self-signed
    CA root certificate.
   The CA root certificate will expire after 3600 days.
   ```

5.
```
Please enter the hostname.domain of the server (For example "hostname.domainname"
 of
Platform services) for which certificates are needed. :
slcw5dd.us.example.com

```
```
Generating keys and certificates for slcw5dd.us.example.com
 components.
```
6.
```
The keystore for slcw5dd.us.example.com host will be protected using a
passphrase.
Select any passphrase you wish (6 or more characters)
  Enter passphrase for KeyStore: ******
  Re-enter your KeyStore passphrase: ******
```
7.
```
Enter the keyname to store this keystore passphrase in oracleCom¬
merceSSLPassPhrase map of OCS.
  :S-slcw5dd
```
8.
```
Would you like to set a different passphrase for Truststore? (Y/N)
The typical default is N to reuse new keystore passphrase.
 :Y
```
9. Enter a passphrase for Truststore.
```
Enter passphrase for truststore for for  slcw5dd.us.example.com host
  Enter passphrase for TrustStore: ******
  Re-enter your TrustStore passphrase: ******
```
10. Add the Truststore to OCS.
```
Enter the keyname to store this truststore passphrase in oracleCom¬
merceSSLPassPhrase map of OCS
 : TS-slcw5dd
```
11. Instruct `generateSSLCertificates` to generate certificates for other hosts. (optional)
```
Would you like to generate certificates for any other host? (Y/N)?
The typical default is N to exit the utility.
 :Y
```
12. If you entered Y in the preceding step, specify the hostname and domain of the second host for which you want to create a certificate; for example:
```
Please enter the hostname.domain of the server for which certificates are
needed.
 :busgt5706.oradev.oraclecorp.com
```
```
Generating keys and certificates for busgt5706.oradev.oraclecorp.com
```
13. Enter and confirm the passphrase that you want to use for the keystore:
```
keystore for busgt5706.oradev.oraclecorp.com host will be protected using a
passphrase.
Select any passphrase you wish (6 or more characters)
  Enter passphrase for KeyStore:*******
  Re-enter your KeyStore passphrase:*******
```

14.
```
Enter the keyname to store this keystore passphrase in oracleCom¬
merceSSLPassPhrase
map of OCS: **KS-busgt5706**
```

15. Specify a different passphrase for the Trust store. (optional)

```
Would you like to set a different passphrase for Truststore? (Y/N)
The typical default is N to reuse the keystore passphrase.
 :Y
```

16.
```
 Enter passphrase for truststore for busgt5706.oradev.oraclecorp.com host.
  Enter passphrase for TrustStore:******
  Re-enter your TrustStore passphrase:******
```

17.
```
Enter the keyname to store this truststore passphrase in oracleCom¬
merceSSLPassPhrase
map of OCS :TS-busgt5706

The host busgt5706.oradev.oraclecorp.com certificates will expire after 1095
days.
```

18. Indicate whether you want to generate certificates for additional hosts or exit the `generateSSLCertifi¬cates` Utility.

```
Would you like to generate certificates for any other host? (Y/N)?
The typical default is N to exit the utility.:N
```

Chapter 5

Generating Certificates with enecerts

This section describes how to generate certificates using the enecerts utility.

Generating SSL certificates

Oracle Commerce Guided Search implementations that do *not* use the Assembler can run the MDEX Engine under SSL.

Note: Use of the `enecerts` utility described in this section is deprecated. Oracle recommends that you use the `generateSSLCertificates` utility instead. The root certificate created by enecerts cannot be used with certificates generated by the generateSSLcertificates utility. For more information, see *Creating SSL Certificates* on page 15.

The `enecerts` utility generates new SSL certificate files. The `enecerts` utility resides in the `$ENDECA_MDEX_ROOT/bin` directory (`%ENDECA_MDEX_ROOT%\bin` on Windows) under the name `enecerts` (`enecerts.exe` on Windows).

The two typical scenarios for generating SSL certificates are:

- You are setting up SSL for the first time and need to generate the set of standard certificates.
- You want to generate custom certificates, such as those with a private key size greater than the default 1024 bits.

Generating custom certificates

You can use the `enecerts` utility to generate customized certificates.

You can generate two types of customized certificates by:

- Specifying a private key size larger or smaller than the default 1024-bit size.
- Using your own Certificate Authority (CA) file and private key to generate the `eneCert.pem` certificate.

The next two sections describe these operations.

Specifying a different certificate key size

The `--keysize` flag of the `enecerts` utility enables users to specify the size of the generated private key. The flag syntax is:

```
--keysize bits
```

where *bits* is the private key size in bits. (The default value is 1024.)

For example, the following `Windows` command creates certificates with a private key size of 2048 bits:

```
enecerts --keysize 2048
```

Keep in mind that using larger keys will slow system performance. A recommended alternative to the default 1024-bit size is a key size of 512 bits, which will give you a good balance between security and performance considerations.

Using your Certificate Authority file to generate certificates

By default, the `enecerts` utility produces the `eneCert.pem` certificate (used by all clients and servers to specify their identity when using SSL) and the `eneCA.pem` Certificate Authority (CA) certificate (used by all clients and servers that wish to authenticate the other endpoint of a communication channel).

If you have your own CA certificate and private-key files, you can use the `--CAkey` and `--CAcert` flags to generate the `eneCert.pem` certificate. The private-key file (.key extension) is used to digitally sign the public key that is generated by the `enecerts` utility. Both flags must be used for this operation.

The syntax for the `--CAkey` flag is:

```
--CAkey private-key
```

where *private-key* is your own .key file with the private key for the CA that should be used to sign the generated certificate.

The syntax for the `--CAcert` flag is:

```
--CAcert cert-pem
```

where *cert-pem* is your CA certificate (.pem extension). This file is the same type of file as the default `eneCA.pem` CA certificate.

For example, the following Windows command creates a signed certificate file using your own CA certificate and private-key files:

```
enecerts --CAkey myCA.key --CAcert myCA.pem
```

You would then use the resulting `eneCert.pem` certificate and your CA file (`myCA.pem` in the example) to configure SSL for your Guided Search components. If you have multiple machines in your deployment, you must also copy these files to the other machines.

Selecting an Encryption Algorithm

The MDEX Engine can optionally be configured to require encryption when other components of your Oracle Commerce Guided Search implementation communicate with it.

Encryption is required whenever the the `-sslcertfile` option is used with the `dgraph` command that starts the MDEX Engine. For information about the `dgraph` command and its options, refer to the *Oracle Commerce Guided Search Administrator's Guide*.

Whenever encryption is required, the MDEX Engine and the client with which it is negotiating a connection together choose an appropriate encryption algorithm from Oracle's approved list of algorithms.

However, you may want to limit the available choices to a specific algorithm or algorithms on the approved list. To do this, you can specify the algorithm or algorithms in configurations or on the command line where encryption algorithms are accepted. If you specify more than one algorithm, the component and the MDEX Engine will negotiate and decide which one to use.

You specify the algorithms by their standard names, such as `DHE-RSA-AES256-SHA`. If you specify more than one algorithm, you must separate their names with colons; for example: `DHE-RSA-AES256-SHA:DHE-DSS-AES256-SHA`.

Each Guided Search component uses its own syntax for accepting specific algorithms as input. For example, the dgraph command uses the `--sslcipher` option, as follows:

```
dgraph --sslcipher DHE-RSA-AES256-SHA:DHE-DSS-AES256-SHA
```

Note: You can also specify encryption algorithms not on the Oracle-approved list in Appendix A, provided that these algorithms are supported by the client with which the MDEX Engine is communicating. In this case, however, the encryption is not guaranteed to be secure.

Chapter 6

Enabling SSL Communication for the Deployment Template

This section describes the steps that you must follow to enable SSL communication to and from the Deployment Template.

Enabling SSL Communication between the Deployment Template and SSL-enabled components

Follow these steps to enable secure SSL communication between the Deployment Template and the EAC Central Server, the ECR, the Assembler, the Dgraph (the MDEX engine) and the Content Acquisition System (CAS) version 3.0.x and later:

1. Create a Java keystore and truststore to contain your certificates. For information about how to do this, see *Storing generateSSLCertificates Credentials* on page 18.
2. Upload a copy of these certificates to the server on which your Deployment Template scripts will run.
3. Update `runcommand.bat/.sh` to load your SSL keystore and truststore.
    - On Windows, edit `runcommand.bat` to contain the following lines:

```
...
set JAVA_ARGS=%JAVA_ARGS% "-Djava.util.logging.config.file=%~dp0..\con¬
fig\script\logging.properties"

if exist [\path\to\truststore] (
set TRUSTSTORE=[\path\to\truststore]
) else (
echo WARNING: Cannot find truststore at [path\to\truststore]. Secure EAC
communication may fail.
)

if exist [\path\to\keystore] (
set KEYSTORE=[\path\to\keystore]
) else (
echo WARNING: Cannot find keystore at [\path\to\keystore]. Secure EAC com¬
munication may fail.
)

set JAVA_ARGS=%JAVA_ARGS% "-Djavax.net.ssl.trustStore=%TRUSTSTORE%" "-
Djavax.net.ssl.trustStoreType=JKS"
```

```
set JAVA_ARGS=%JAVA_ARGS% "-Djavax.net.ssl.keyStore=%KEYSTORE%" "-
Djavax.net.ssl.keyStoreType=JKS"

set JAVA_ARGS=%JAVA_ARGS% "-Djavax.net.ssl.trustStorePassword=TS-password"

set JAVA_ARGS=%JAVA_ARGS% "-Djavax.net.ssl.keyStorePassword=KS-password"

set OCS_ARGS="-Dcom.endeca.ssl.jpsConfigPath=[\path\to\jps-config.xml]"

"-Dcom.endeca.ssl.storeMapName=[Map Name]" "-Dcom.endeca.ssl.trustStoreKey¬
Name=[TS-Passphrase]"
"-Dcom.endeca.ssl.keyStoreKeyName=[KS-Passphrase]"

set JAVA_ARGS=%JAVA_ARGS% %OCS_ARGS%

set CONTROLLER_ARGS=--app-config AppConfig.xml
...
```

Note: In `runcommand.bat`, set commands must not contain line breaks. In the preceding example, the set commands are wrapped only to fit the page size of this document.

- On UNIX, edit `runcommand.sh` to contain the following lines:

```
...
JAVA_ARGS="${JAVA_ARGS} -Djava.util.logging.config.file=${WORKING_DIR}/../con¬
fig/script/logging.properties"
if [ -f "[/path/to/truststore]" ] ; then
if [ -f "[/path/to/keystore]" ] ; then
TRUSTSTORE=[/path/to/truststore]
KEYSTORE=[/path/to/keystore]
JAVA_ARGS="${JAVA_ARGS} -Djavax.net.ssl.trustStore=${TRUSTSTORE}"
JAVA_ARGS="${JAVA_ARGS} -Djavax.net.ssl.trustStoreType=JKS"
JAVA_ARGS="${JAVA_ARGS} -Djavax.net.ssl.keyStore=${KEYSTORE}"
JAVA_ARGS="${JAVA_ARGS} -Djavax.net.ssl.keyStoreType=JKS"
JAVA_ARGS="${JAVA_ARGS} -Djavax.net.ssl.trustStorePassword=TS-password"
JAVA_ARGS="${JAVA_ARGS} -Djavax.net.ssl.keyStorePassword=KS-password"
OCS_ARGS="-Dcom.endeca.ssl.jpsConfigPath=[/path/to/jps-config.xml]"
"-Dcom.endeca.ssl.storeMapName=[Map Name]" "-Dcom.endeca.ssl.trustStoreKey¬
Name=[TS-Passphrase]"
"-Dcom.endeca.ssl.keyStoreKeyName=[KS-Passphrase]"
JAVA_ARGS=${JAVA_ARGS} ${OCS_ARGS}
else
echo "WARNING: Cannot find keystore at [/path/to/keystore]. Secure EAC
communication may fail."
fi
else
echo "WARNING: Cannot find truststore at [/path/to/truststore]. Secure EAC
communication may fail."
fi
CONTROLLER_ARGS="--app-config AppConfig.xml"
...
```

4. In the app element of the `AppConfig.xml` document, update the `sslEnabled` attribute to `true`.

   The `sslEnabled` attribute is a application-wide setting that applies to the EAC and to CAS (if used in your application).

5. Specify the SSL-enabled port for the EAC.

The Endeca HTTP Service uses a separate port to communicate securely. For example, the default non-SSL connector is on port 8888 and the default SSL connector listens on port 8443. The SSL port should be specified in the `eacPort` attribute of the app element in the `AppConfig.xml` document.

6. If you are using CAS in your application, specify the SSL-enabled port for CAS.

   The CAS Service uses a separate port to communicate securely. For example, the default non-SSL port is 8500 and the default SSL port is 8505. The SSL port should be specified in the `value` attribute of `cas¬Port`.

   The following example shows a sample configuration for an SSL-enabled application.

```
  <!--
    ########################################################################
    # EAC Application Definition
    #
  -->
  <app appName="test" eacHost="slcw5dd.us.oracle.com" eacPort="8443"
      dataPrefix="test" sslEnabled="true" lockManager="LockManager">
    <working-dir>${ENDECA_PROJECT_DIR}</working-dir>
    <log-dir>./logs</log-dir>
  </app>

  <!--
    ########################################################################
    # Lock Manager - Used to set/remove/test flags and obtain/release locks
    #
  -->
  <lock-manager id="LockManager" releaseLocksOnFailure="true" />

<!--
########################################################################
# Content Acquisition System Server
#

<custom-component id="CAS" host-id="CASHost" class="com.Oracle Endeca.eac.toolk¬
it.component.cas.ContentAcquisitionServerComponent">
  <properties>
    <property name="casHost" value="slcw5dd.us.oracle.com" />
    <property name="casPort" value="8505" />
  </properties>
</custom-component>

-->
```

7. In `WorkbenchConfig.xml`, update the protocol to https in the `repositoryUrl` property of the IFCR custom-component; for example:

```
<custom-component id="IFCR" host-id="ITLHost" class="com.endeca.soleng.eac.toolk¬
it.component.IFCRComponent">
  <properties>
     <property name="repositoryUrl" value="https://busgt5706.oradev.ora¬
clecorp.com:8446/ifcr" />
      ...
  </properties>
<custom-component>
```

8. In `AuthoringDgraphCluster.xml` , set the `useSsl` property of the `<host>` element to "false" or "true" to disable or enable SSL communication:

Disable SSL:

```
<host id="AuthoringMDEXHost" hostName="FullyQualifiedHostName" port="8888"
useSsl="false"/>
```

Enable SSL:

```
<host id="AuthoringMDEXHost" hostName="FullyQualifiedHostName"  port="8443"
useSsl="true"/>
```

9. In `LiveAppServerCluster .xml`, set the `useSsl` property of the `<web-app>` element under the `<app-server>` element to "false" or "true" to disable or enable SSL communication:

   Disable SSL:

```
<app-server id="LiveDiscover" hostName="FullyQualifiedHostName" port="8006">
  <web-app id="DiscoverWebApp1" contextPath="/discover" sslEnabled="false"/>
  ...
</app-server>
```

   Enable SSL:

```
<app-server id="LiveDiscover" hostName="FullyQualifiedHostName" port="8446">
  <web-app id="DiscoverWebApp1" contextPath="/discover" sslEnabled="true"/>
  ...
</app-server>
```

10. In `ReportGeneration.xml`, set the `useSsl` property of the `<host>` elements shown below to "false" or "true" to disable or enable SSL communication:

    Disable SSL:

```
<host id="ReportGenerationHost" hostName="FullyQualifiedHostName" port="8888"
 useSsl="false"/>
<host id="WorkbenchHost" hostName="FullyQualifiedHostName" port="8888"
useSsl="false">
  . . .
</host>
```

    Enable SSL:

```
<host id="ReportGenerationHost" hostName="FullyQualifiedHostName" port="8443"
 useSsl="true"/>
<host id="WorkbenchHost" hostName="FullyQualifiedHostName" port="8443"
useSsl="true">
  . . .
</host>
```

Chapter 7

Enabling SSL for the Log Server, Dgraph, emgr_update, and Forge

This section describes how to enable secure two-way SSL communication for the Log Server, the Dgraph (the MDEX engine), emgr_update, and Forge.

Configuring Log Server, Dgraph, emgr_update, and Forge for SSL

To enable SSL communication for the LogServer, Dgraph, emgr_update, and Forge, you must edit the appropriate configuration files items to reference the following items:

- Root authentication certificates and host certificates generated by the `generateSSLCertificates` utility. For information about this utility, see *Creating SSL Certificates* on page 15.
- An encryption algorithm such as `AES128-SHA` that has been approved for use by Oracle, Inc. A complete list of the encryption algorithms approved by Oracle can be obtained from Oracle customer support.

To enable SSL communication for LogServer, Dgraph, emgr_update, and Forge, follow these steps:

1. Configure the `sslConfig` Java bean with global SSL configuration parameters. The `sslConfig` java bean should be created for each host separately if any of the components (Dgraph, Forge, or Log Server) are hosted on another machine.
2. Pass SSL configuration parameters to the Log Server, Dgraph, and Forge by editing their respective configuration files to reference the sslConfig Java bean, or to specify the global SSL configuration parameters individually.

The following sections explain these steps in detail.

Configure the sslConfig Java Bean with Global SSL Configuration Parameters

To enable LogServer, Dgraph, emgr_update, and Forge for SSL communication, you must supply appropriate values for the properties of the `<ssl-config>` element in the `AppConfig.xml` file. Supplying properties values for `<ssl-config>` configures the Java bean `sslConfig`, which is the source of SSL configuration data for the LogServer, Dgraph, and Forge.

In the `<ssl-config>` element, specify the global SSL configuration parameters by supplying values for the following element properties:

- certfile: The root authentication certificate generated by `generateSSLCertificates`. (Specify full pathname.)

- cafile: The authentication certificate generated for this host by `generateSSLCertificates`. (Specify full pathname.)
- cipher: Specify `AES128-SHA` , the Oracle-approved encryption algorithm.

The following example illustrates how to specify global SSL configuration parameters in the `<ssl-config>` element of the `AppConfig.xml` file:

```
<!--
  ######################################################################
  # Configure the Java bean sslConfig with values that you can then pass
  # to Forge, Dgraph, LogServer, and custom components.
  #
-->
  <ssl-config id="globalSslConfig">
    <property name="certFile"
      value="/localdisk/endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
plate/ssl_certs_utility/bin/ssl/slcw5dd.us.example.com.pem"/>
    <property name="caFile"
      value="/localdisk/endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
plate/ssl_certs_utility/bin/ssl/ca-cert.pem"/>
    <property name="cipher" value="AES128-SHA"/>
   </ssl-config>
```

Passing SSL Configuration Parameters to the Log Server

To enable SSL communication for the Log Server, you must edit the `<logserver>` element of the `Report¬Generation.xml` file to reference the Java bean (sslConfig) that contains the global SSL configuration parameters. In the `<logserver>` element, you must specify appropriate values for the properties of the `<ssl-config>` subelement, as follows:

- bean: Specify "`sslConfig`", the name of the Java bean that contains the global SSL parameters.
- ref: Specify the value of the id parameter of the `<ssl-config>` element of `AppConfig.xml`.

The following example illustrates how to reference the Java bean that contains the global SSL communication parameters:

```
<logserver id="LogServer" host-id="ReportGenerationHost" port="15010">
  . . . .
  <gzip>false</gzip>
  <ssl-config bean="sslConfig" ref="globalSslConfig"/>
</logserver>
```

Passing SSL Configuration Parameters to Dgraph

To enable SSL communication for the Dgraph, you can modify the `<Dgraph>` element of the `AuthoringD¬graph.xml` configuration file in either of two ways:

- Edit the `<ssl-config>` subelement of the `<Dgraph>` element to reference the `sslConfig` Java bean, or
- Specify the `sslConfig` parameters individually, without referencing sslConfig itself.

The following example illustrates how to edit the `<ssl-config>` element to reference the sslConfig Java bean. Note that the `ref` property of `<ssl-config>` must be set to the value of the `id` property in the `<ssl-config>` element of the `AppConfig.xml` file:

```
<dgraph id="AuthoringDgraph" host-id="AuthoringMDEXHost" port="15002"
  post-startup-script="AuthoringDgraphPostStartup">
   . . . .
   <input-dir>./data/dgraphs/AuthoringDgraph/dgraph_input</input-dir>
   <update-dir>./data/dgraphs/AuthoringDgraph/dgraph_input/updates</update-dir>

   <ssl-config bean="sslConfig" ref="globalSslConfig"/>
</dgraph>
```

The following example illustrates how to edit the `<dgraph>` element to specify the `sslConfig` parameters individually, without referencing `sslConfig` itself:

```
<dgraph id="AuthoringDgraph" host-id="AuthoringMDEXHost"
   port="15002" post-startup-script="AuthoringDgraphPostStartup">
   . . .
   <input-dir>./data/dgraphs/AuthoringDgraph/dgraph_input</input-dir>
   <update-dir>./data/dgraphs/AuthoringDgraph/dgraph_input/updates</update-dir>

   <cert-file>/localdisk/endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
plate/ssl_certs_utility/bin/ssl
     /slcw5dd.us.example.com.pem</cert-file>
   <ca-file>/localdisk/endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
plate/ssl_certs_utility/bin/ssl
     /ca-cert.pem</ca-file>
   <cipher>AES128-SHA</cipher>
</dgraph>
```

Passing SSL configuration parameters for the WorkbenchManager Component

If your Workbench is running in two-way SSL mode, follow these steps to enable the `emgr_`update script:

1. Configure an `sslConfig` Java bean with the parameters `certFile` , `caFile`, and `cipher`.

   Note: You can reuse an existing global `sslConfig` Java bean or define a new `sslConfig` bean.

2. Add the `<ssl-config>` tag to the custom component `WorkbenchManager` in `WorkbenchConfig.xml` with the SSL configuration. For example:

   ```
   <ssl-config bean="sslConfig" ref="globalSslConfig"/>
   ```

   Adding the `<ssl-config>` tag enables the emgr scripts to run in SSL mode.

Passing SSL Configuration to Forge

In the same way, you can pass SSL configuration parameters to Forge by editing the `<forge>` element of the `DataIngest.xml` configuration file either to reference the Java bean sslConfig or to specify the `sslConfig` parameters individually, without referencing `sslConfig` itself.

The following example illustrates how to edit the `<ssl-config>` subelement of the `<forge>` element to reference the `sslConfig` Java bean. Note that the `ref` property of the `<ssl-config>` element must be set to the value of the `id` property in the `<ssl-config>` element of the `AppConfig.xml` file:

```
<forge id="Forge" host-id="ITLHost">
   . . .
   <pipeline-file>./data/processing/pipeline.epx</pipeline-file>
   <ssl-config bean="sslConfig" ref="globalSslConfig"/>
</forge>
```

The following example illustrates how to edit the `<forge>` element to specify the `sslConfig` parameters individually, without referencing sslConfig itself:

```
<forge id="Forge" host-id="ITLHost">
  . . . .
  <pipeline-file>./data/processing/pipeline.epx</pipeline-file>
  <cert-file>/localdisk/endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
plate/ssl_certs_utility/bin/ssl/slcw5dd.us.example.com.pem</cert-file>
  <ca-file>/localdisk/endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
plate/ssl_certs_utility/bin/ssl/ca-cert.pem</ca-file>
```

```
  <cipher>AES128-SHA</cipher>
</forge>
```

Running Parallel Forge Processes in SSL and non-SSL Modes

To increase the efficiency with which your application processes source data, you can create groups of separate Forge processes that run in parallel with each other. Parallel Forge processing can increase processing efficiency when there is a large amount of source data to process or when the source data comes from multiple sources.

You can enable SSL communication for parallel Forge processes by specifying values for the following flags in the Forge commands that start the Forge server and each of the Forge clients:

- sslcertfile: The root authentication certificate generated by `generateSSLCertificates`. (Specify full pathname; for example: `/localdisk/endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬plate/ssl_certs_utility/bin/ssl/slcw5dd.us.example.com.pem`
- sslcafile: The authentication certificate generated for this host by `generateSSLCertificates`. (Specify full pathname; for example: `/localdisk/endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬plate/ssl_certs_utility/bin/ssl/ca-cert.pem`.)

Only certificates generated by the `generateSSLCertificates` utility can be used for `sslcertfile` and `sslcafile`. For information about how to use the `generateSSLCertificates` utility, see *Creating SSL Certificates* on page 15.

The following examples illustrate Forge commands that start a Forge server and two Forge clients in SSL mode. For information about Forge command flags, refer to the *Oracle Commerce Forge Guide*.

Command to run Forge server

