Oracle Commerce Guided Search

Administrator's Guide

Version 11.1 • July 2014

ORACLE

COMMERCE

Contents

Copyright and disclaimer

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.

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

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 tasks involved in administering and maintaining applications built with Oracle Commerce Guided Search. It bridges the gap between the work performed when your Guided Search implementation is initially deployed, and the issues that you, as a system administrator, may need to address to maintain the system.

This guide assumes that the Guided Search software is already installed on a development server. It may be already installed in a production environment. It also assumes that you, or your Oracle Services representatives, have already used the Deployment Template to configure the application on the development server.

License restrictions for Rule Manager

You can use Rule Manager to create business rules to control content spotlighting. Using the record spotlight cartridge to spotlight records and spotlighting content such as promotional images is supported.

You cannot use Rule Manager to provide business user control over contextual, rules-driven dimensions or dimension values (for example, functionality similar to the guided navigation cartridges). You cannot provide contextual business user control over results lists (for example. the search results cartridge). In short, if you use Rule Manager, guided navigation and results lists must be configured globally or through APIs. Business user tooling for rules-driven control of either of these aspects requires an Experience Manager license.

Who should use this guide

This guide is intended for system administrators who maintain an Guided Search implementation.

You should also be familiar with the Guided Search concepts described in the *Oracle Commerce Guided Search Concepts Guide*.

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*.

Part 1

Creating an Application Environment

Chapter 1

About Assembler Application Components

This section describes the components of an Oracle Commerce Guided Search implementation, including the main administrative tools — the Deployment Template and Oracle Endeca Application Controller (EAC).

About the components of an Oracle Commerce Guided Search application

An Oracle Commerce Guided Search application typically consists of the Tools and Frameworks, MDEX Engine, and Platform Services packages. Optionally, it may use the Content Acquisition System to ingest and manipulate data prior to sending it to the Engine for indexing.

Tools and Frameworks

The Tools and Frameworks package includes the Assembler, a language-independent server-side framework that makes up the core of an Assembler application. While most Oracle Commerce Guided Search implementations use the Assembler to communicate with the MDEX Engine, an Assembler application can query any CMS, database, or repository.

MDEX Engine

The MDEX Engine is the query engine that backs most Assembler applications. It consists of two main parts:

- Dgidx, the program that indexes the records output either by Forge (a Platform Services component) or the Content Acquisition System.
- Dgraph, the main MDEX Engine process that executes queries against indexed data

Platform Services

The Platform Services package includes the Oracle Endeca Application Controller, or EAC, a control system you can use to control, manage, and monitor components in your Guided Search implementation. It also includes the Data Foundry, which is used to ingest data and transform it into Endeca records.

Content Acquisition System

The Content Acquisition System is a set of components that add, configure, and crawl data sources for use in a Guided Search application and transform the information into Endeca records.

For additional information on these components, see the *Getting Started Guide* or refer to the documentation for a specific component.

About the Oracle Endeca Application Controller

The Oracle Endeca Application Controller (EAC) is a control system you can use to control, manage, and monitor components in your Endeca implementation. It is installed with the Platform Services package. The Endeca HTTP service, which controls the EAC, is created, registered, and configured when installing Platform Services, and starts when you reboot the machine after installation.

The EAC must run on each server that hosts Guided Search components. One instance acts as the EAC Central Server (selected during Platform Services installation), while the other instances act as EAC Agents and receive configuration information and commands from the central server.

Note: You can enable SSL communication between the EAC Central Server and EAC agents. For details, see the *Oracle Commerce Guided Search Security Guide*.

The EAC is platform and language independent, and uses Web Services Descriptive Language (WSDL) to communicate both internally (between the central server and agents) and with external resources. It enables management of operations such as partial updates, delta updates, phased MDEX Engine updates, and more. The EAC Central Server stores and propagates configuration for the following components across all EAC Agents:

- **Forge** — Data ingest software that transforms source data into tagged Endeca records.
- **Dgidx** — An MDEX Engine component that generates indices from Endeca records and sends them to all Dgraphs.
- **Dgraphs** — The MDEX Engine software that processes queries against indexed records.
- **Log Server** — The MDEX Engine log server.
- **Report Generator** — The MDEX Engine report generator that generates reports from log files.

EAC architecture

The EAC is installed on each machine that runs the Guided Search software and is typically run in a distributed environment.

Each instance of the EAC takes one of two roles:

- EAC Central Server (one per Oracle Commerce Guided Search deployment)
- EAC Agent

The EAC is accessible through three methods:

- **The Deployment Template** communicates with the EAC programatically, and is the system of record for storing EAC configuration on disk and for defining or updating application provisioning.
- **Workbench** communicates with the EAC Central Server through the WSDL interface. You can use the **EAC Admin Console** in Workbench to monitor, start, and stop application components.
- The command line utility, **`eaccmd`** , lets you script the EAC using Perl, shell, or batch scripts. This utility is recommended for debugging use only.

EAC Central Server

One instance of the EAC serves as the EAC Central Server for your implementation. This instance includes a WSDL-enabled interface, through which Workbench and the Deployment Template communicate with the EAC.

The EAC Central Server also contains a repository that stores provisioning information — that is, data about the hosts, components, applications and scripts that the EAC manages.

Note: Generally, you should configure only one EAC Central Server to manage all of your Guided Search applications. Provisioning an application with more than one Central Server leads to conflicting instructions being issued across EAC Agents.

EAC Agents

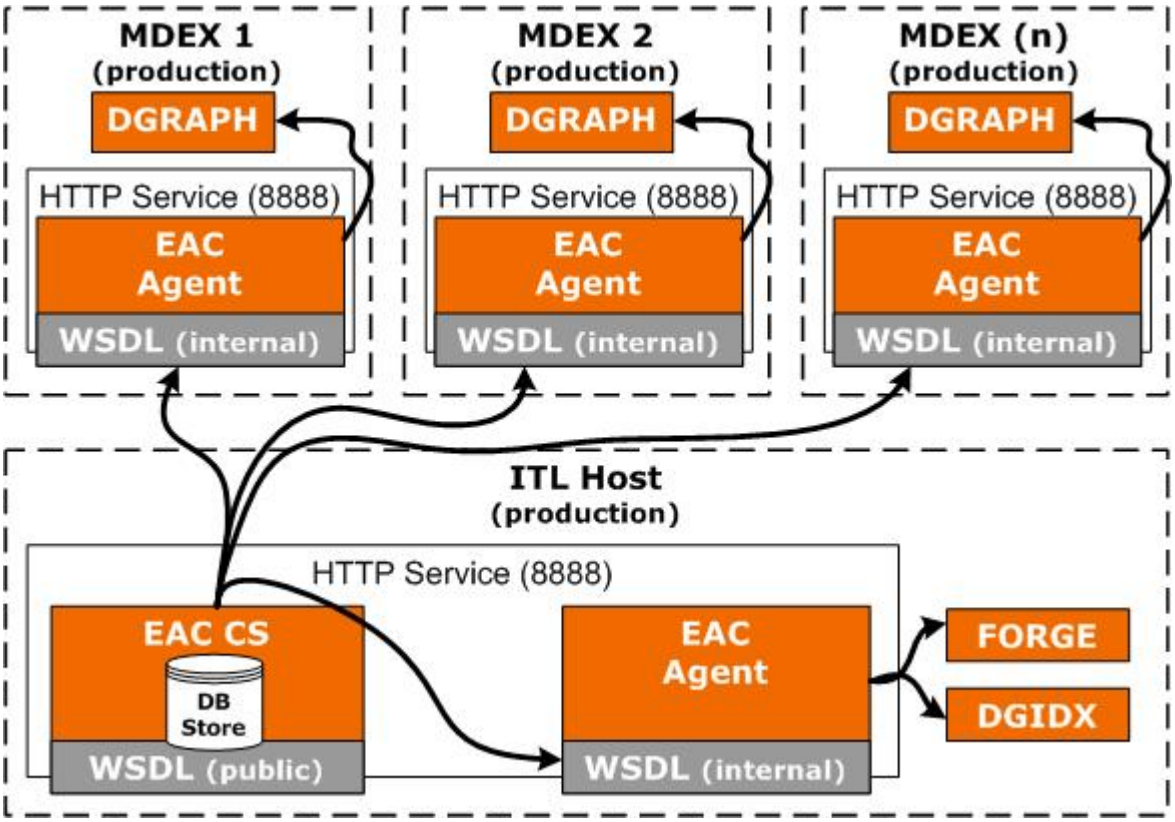
All other instances of the EAC serve as Agents. The EAC Central Server communicates with Agents through an internal Web service interface: all command, control, and monitoring functions are dispatched from the Central Server. The Agents, in turn, instruct their host machines to do the actual work of a Guided Search implementation, such as processing data with a Forge component or indexing it using Dgidx.