```
forge -vi -o out.log_server --dtdInHeader false -c "client_val=server"
    --usingManager --javaArgument -Xmx256m --logLevel DEBUG --server
    <server-port> --numClients 2
    --sslcertfile <server host cert>
    --sslcafile <root ca cert>

    --outputDir <output directory location>
    --stateDir <state directory location> pipeline.epx
```

Command to run first Forge client

```
forge -vi -o out.log_0 --dtdInHeader false --usingManager
    --javaArgument -Xmx256m --logLevel DEBUG -c "client_val=instance0" --client
    10.152.105.73:1234 --clientNum 0
    --sslcertfile <client host cert>
    --sslcafile <root ca cert>
    --outputDir <output directory location>
    --stateDir <state directory location> pipeline.epx
```

Command to run second Forge client

```
forge -vi -o out.log_1 --dtdInHeader false --usingManager
    --javaArgument -Xmx256m --logLevel DEBUG -c "client_val=instance1" --client
    10.152.105.73:1234 --clientNum 1
    --sslcertfile <client host cert>
    --sslcafile <root ca cert>
    --outputDir <output directory location>
    --stateDir <state directory location> pipeline.epx
```

Chapter 8

Modifying MDEX Security Defaults (optional)

This section describes how you can override the default security configuration for MDEX. Note that this is optional, because the default configuration is appropriate for most purposes.

Overriding Default Dgraph Configuration

The EAC stores default configuration that specifies the host, port, and SSL configuration (ca-file, cert-file and cipher) of the Dgraph, and enables SSL communication. Editors and the Relevance Ranking Evaluator (RRE) that run within the Workbench container access Dgraph configuration from the EAC and establish secure or non-secure communication.

The default configuration for DGraph is accessed from the EAC automatically whenever the Dgraph is started. By default, all editors and the Relevance Ranking Evaluator:

- Use the first authoring Dgraph
- Support secure SSL communication with the Dgraph

These defaults are suitable for most purposes. However, you can override them by editing the following properties in the appropriate configuration files:

- `Dgraph_id`, or
- `host` and `port` (for backward compatibility)

The configuration files in which you can edit these values are the following:

- `endecaBrowserService.json`: Used by `SpotlightSelectionEditor`, `BoostBuryRecordEditor`, and `LinkBuilderEditor`.
- `editors.xml`: Used by the MediaEditor.
- `evaluator.properties`: Used by the Relevance Ranking Evaluator. The values specified in this file apply globally to all applications.

Specifying Non-Default Values in endecaBrowserService.json

The following `endecaBrowserService.json` file illustrates the use of the `dgraphID` key to specify a non-default Dgraph (MyDgraph):

```
{
"dgraphID":"MyDgraph",
"recDisplayNameProp":"product.name",
"recSpecProp":"common.id",
. . .
"textSearchMatchMode":"ALLPARTIAL"
}
```

The following `endecaBrowserService.json` file illustrates the use of the `host` and `port` keys to specify a non-default Dgraph:

```
{
"host":"slcw5dd.us.example.com",
"port":"15000","sslEnabled":"true",
"recDisplayNameProp":"product.name",
"recSpecProp":"common.id",
.  . .
"textSearchMatchMode":"ALLPARTIAL"
}
```

Note: If both `Dgraph_id`, and `host` and `port`, are defined in a configuration file, the `Dgraph_id` value is used, rather than the `host` and `port` values. In this case, if you do not specify the `sslEnabled` key, it defaults to false and non-ssl communication is established between editors and Dgraph.

Specifying Non-Default Values in editors.xml

The following excerpt from `editors.xml` specifies that the MediaEditor is to use a non-default Dgraph whose ID is `Dgraph1`:

```
...
<Editor name="editors:MediaEditor">
  <EditorConfig>
    <useMediaBrowser>true</useMediaBrowser>
    ...
    <mediaApplicationID>media</mediaApplicationID>
    <mediaDgraphID>Dgraph1</mediaDgraphID>
    ...
  </EditorConfig>
</Editor>
...
```

The following excerpt from `editors.xml` specifies that SSL is enabled for communication between the MediaEditor and the Dgraph, and that the MediaEditor is to use a non-default Dgraph at port 17000 on the host `slcw5dd.us.example.com`.

```
...
<Editor name="editors:MediaEditor">
  <EditorConfig>
     <useMediaBrowser>true</useMediaBrowser>
     ...
     <sslEnabled>true</sslEnabled>
     <mdexPort>17000</mdexPort>
     <mdexHost>slcw5dd.us.example.com</mdexHost>
     ...
  </EditorConfig>
</Editor>
...
```

Note: If your media application id is different from "media", specify the media application id in a `<medi¬ aApplicationID>`... `</mediaApplicationID>`element.

Specifying Non-Default Values in evaluator.properties

The following excerpt from `evaluator.properties` selects a non-default Dgraph for the Relevance Ranking Evaluator to communicate with by specifying the ID of that Dgraph (`Dgraph1`) as the value of the `DGRAPH_ID` key:

```
...
DGRAPH_ID=Dgraph1
...
```

Note: SSL communication can be neither enabled or disabled by the `evaluator.properties` file.

Overriding Configuration for MDEX at Run-Time

You can override the default configuration for a specific MDEX at runtime, using the Relevance Ranking Evaluator as follows:

1. Start the Relevance Ranking Evaluator.
2. Click **Change MDEX Engine**. The Specify MDEX Engine dialog appears. In this dialog, you can override defaults for the Dgraph in either of two ways:

   - Click **Using DGraphID** (the default)

     1. Select the DGraph to configure from the drop-down list labeled Dgraph ID. (The list shows only the DGraphs that are bound to the current active application.)
     2. Click **Submit**.

   - Click **Using Host and Port**

     1. For **Host**, enter the host name; for example: `slcw5dd.us.example.com`.
     2. For **Port**, enter the number of the port on this host that you intend to use; for example: 15002.
     3. Select either `True` or `False` for **Enable SSL**.
     4. Click **Submit**.

Chapter 9

Enabling SSL Communication for Workbench

This section describes the steps that you must follow to enable a two-way SSL communication between Workbench and other components of your Oracle Commerce Guided Search application.

Steps for Enabling SSL Communication for Workbench, Assembler, and Tools and Frameworks

Follow these steps to enable SSL communication to and from the Workbench, the Assembler, and Tools and Frameworks:

Note: In several lines of the following sample code, line breaks (denoted by the symbol ~) are inserted to make the lines fit the width of page. Do not insert line breaks into lines of actual working code.

1. Make the following changes in `%ENDECA_TOOLS_CONF%\conf\server.xml`:
   a. Comment the HTTP connector as follows:
   ```
   <!-- <Connector port="8006" protocol="HTTP/1.1" connectionTimeout="20000"~
     maxPostSize="0" redirectPort="8446" URIEncoding="UTF-8"/> -->
   ```
   b. Remove comments from the HTTPS connector. Then set the `clientAuth` property to "true", and specify values for the `keyStoreFile` and `trustStoreFile` properties; for example:
   ```
   <Connector port="8446" SSLEnabled="true"
   protocol="org.apache.coyote.http11.Http11Protocol" maxPostSize="0"
   maxThreads="150" scheme="https" secure="true"
   clientAuth="true"
   sslProtocol="TLS"
   keystoreFile="/localdisk/endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
   plate/ssl_certs_utility/bin/ssl
     /slcw5dd.us.example.com.ks"
   keystorePass="eacpass"
   truststoreFile="/localdisk/endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
   plate/ssl_certs_utility/bin/ssl
     /TS-slcw5dd.us.example.com.ks"
   truststorePass="eacpass" URIEncoding="UTF-8"/>
   ```
2. Make the following changes in `%ENDECA_TOOLS_CONF%\webstudio.properties`:

a. If the EAC server is running over SSL, update the EAC server host and port properties; for example:

```
  # The EAC Central Server that this Workbench uses
  com.endeca.webstudio.eac.hostname=slcw5dd.us.example.com
  com.endeca.webstudio.eac.port=8443
  com.endeca.webstudio.eac.useHttps=true
```

b. Set the flag to true to enable SSL communications to and from T&F:

```
  # The SSL settings for connecting to SSL-enabled Components
  # like EAC, MDEX
  com.endeca.webstudio.client.communication.ssl=true
```

c. Uncomment following snippet to enable either plain text passphrases or credential store [OCS] passphrases:

```
# Configure your key store and trust store information here.
    # Note : javax.net.ssl.trustStorePassword and
    javax.net.ssl.keyStorePassword must not
   # be used in case of using OCS to store trust Store/ key store  passwords.


    javax.net.ssl.trustStore=C:/Endeca/ToolsAndFrameworks/11.1.0~
      /deployment_template/ssl_certs_utility/bin/ssl/TS-slcw5dd.us.exam¬
ple.com.ks
    javax.net.ssl.trustStoreType=JKS
    #javax.net.ssl.trustStorePassword=eacpass

    javax.net.ssl.keyStore=C:/Endeca/ToolsAndFrameworks/11.1.0~
     /deployment_template/ssl_certs_utility/bin/ssl/slcw5dd.us.example.com.ks

    javax.net.ssl.keyStoreType=JKS
    javax.net.ssl.keyStorePassword=eacpass
    #------ Oracle Credential Store configuration to retrieve
    #------ trustStorePassword and keyStorePassword from credential store
    #
    #------ jpsConfigPath - Absolute path to jps-config.xml
    #------ storeMapName - Map name under which trustStorePassword &
    keyStorePassword are pushed into OCS.
    #------ trustStoreKeyName - Key name under which trustStorePassword
    #------ is pushed into OCS.
    #------ keyStoreKeyName - Key name under which keyStorePassword
    #------ is pushed into OCS.

    com.endeca.webstudio.ssl.jpsConfigPath=C:/Endeca/ToolsAndFrameworks~
      /11.1.0/server/workspace/credential_store/jps-config.xml
    com.endeca.webstudio.ssl.storeMapName=oracleCommerceSSLPassPhrase
    com.endeca.webstudio.ssl.trustStoreKeyName=ts-key
    com.endeca.webstudio.ssl.keyStoreKeyName=ks-key
```

Enabling SSL Communication between Workbench and an ATG server

To enable two-way SSL communication between Workbench and an ATG (Oracle Commerce Platform) server, follow these steps:

1. When ATG Server is running in SSL mode, update the protocol and the hostname in the file `<Ende¬ ca_app>/config/editors_config/atgServices.json`. The protocol must be https and the machine name must be the ATG server machine name.

2. Configure the client certificates in the `%ENDECA_TOOLS_CONF%\conf\ webstudio.properties` file; for example:

```
   javax.net.ssl.trustStore=@ENDECA.TOOLS.CONF@/conf/ca.ks
   javax.net.ssl.trustStoreType=JKS
   javax.net.ssl.trustStorePassword=eacpass
   javax.net.ssl.keyStore=@ENDECA.TOOLS.CONF@/conf/eac.ks
   javax.net.ssl.keyStoreType=JKS
   javax.net.ssl.keyStorePassword=eacpass
   javax.net.ssl.com.endeca.webstudio.client.communication.ssl="true"
```

3. Make sure that the root certificates of the ATG server are present in the client's trust store, and that the root certificates of the client applcation are in the ATG server's trust store. Root certificates can be specified either as external path entries or in the `cacerts` file.

Note: If Oracle Commerce Guide Search and ATG certificates are generated using same certificate authority, it is not necessary to export and import certificates from one machine to the other.

4. The following command can be used to export and import the root certificates from the machines.

- To export a root certificate, use a keytool command of the following form:

```
keytool -export -alias alias_used_during_certificate_gen -file ROOT_CERT.cer
-keystore truststore.ks
```

  where:

`alias` is the name of the alias used during certificate generation.
`file` is the name of the root certificate file that is to be exported.
`keystore` is the file name of the trust store to which the root certificate is exported.

- To import a root certificate, use a keytool command of the following form:

```
keytool -importcert -alias ca_root -file ROOT_CERT.cer -keystore TRUSTSTORE.ks
```

  or

```
keytool -importcert -alias ca_root -file ROOT_CERT.cer -keystore cacerts
```

Enabling SSL Communication between Workbench and Developer Studio

To enable two-way SSL communication between Workbench, Developer Studio, and the `emgr_update` script, follow these steps:

1. Uncomment the HTTP connector and SSL HTTP connector in the `%ENDECA_TOOLS_CONF%\conf\serv¬er.xml` file of the ToolsAndFrameworks Tomcat instance, as follows:

```
    <!-- A "Connector" represents an endpoint by which requests are
      received and responses are returned. Documentation at:
      Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)
      Java AJP  Connector: /docs/config/ajp.html
      APR (HTTP/AJP) Connector: /docs/apr.html
      Define a non-SSL HTTP/1.1 Connector on port 8080
      Disable maxPostSize, to avoid 2Mb limit on HTTP POST requests. -->

    <Connector port="8006" protocol="HTTP/1.1"
      connection timeout="20000"
      maxPostSize="0"
      redirector="8446"
      URIEncoding="UTF-8"/>

    <!-- Define an SSL HTTP/1.1 Connector on port 8446
     This connector uses the JSSE configuration, when using APR,
     the connector should be using the OpenSSL style configuration
     described in the APR documentation. -->

    <Connector port="8446"
      SSLEnabled="true"
      protocol="org.apache.coyote.http11.Http11Protocol"
      maxPostSize="0"
      maxThreads="150"
      scheme="https"
      secure="true"
      clientAuth="true"
      sslProtocol="TLS"
     keystoreFile="D:/localdisk/endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
plate
        /ssl_certs_utility/bin/ssl/slcw5dd.us.example.com.ks"
      keystorePass="eacpass"
      truststoreFile="D:/localdisk/endeca/ToolsAndFrameworks/11.1.0/deploy¬
ment_template
        /ssl_certs_utility/bin/ssl/TS-slcw5dd.us.example.com.ks"
      truststorePass="eacpass"
      URIEncoding="UTF-8"/>
```

2. Add the following `<security-constraint>` element to the `WEB-INF\web.xml` configuration file in the WAR file `%ENDECA_TOOLS_CONF%\..\webapps\ifcr-11.1.0.war`:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Entire Site</web-resource-name>
    <url-pattern>/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Chapter 10

Securing the Assembler Admin Servlet

This section describes how to provide secure communications with your web application's Admin servlet using the BASIC authentication mechanism.

Configuring Secure Access to the Admin Servlet

You can use the BASIC authentication mechanism to provide secure communication between your web application's Admin servlet and the `AssemblerUpdateComponent` and the Usage collector components of the Deployment Template.

To establish secure access to the Admin servlet, follow these steps:

1. Before modifying the configuration of the Endeca Application Controller (EAC), configure BASIC authentication for the Admin servlet using the standard J2EE mechanism.
2. Add the BASIC authentication credentials to the credential store. EAC components reference these credentials from the credential store to authenticate the Admin servlet. For example, if you specified `webAppAdmin` (user name) and `complexP@ssword` (password) as the BASIC credentials, you can follow these steps to add them to the credentials store:
   a) Go to the `credential_store\bin` directory of your Tools and Framework installation.
   b) Run following command: `manage_credentials add --key webAppAdminCredentialsKey --user webAppAdmin`
   c) When prompted, enter the password `complexP@ssword` from Step 2. The following output appears on the console:

```
manage_credentials add --key webAppAdminCredentialsKey --user webAppAdmin

Enter password for user webAppAdmin :

Re-enter password to conform :
21 Oct 2013 12:43:51,547  INFO CSFHandler:139 - Credential successfully cre¬
ated for map : endecaToolsAndFrameworks.
```

3. Modify LiveAppServerCluster.xml to reference credentials. To do this, follow these steps:
   a) Open `LiveAppServerCluster.xml` in the `config\script` directory of your EAC application.
   b) Add the following code to `LiveAppServerCluster.xml`, to reference the credentials that you created: `<basic-credentials id="webAppAdminCredentials" credentialsStore="csfManager" credentialsKey="webAppAdminCredentialsKey"/>`

c) Modify a `<web-app>` element to enable your web application to reference these BASIC credentials:

```
<web-app id="MyWebApp"
contextPath="/my-web-app"
adminCredentials="webAppAdminCredentials" />
```

4. If you enable SSL in Tools and Framework server where you web application resides, add the following property to the a `<web-app>` element to enable SSL communication:

```
sslEnabled=true
```

5. Follow these steps to verify that the usage collection and promotion mechanism are able to authenticate access to the Admin servlet:

a) Go to the control directory of your EAC application.
b) Run the `collect_usage` command.
c) Verify that usage information is collected in the `logs\usage` directory of your EAC application.
d) Verify that the promotion mechanism works correctly by making changes in your Authoring environment and running the `promote_content` command from the control directory of your EAC application. Verify that your changes are successfully promoted to the live environment.

Chapter 11

Enabling SSL for Platform Services

Oracle security standards require that all keystore and truststore passphrases be stored in the Oracle Credential Store (OCS). This section describes how to configure Platform Services to store its passphrases in the OCS.

Configuring Platform Services for SSL Communication

This section describes how to configure Platform Services to store its passphrases in the OCS and to enable SSL communication for Platform Services.

Note: In several lines of the following sample code, line breaks (denoted by the symbol ~) were inserted to make the lines fit the width of page. Do not insert line breaks into lines of actual working code.

To configure Platform Services for SSL communication, follow these steps:

1. Copy the `credential-store` folder from `%ENDECA_TOOLS_ROOT%\credential_store` and `jps-config.xml` from `%ENDECA_TOOLS_ROOT%\server\workspace\credential_store` into any location in Platform Services.
2. Run the `manage_credentials.bat` utility to store the truststore and keystore Passphrases in the credential store; specify the `jps-config.xml` location using the `--config` option. For information about how to use `manage_credentials.bat`, see *Storing generateSSLCertificates Credentials* on page 18.
3. Set the `sslEnabled` flag in `%ENDECA_CONF%/conf/eac.properties` to true when enabling the EAC server to run over SSL:

```
# Set to true if Eac is sslEnabled
# This flag is used by Eac to communicate with local EacAgent.

com.endeca.eac.sslEnabled=true
```

4. Uncomment the following lines in `%ENDECA_CONF%/conf/eac.properties` to enable either plain text passphrase or credential store (OCS) passphrase. (If both plain text passphrase and credential store (OCS) passphrases are enabled, plain text passphrases are used.)

```
# One of the steps required for enabling SSL is to uncomment the following
# SSL-related configuration options. Please see the documentation for more de¬
tails.

# This must be a JKS key store type
com.endeca.eac.sslKeyStore=C:\\Endeca\\ToolsAndFrameworks\\11.1.0~
  \\deployment_template\\ssl_certs_utility\\bin\\ssl~
  \\slcw5dd.us.example.com.ks
#com.endeca.eac.sslKeyStorePassphrase=
# This must be a JKS trust store type
com.endeca.eac.sslTrustStore=C:\\Endeca\\ToolsAndFrameworks\\11.1.0~
 \\deployment_template\\ssl_certs_utility\\bin\\ssl~
 \\TS-slcw5dd.us.example.com.ks

#com.endeca.eac.sslTrustStorePassphrase=eacpass

# If using the OCS, specify values for the following parameters:
com.endeca.eac.ocs.jpsConfigPath=C:\\Endeca\\ToolsAndFrameworks~
  \\11.1.0\\server\\workspace\\credential_store\\jps-config.xml
com.endeca.eac.ocs.mapName=oracleCommerceSSLPassPhrase
com.endeca.eac.sslKeyStorePassKey=ks-key
com.endeca.eac.sslTrustStorePassKey=ts-key
```

5. In `%ENDECA_CONF%/conf/server.xml`, make sure that any non-SSL connector is commented and uncomment the following SSL connector and specify your key store and trust store files and passphrase:

```
 <!-- Define a SSL HTTP/1.1 Connector on port 8443 -->
   <Connector port="8443" maxHttpHeaderSize="8192" SSLEnabled="true"
   maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
   enableLookups="false" disableUploadTimeout="true"
   acceptCount="100" scheme="https" secure="true"
   clientAuth="true" sslProtocol="TLS"
   keystoreFile="C:/Endeca/ToolsAndFrameworks/11.1.0/deployment_template~
     /ssl_certs_utility/bin/ssl/slcw5dd.us.example.com.ks"
   keystorePass="eacpass"
   truststoreFile="C:/Endeca/ToolsAndFrameworks/11.1.0/deployment_template~
     b/ssl_certs_utility/in/ssl/TS-slcw5dd.us.example.com.ks"
   truststorePass="eacpass"
   URIEncoding="UTF-8"/>
```

Editing eaccmd to enable SSL communication with the EAC server

To enable SSL communication with the EAC server, follow these steps:

1. Edit `eaccmd.bat` or `eaccmd.sh` to configure SSL communication with the EAC server.

   (a).

```
IF EXIST %ENDECA_CONF%\conf\truststore.ks ( SET
    TRUSTSTORE=%ENDECA_CONF%\conf\truststore.ks
    ) ELSE (
    SET TRUSTSTORE=%EAC_ROOT%\..\workspace\conf\truststore.ks
    )
    IF EXIST %ENDECA_CONF%\conf\keystore.ks ( SET
    KEYSTORE=%ENDECA_CONF%\conf\keystore.ks
    ) ELSE (
    SET KEYSTORE=%EAC_ROOT%\..\workspace\conf\keystore.ks
    )
    ……
    SET JVM_ARGS=%JVM_ARGS% -Djavax.net.ssl.trustStore=%TRUSTSTORE%
    -Djavax.net.ssl.trustStoreType=JKS
    -Djavax.net.ssl.trustStorePassword=eacpass
    SET JVM_ARGS=%JVM_ARGS% -Djavax.net.ssl.keyStore=%KEYSTORE%
     -Djavax.net.ssl.keyStoreType=JKS
    -Djavax.net.ssl.keyStorePassword=eacpass
```

To make use of the credential store, pass keystore and truststore passphrase; thus:

```
 SET JVM_ARGS=%JVM_ARGS% -Djavax.net.ssl.trustStore=%TRUSTSTORE%
    -Djavax.net.ssl.trustStoreType=JKS
    SET JVM_ARGS=%JVM_ARGS% -Djavax.net.ssl.keyStore=%KEYSTORE%
    -Djavax.net.ssl.keyStoreType=JKS
```

(b). To make use of credential store, edit `eaccmd.bat` and `eaccmd.sh` to pass credentials to the OCS:

eaccmd.bat

```
 rem If using a credential store mention the below parameter values
    and rem avoid giving passphrase as java args.
    # jps-config.xml location
    set JPSCONFIGPATH=
    # mapName used to store ssl passphrases
    set MAPNAME=
    # key used to store truststore passphrase
    set TRUSTSTOREKEY=
    # key used to store keystore passphrase
    set KEYSTOREKEY=
```

eaccmd.sh

```
# Specify the credential store details to avoid passphrase in plaintext
    # JPSCONFIGPATH=
    # MAPNAME=
    # TRUSTSTOREKEY=
    # KEYSTOREKEY=
```

2. Run the `eaccmd` utility using the following syntax:

```
eaccmd fully-qualified-hostname:SSL-port command --force-ssl
```

For example:

```
eaccmd slcw5dd.us.example.com:8443 list-apps --force-ssl
```

For information about the syntax of the `eaccmd` utility, refer to the *Oracle Commerce Guided Search Platform Services Application Controller Guide*.

Note: You can also specify the fully qualified hostname and SSL port (8443) in the `%ENDE¬CA_CONF%/conf/eaccmd.properties` file, rather than in the `eaccmd` utility command line; for example:

```
host=slcw5dd.us.example.com
port=8443
```

Chapter 12

Enabling SSL with CAS

The section describes how to enable two-way SSL communication between CAS and the Deployment Template, Forge, and the ECR.

Steps for Creating and Storing SSL Credentials for CAS

You must follow the procedures described in this section to enable SSL communication between CAS and the following components:

Enabling Two-Way SSL Communication for CAS

Note: In several lines of the following sample code, line breaks (denoted by the symbol ~) are inserted to make the lines fit the width of page. Do not insert line breaks into lines of actual working code.

1. Set the property `useSsl` in the file `%DISCOVER_DATA_CAS_APP%\config\cas\last-mile-crawl.xml` to **true**. The default is **false**. For example:

```
<moduleProperty>
   <key>useSsl</key>
   <value>true</value>
</moduleProperty>
```

2. Set the `sslEnabled` property in the `<custom-component id="CAS" . . . >` element in the `%ENDE¬CA_APP%\config\script\DataIngest.xml` file to **true**, as follows:

```
<custom-component id="CAS" host-id="ITLHost"
   class="com.endeca.eac.toolkit.component.cas.ContentAcquisitionServerCompo¬
nent">
  <properties>
    .........
    <property name="sslEnabled" value="true" />
    .........
  </properties>
    .........
</custom-component>
```

3. Update the hostname in `initialize_services.bat` to specify a fully qualified name (for example, `slcw5dd.us.example.com`) and port.
4. Update the hostname in the `load_baseline_test_data.bat` file, under `%DISCOVER_DA¬TA_CAS_APP%\control\`.

5. Add the following settings to `index_config_cmd.bat`:

```
SET JAVA_ARGS=%JAVA_ARGS%
-Djavax.net.ssl.trustStore=C:/Endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
plate/ssl_certs_utility
   /bin/ssl/TS-slcw5dd.us.example.com.ks
-Djavax.net.ssl.trustStoreType=JKS
-Djavax.net.ssl.trustStorePassword=eacpass
SET JAVA_ARGS=%JAVA_ARGS%
-Djavax.net.ssl.keyStore=C:/Endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
plate/ssl_certs_utility
   /bin/ssl/slcw5dd.us.example.com.ks
-Djavax.net.ssl.keyStoreType=JKS
-Djavax.net.ssl.keyStorePassword=eacpass
```

Note: The two following steps alone are sufficient to enable SSL communication for the CAS Server.

6. Specify passphrases, and keystore and truststore file configuration, in the `jetty.xml` file, as in the sample below.

Note: Be sure to place copies of your Truststore and Keystore inside CAS_ROOT. The `jetty.xml` file reads paths relative to CAS_ROOT only, and cannot read absolute paths.

```
 <Call class="java.lang.System" name="setProperty">
  <Arg>com.endeca.cas.port</Arg>
  <Arg><SystemProperty name="com.endeca.cas.port" default="8500"/></Arg>
 </Call>
 <Call class="java.lang.System" name="setProperty">
  <Arg>com.endeca.cas.ssl.port</Arg>
  <Arg><SystemProperty name="com.endeca.cas.ssl.port" default="8505"/></Arg>
 </Call>
 <Call class="java.lang.System" name="setProperty">
  <Arg>com.endeca.cas.fullyQualifiedHostName</Arg>
  <Arg><NIRAD-LAP1</Arg>
 </Call>
 <Call class="java.lang.System" name="setProperty">
  <Arg>javax.net.ssl.trustStore</Arg>
  <Arg><SystemProperty name="jetty.home" default="." />/../workspace/conf/TS-
NIRAD-LAP1.ks</Arg>
 </Call>
 <Call class="java.lang.System" name="setProperty">
  <Arg>javax.net.ssl.trustStorePassword</Arg>
  <Arg>eacpass</Arg>
 </Call>
 <Call class="java.lang.System" name="setProperty">
  <Arg>javax.net.ssl.trustStoreType</Arg>
  <Arg>JKS</Arg>
 </Call>
 <Call class="java.lang.System" name="setProperty">
  <Arg>javax.net.ssl.keyStore</Arg>
  <Arg><SystemProperty name="jetty.home" default="." />/../workspace/conf/KKO¬
RIVI-LAP1.ks</Arg>
 </Call>
 <Call class="java.lang.System" name="setProperty">
  <Arg>javax.net.ssl.keyStorePassword</Arg>
  <Arg>eacpass</Arg>
 </Call>
 <Call class="java.lang.System" name="setProperty">
  <Arg>javax.net.ssl.keyStoreType</Arg>
  <Arg>JKS</Arg>
</Call>
```

7. Uncomment the following section in `jetty.xml` to add a secure connector.

```
<!-- Uncomment this section to add a secure connector -->
<Call name="addConnector">
  <Arg>
    <New class="org.mortbay.jetty.security.SslSocketConnector">
       <Set name="Port"><SystemProperty name="com.endeca.cas.ssl.port"/></Set>

        <Set name="maxIdleTime">600000</Set>
      <Set name="keystore"><SystemProperty name="javax.net.ssl.keyStore"/></Set>

      <Set name="keyPassword"><SystemProperty name="javax.net.ssl.keyStorePass¬
word"/></Set>
        <Set name="truststore"><SystemProperty name="javax.net.ssl.trust¬
Store"/></Set>
        <Set name="trustPassword"><SystemProperty name="javax.net.ssl.trust¬
StorePassword"/></Set>
        <Set name="needClientAuth">true</Set>
    </New>
  </Arg>
</Call>
```

Steps to store SSL credentials for CAS in the OCS

To store SSL credentials for CAS in the OCS, follow these steps:

1. To enable SSL communication between CAS and the ECR, provide values for the properties `Jpsconfig¬Path` , `ocsMapName` , and `OcsKeyName` in `%DISCOVER_DATA_CAS_APP%\config\cas\last-mile-crawl.xml`; for example:

```
<outputConfig>
  <moduleId>
    <id>com.endeca.cas.output.Mdex</id>
  </moduleId>
  <moduleProperties>
     .......
     <moduleProperty>
       <key>JpsConfigPath</key>
       <value><path-to-jps-config>/jps-config.xml</value>
     </moduleProperty>
     <moduleProperty>
       <key>OcsMapName</key>
       <value>endecaCAS</value>
     </moduleProperty>
     <moduleProperty>
       <key>OcsKeyName</key>
       <value>cas-ecr-communication</value>
     </moduleProperty>
   </moduleProperties>
</outputConfig>
```

2. To store user-authentication credentials for a website in the OCS, specify values for `JpsConfigPath` , `OcsMapName` , and `OcsKeyName` in the `%CAS_ROOT%\workspace\conf\web-crawler\default\de¬fault.xml` file; for example:

```
<property>
   <name>JpsConfigPath</name>
   <value>path-to-jps-config\jps-config.xml</value>
```

```
</property>
<property>
   <name>OcsMapName</name>
   <value>endecaCAS</value>
</property>
<property>
   <name>OcsKeyName</name>
   <value>basic-crawl</value>
 </property>
```

3. To store form-based authentication credentials for a webcrawler in the OCS, provide values for the following properties in the `%CAS_ROOT%\..\workspace\conf\web-crawler\default\form-creden¬tials.xml` file: `jpsconfig` , `OcsMapName` , `OcsKeyName` , `UserFormParameterId` , and `Password¬FormParameterId` ; for example:

```
<parameters>
   <ocs-parameters>
     <parameter>
       <name>JpsConfigPath</name>
       <value>path-to-jps-config\jps-config.xml</value>
     </parameter>
     <parameter>
       <name>OcsMapName</name>
       <value>endecaCAS</value>
     </parameter>
     <parameter>
       <name>OcsKeyName</name>
       <value>WebCrawler</value>
     </parameter>
   </ocs-parameters>
   <form-parameters>
     <parameter>
       <name>UserFormParameterId</name>
       <value>login</value>
     </parameter>
     <parameter>
       <name>PasswordFormParameterId</name>
       <value>password</value>
     </parameter>
   </form-parameters>
</parameters>
```

4. To enable server authentication mechanisms for crawls, add the following properties to `%CAS_ROOT%\..\workspace\conf\web-crawler\default\default.xml` :

   a. Update the property `http.auth.basic` with basic authentication configuration required for OCS:

```
<property>
   <name>http.auth.basic</name>
   <value>USE_OCS~~~[JPS_CON¬
FIG_PATH]~~~[MAP_NAME]~~~[KEY_NAME]~~~ANY_HOST~~~ANY_PORT~~~ANY_REALM|||
          USE_OCS~~~[JPS_CON¬
FIG_PATH]~~~[MAP_NAME]~~~[KEY_NAME1]~~ANY_HOST~~~ANY_PORT~~~ANY_REALM|||
          ...
   </value>
   <description>...</description>
<property>
```

   b. Update the property `http.auth.digest` to have the digest configuration required for OCS:

```
<property>
   <name>http.auth.digest</name>
```

```
   <value>USE_OCS~~~[JPS_CON¬
FIG_PATH]~~~[MAP_NAME]~~~[KEY_NAME]~~~ANY_HOST~~~ANY_PORT~~~ANY_REALM|||
          USE_OCS~~~[JPS_CON¬
FIG_PATH]~~~[MAP_NAME]~~~[KEY_NAME1]~~ANY_HOST~~~ANY_PORT~~~ANY_REALM|||
           ...
   </value>
   <description>...</description>
</property>
```

c. Update the property `http.auth.ntlm` to have the NTLM Authentication configuration required for OCS.