EAC architecture example

A typical Guided Search implementation is usually spread across multiple host servers. Each of these physical servers must have an EAC Agent that controls the components installed on the server.

The following diagram shows the architecture of the EAC.

The EAC Central Server communicates with EAC Agents that run on each machine hosting an entire implementation (or components that comprise an implementation). The EAC Server communicates to the Agents the information about the instance configuration and resource configuration. The Agents run the necessary components and their processes on each machine, such as Forge, Dgidx, and Dgraph.



About the Deployment Template

The Deployment Template provides a collection of operational components that serve as a starting point for application provisioning. It is installed with the Tools and Frameworks package.

The Deployment Template creates the complete directory structure required for provisioning and managing an application with the EAC. Along with creating this file structure, it integrates deployed applications with tooling in Workbench and creates configuration files and control scripts that wrap common EAC functionality.

You can use the provided scripts as-is, or create custom scripts that wrap additional functionality specific to your environment.

About the AppConfig.xml file

The `<app directory>\config\script\AppConfig.xml` file in a deployed application serves as that application's central configuration file. It defines the Guided Search components and host machines that make up the application in the EAC. It also includes BeanShell scripts that perform various operations.

About deploying an application with the Deployment Template

This section outlines the process of deploying an application using the Deployment Template.

The Deployment Template takes the following inputs:

- A `deploy.xml` application descriptor file, passed in at the command line with the `--app` flag. This defines the source-to-destination file mappings that create a correctly structured DT application.
- The application name and destination directory on disk
- The EAC port
- The Workbench port
- The Dgraph ports for each Dgraph defined in the `AppConfig.xml` file
- The Log Server port

1. The Deployment Template creates the directory structure, configuration files, and control scripts for the application on disk. At this point, the application is not yet provisioned with the EAC or registered with Workbench.
2. The application administrator runs the `<app.dir>\control\initialize_services` script to provision the application with the EAC and to upload application content to the Endeca Configuration Repository.
3. The application administrator runs the `<app directory>\control\load_baseline_test_data` script to load data in preparation for starting the MDEX Engine.
4. The application administrator runs the `<app.dir>\control\baseline_update` script to run the data ingest process and initialize the MDEX Engine. At this point, one or more Dgraph processes are running and able to respond to queries.

Files in an Assembler application

The configuration files, scripts, and state files that make up a typical Guided Search application are listed below.

Application configuration files

The `AppConfig.xml` file and its associated files describe an application's provisioning information. The Deployment Template creates the file in the `<app dir>\config\script` directory.

Although configuration can be edited in Workbench, Oracle recommends modifying the `AppConfig.xml` file directly. By storing the current configuration on disk, it is easier to copy it between environments. You can use the Workbench to review the configuration, and to start or stop individual components.

Some parts of the `AppConfig.xml` file include settings that are environment specific, such as the application's name, file system paths, and host addresses in the environment. These settings should be collected and stored in a custom file for each machine. For more information about how to create this file, see *Creating a custom file for environment-specific settings* on page 22.

Instance configuration files

Instance configuration XML files include Forge and Dgraph configuration and Experience Manager templates and content. These files are maintained in the Endeca Configuration Repository in Workbench. You can upload or retrieve changes to these files by using the scripts in the `<app dir>\control` directory of a deployed application.

Command-line scripts

The Deployment Template creates batch or shell scripts in the `<app dir>\control` of a deployed application. These scripts allow you to run baseline or partial updates on the MDEX Engine, or to retrieve or update configuration in the Endeca Configuration Repository. You can create additional scripts if necessary -- for example, if you wish to crawl a Web page prior to running the `baseline_update` script.

Command-line scripts, along with their input data and output directories, should be shared among development, staging and production environments.

Java libraries

By convention, Java libraries are located under `<app dir>\config\lib`.

Forge state files

Forge state files are located under `<app dir>\data\state`.

In most cases, these files *do not* need to be included as part of an application definition. However, when an application uses dimension values from auto-generated or external dimensions, then Forge state files *do* need to be synchronized across environments. In this situation, the state files contain the IDs of these dimension values and ensure that the same dimension value always gets the same ID no matter how many times Forge is run. These dimension values may be used in a variety of ways, including dynamic business rules, landing pages, dimension ordering, and precedence rules.

In other words, Forge state files should be identified as part of the application definition if the application uses auto-generated dimensions or external dimensions, and values from these dimensions are referenced anywhere in the application configuration (for example, in landing pages or content items, explicit dimension ordering, or in precedence rules).

Chapter 2

Designing an Application

This section covers design considerations that you should consider when designing an application to ensure that it is portable across different server environments.

Controlling paths to ensure interoperability across environments

The development, staging, and production environments often include servers with different operating systems and directory structures. Using forward slashes, relative path names and variables can ensure that application definitions propagated across environments work in the target environment.

Use forward slashes

To ensure that paths work in all locations, use forward slashes ("/") wherever possible in configuration files and scripts. Windows and Unix environments both work well with forward slashes. The only exceptions are platform-specific scripts, such as Windows .bat files or UNIX shell scripts.

Use relative paths in the pipeline

Using absolute paths in the Forge pipeline ties the application to a particular location in one environment. It is therefore better to use relative paths.

In record-adapters, paths are relative to `<app dir>/data/processing`, which contains the content of `<app dir>/data/incoming` before the pipeline is run. Therefore any files from `<app dir>/data/incoming` can be referenced directly by name, without specifying a directory.

Java manipulator paths are relative to `<app dir>`. This applies both to the manipulator's class path and to any files the Java code may try to access. Since these file names are often given as pass-throughs, keep such pass-through values relative to `<app dir>`.

Use variables in scripts

You can specify relative paths in scripts, but keep in mind that scripts are not always invoked from the same directory. For example, there might be a script called `crawl.sh` in `<app dir>/control`. If this script uses paths relative to `<app dir>/control`, then it will only work correctly if it is invoked while stored in this directory. But if the script is invoked as `/control/crawl.sh` from `<app dir>`, then the paths will be incorrect and the script may fail.

To avoid this problem, scripts can use preset variables to refer to a known directory without presuming any particular absolute path. In Windows batch files, the variable `%~dp0` resolves to the directory containing the script. In UNIX Bash scripts, the variable `$0` resolves to the script name, and running the `dirname` command

on it will give the script's directory. In BeanShell scripts for AppConfig.xml, the variable ${ENDECA_PROJECT_DIR} points to <app dir>. By using these variables, you can construct portable paths in location-independent scripts.