```
<property>
   <name>http.auth.ntlm</name>
   <value>USE_OCS~~~[JPS_CON¬
FIG_PATH]~~~[MAP_NAME]~~~[KEY_NAME]~~~ANY_HOST~~~ANY_PORT~~~ANY_REALM~~~DO¬
MAIN1|||
          USE_OCS~~~[JPS_CON¬
FIG_PATH]~~~[MAP_NAME]~~~[KEY_NAME1]~~~ANY_HOST~~~ANY_PORT~~~ANY_REALM~~DO¬
MAIN2|||
          ...
   </value>
   <description>...</description>
</property>
```

Setting commandline properties for SSL communication with CAS

1. Open `%ENDECA_CAS_ROOT%/../workspace/conf/commandline.properties` for editing.
2. Specify a fully qualified hostname, an SSL port (must be 8505), and an SSL flag value (must be true):

```
com.endeca.itl.cas.server.host=busgg2007.us.example.com
com.endeca.itl.cas.server.port=8505
# If set to true, the CAS Server port is interpreted as
# the SSL port. If using redirects from the non-SSL port
# to the SSL port this should be false.
com.endeca.itl.cas.server.isPortSsl=true
```

Update utilities for SSL communication with CAS

To update utilities for SSL communication with CAS, follow these steps:

1. Open the following files in `%ENDECA_CAS_ROOT%/bin/` for editing:
   - cas-cmd.[bat|sh]
   - component-manager-cmd.[bat|sh]
   - recordstore-cmd.[bat|sh]
   - web-crawler.[bat|sh]
2. In each of the files listed in the preceding step, uncomment lines in the following code to enable either plain text passphrases or credential store OCS passphrases:

```
 REM Set the JVM default trust store and key store locations.
 SET JVM_ARGS=-Djavax.net.ssl.trustStore="C:\Endeca\ToolsAndFrameworks\11.1.0\de¬
ployment_template~
  \ssl_certs_utility\bin\ssl\TS-slcw5dd.us.oracle.com.ks" %JVM_ARGS%
 SET JVM_ARGS=-Djavax.net.ssl.trustStoreType=JKS %JVM_ARGS%
 REM SET JVM_ARGS=-Djavax.net.ssl.trustStorePassword=eacpass %JVM_ARGS%
 SET JVM_ARGS=-Djavax.net.ssl.keyStore="C:\Endeca\ToolsAndFrameworks\11.1.0\de¬
ployment_template~
```

```
  \ssl_certs_utility\bin\ssl\slcw5dd.us.example.com.ks" %JVM_ARGS%
 SET JVM_ARGS=-Djavax.net.ssl.keyStoreType=JKS %JVM_ARGS%
 REM SET JVM_ARGS=-Djavax.net.ssl.keyStorePassword=eacpass %JVM_ARGS%
 SET JVM_ARGS=-Dcom.endeca.itl.credstore.keyStoreKeyName=keystorekey %JVM_ARGS%

 SET JVM_ARGS=-Dcom.endeca.itl.credstore.trustStoreKeyName=truststorekey
%JVM_ARGS%
 SET JVM_ARGS=-Doracle.security.jps.config=C:\Endeca\ToolsAndFrameworks\11.1.0~

  \credential_store\conf\jps-config.xml %JVM_ARGS%
 SET JVM_ARGS=-Dcom.endeca.itl.credstore.mapName=oracleCommerceSSLPassPhrase
%JVM_ARGS%
```

Chapter 13

Enabling SSL with the Reference Application

You must modify the `assembler.properties` file to enable SSL communication for the Discover Electronics reference application.

Steps for Enabling SSL Communication with the Reference Application

To enable SSL communication with the Discover Electronics reference application, follow these steps:

1. Open the `assembler.properties` file at one of the following locations, depending on whether you want only to run the reference application, or whether you want to edit it (authoring):

   ```
   %ENDECA_TOOLS_ROOT%/reference/discover-electronics/WEB-INF/assembler.properties
    (run only)
   %ENDECA_TOOLS_ROOT%/reference/discover-electronics-authoring/WEB-INF/assem¬
   bler.properties (edit)
   ```

2. Set the `workbench.port` property to 8446:

   ```
   workbench.port=8446
   ```

3. Set the `mdex.sslEnabled` and `logserver.sslEnabled` properties to true:

   ```
     mdex.sslEnabled=true
     logserver.sslEnabled=true
   ```

Chapter 14

Enabling SSL for Export and Import of User Scripts

To enable the export and import of user properties, you must modify the `export_user.properties` and `import_user.properties` files, as described in this section.

Steps for Enabling SSL Communication for Export of User Scripts

To enable SSL for the export of user scripts, follow these steps:

1. Open `%ENDECA_TOOLS_ROOT%/admin/conf/export_users.properties` for editing.
2. Specify a fully qualified hostname, SSL port number (must be 8446), and SSL flag (set to true); for example:

```
source.workbench.host=slcw5dd.us.example.com
source.workbench.port=8446
source.workbench.sslEnabled=true
```

3. Uncomment the following snippet to enable either plain text passphrases or credential store [OCS] passphrases. Note: Plain text passphrases have higher priority than credential store [OCS] passphrases.

```
  javax.net.ssl.trustStore=C:/Endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
plate/ssl_certs_utility/bin/ssl
    /TS-slcw5dd.us.example.com.ks
  javax.net.ssl.trustStoreType=JKS
  #javax.net.ssl.trustStorePassword=eacpass

  javax.net.ssl.keyStore=C:/Endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
plate/ssl_certs_utility/bin/ssl
    /slcw5dd.us.example.com.ks

  javax.net.ssl.keyStoreType=JKS
  #javax.net.ssl.keyStorePassword=eacpass

  com.endeca.ssl.jpsConfigPath=C:/Endeca/ToolsAndFrameworks/11.1.0/serv¬
er/workspace/credential_store/jps-config.xml
  com.endeca.ssl.storeMapName=oracleCommerceSSLPassPhrase
  com.endeca.ssl.trustStoreKeyName=ts-key
  com.endeca.ssl.keyStoreKeyName=ks-key
```

Steps for Enabling SSL Communication for Import of User Scripts

To enable SSL communication for the import of user scripts, follow these steps:

1. Open the `%ENDECA_TOOLS_ROOT%/admin/conf/import_users.properties` file for editing.
2. Specify a fully qualified hostname, an SSL port number (must be 8446), and an SLL flag value (must be true):

   ```
   dest.workbench.host=slcw5dd.us.example.com
   dest.workbench.port=8446
   dest.workbench.sslEnabled=true
   ```

3. Uncomment following lines either to enable plain text passphrases or credential store [OCS] passphrases.

```
  javax.net.ssl.trustStore=C:/Endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
plate/ssl_certs_utility/bin/ssl/TS-slcw5dd.us.example.com.ks
  javax.net.ssl.trustStoreType=JKS
  #javax.net.ssl.trustStorePassword=eacpass
  javax.net.ssl.keyStore=C:/Endeca/ToolsAndFrameworks/11.1.0/deployment_tem¬
plate/ssl_certs_utility/bin/ssl/slcw5dd.us.example.com.ks
  javax.net.ssl.keyStoreType=JKS
  #javax.net.ssl.keyStorePassword=eacpass
  com.endeca.ssl.jpsConfigPath=C:/Endeca/ToolsAndFrameworks/11.1.0/serv¬
er/workspace/credential_store/jps-config.xml
  com.endeca.ssl.storeMapName=oracleCommerceSSLPassPhrase
  com.endeca.ssl.trustStoreKeyName=ts-key
  com.endeca.ssl.keyStoreKeyName=ks-key
```

Appendix A

Default Encryption Algorithms

Guided Search components by default use an encryption algorithm chosen from a list of approved algorithms for secure communication with other Guided Search components.

Default Algorithms

The following list contains the currently approved algorithms. The order in which the items appear in the list is not indicative of any priority among the items.

- ADH-AES128-SHA
- ADH-AES256-SHA
- AES128-SHA
- AES256-SHA
- DHE-DSS-AES128-SHA
- DHE-DSS-AES256-SHA
- DHE-RSA-AES128-SHA
- DHE-RSA-AES256-SHA

Index