All the scripts generated by the Deployment Template use this technique and can serve as examples.

For UNIX, the scripts generally start with the following lines:

```
WORKING_DIR=`dirname ${0} 2>/dev/null`
. "${WORKING_DIR}/../config/script/set_environment.sh"
```

This sets the WORKING_DIR variable to the directory containing the script being run, and then addresses the set_environment.sh file in a different directory by making the path relative to WORKING_DIR. Wherever the script is invoked, the path to set_environment.sh will always resolve correctly.

For Windows, the scripts usually start with the following line:

```
call %~dp0..\config\script\set_environment.bat
```

This addresses set_environment.bat via a path relative to%~dp0. It ensures that the reference is correct wherever the script is invoked.

Creating a custom file for environment-specific settings

Most application configuration settings can be shared among all environments, but a few are environment-specific. These settings should be removed from the AppConfig.xml file and stored in a separate file. That way, each environment will have its own custom file of environment-specific settings that is not changed during synchronization.

Environment-specific settings typically include an application's name, file system paths, host addresses in the environment, and the definition of MDEX processes known as the Dgraph cluster.

To create a custom.xml file:

1. Edit the AppConfig.xml file generated by the Deployment Template to include a line similar to the following :
   ```
   <spr:import resource="custom.xml" />
   ```
2. Move all environment-specific elements from AppConfig.xml to a new custom.xml file. These elements might include the <app>, <host>, <dgraph-cluster>, <dgraph>, and <logserver> elements.
3. Create a custom.xml file for each of the environments, with settings appropriate for that environment.

Here is a sample custom.xml file to use a general reference when creating your own file.

```
- <!--
     ############################################################################
     # Global variables

     -->
- <app appName="wine" eacHost="ConfigMig1" eacPort="8888" dataPrefix="wine"
sslEnabled="false" lockManager="LockManager">
  <working-dir>${ENDECA_PROJECT_DIR}</working-dir>
  <log-dir>./logs</log-dir>
  </app>
- <!--
     ############################################################################
     # Servers/hosts
     #
```

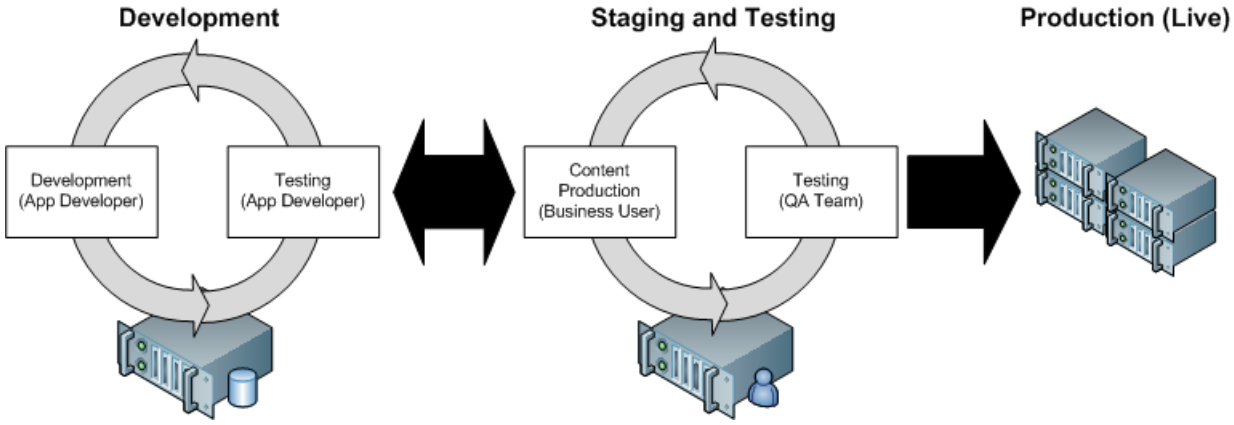
```
    # The "webstudio" host and its "webstudio-report-dir" directory use
    # predefined names to inform Workbench where it should look for reports
    # for this application.
    #
     -->
  <host id="ITLHost" hostName="ConfigMig1" port="8888" />
  <host id="MDEXHost" hostName="ConfigMig1" port="8888" />
- <host id="webstudio" hostName="ConfigMig1" port="8888">
- <directories>
  <directory name="webstudio-report-dir">./reports</directory>
  </directories>
  </host>
- <!--
    ###############################################################################
    # Dgraph Cluster
    #
     -->
- <dgraph-cluster id="DgraphCluster" getDataInParallel="true">
  <dgraph ref="Dgraph1" />
  </dgraph-cluster>
- <!--
    ###############################################################################
    # Dgraphs
    #
     -->
- <dgraph id="Dgraph1" host-id="MDEXHost" port="15000">
- <properties>
  <property name="restartGroup" value="A" />
  <property name="updateGroup" value="a" />
  </properties>
  <log-dir>./logs/dgraphs/Dgraph1</log-dir>
  <input-dir>./data/dgraphs/Dgraph1/dgraph_input</input-dir>
  <update-dir>./data/dgraphs/Dgraph1/dgraph_input/updates</update-dir>
  </dgraph>
- <!--

    ###############################################################################
    # LogServer
    #
     -->
- <logserver id="LogServer" host-id="ITLHost" port="15010">
- <properties>
  <property name="numLogBackups" value="10" />
  <property name="targetReportGenDir" value="./reports/input" />
  <property name="targetReportGenHostId" value="ITLHost" />
  </properties>
  <log-dir>./logs/logservers/LogServer</log-dir>
  <output-dir>./logs/logserver_output</output-dir>
  <startup-timeout>120</startup-timeout>
  <gzip>false</gzip>
  </logserver>
```

Overview of system architectures

This topic provides a general description of typical system architectures for each stage of a Guided Search implementation.

Guided Search implementations typically have three stages:

1. Development
2. Staging and testing
3. Production



This section does not provide specific system sizing requirements for a particular implementation. There are too many variables in each unique implementation to give general guidance. Some of these variables include hardware cost restrictions, data processing demands, application throughput demands, query load demands, scale requirements, failover availability, and so on. Oracle Professional Services can perform a hardware sizing analysis for your implementation.

Development environment

A development environment is one in which developers create or substantially modify a Guided Search implementation.

This implementation does not serve end-user queries. Because data processing and query processing demands are not very important at this stage, development typically occurs on a single machine. The single machine runs the Endeca Application Controller, Forge, Dgidx, a Web server, and the MDEX Engine.

Staging and testing environment

A staging environment is one that validates the correctness of the implementation including data processing and all necessary search and navigation features.

Features such as merchandising, thesaurus entries, and others may require business users to modify the implementation during this implementation phase. This environment is also typically used to test performance of the system. Once the implementation works as required, it is migrated to the production environment.

In terms of hardware architecture, most staging environments closely resemble or exactly match the intended production environment. This means the production environment typically determines the architecture of the staging environment.

Production environments

A production environment is a live Guided Search implementation that serves end-user search and navigation queries.

There are a variety of system architectures in a production environment. Most use at least two servers and one load balancer. As system demand increases, the number of servers necessary in the implementation increases. Demand may take the form of time to crawl source data, frequent source data updates, faster query throughput, faster response time under increasing load, and so on. Several of the most common implementation architectures are described in the following sections.

Implementation size

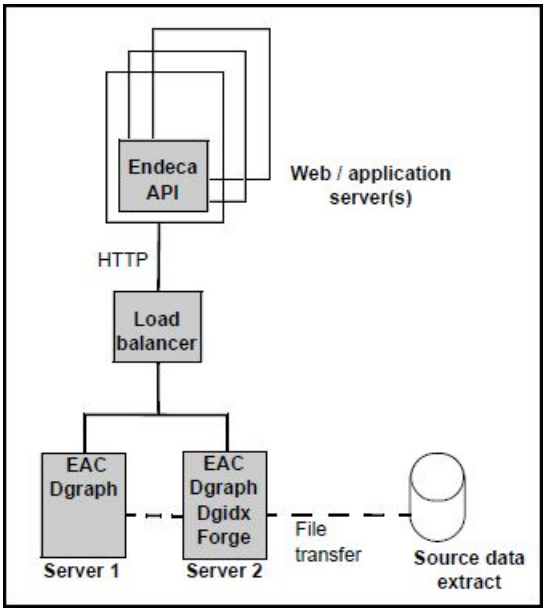
In the following examples, the terms "small" and "medium" are used as follows:

- A small implementation means the Dgraph runs an application's data set on a single processor.
- A medium implementation means a single Dgraph is mirrored several times for throughput (rather than solely for redundancy), and that a dedicated server may be necessary for crawling or indexing.

Small implementation with lower throughput

A simple architecture for smaller implementations is made up of two servers and a single load balancer.

Server 1 runs only the MDEX Engine. Server 2 runs a mirror of the MDEX Engine (for redundancy) and Forge and Dgidx. A single load balancer distributes queries between the MDEX Engines on servers 1 and 2.



The advantage of this scenario is low cost and MDEX Engine redundancy. If one MDEX Engine is offline for any reason, the load balancer distributes user queries to the other MDEX Engine.

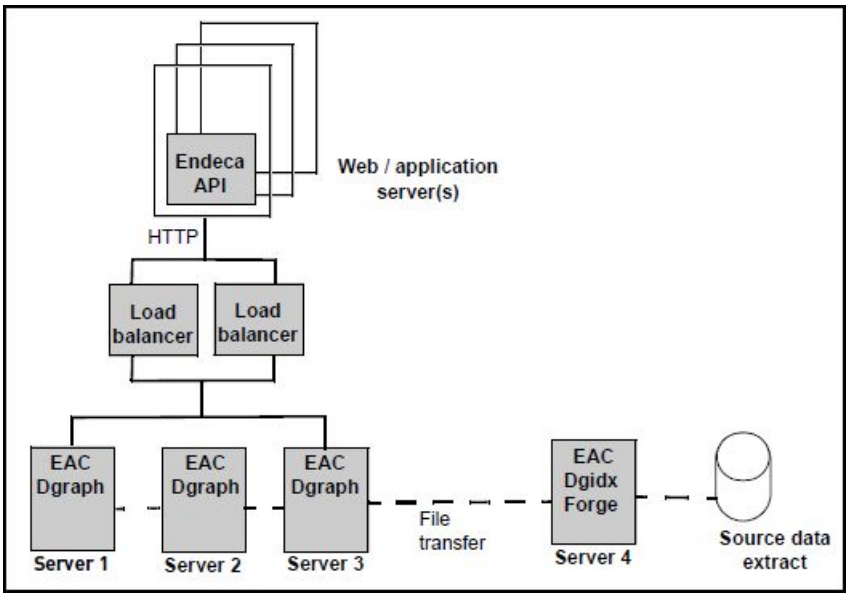
The disadvantage of this scenario is that the system operates at reduced throughput capacity during Forge and Dgidx processing, and during a server failure of either machine. Also, if the single load balancer fails, the system goes offline.

Medium implementation with higher throughput

In this example system architecture, a medium implementation that requires higher query throughput is made up of four servers and two load balancers.

To achieve higher throughput, servers 1, 2, and 3 all run mirror copies of the MDEX Engine. This level of redundancy provides faster throughput by load balancing the incoming queries over a greater number of MDEX

Engines. If either load balancer or any MDEX Engine should fail, then the redundant load balancer and remaining MDEX Engines handle all queries. Server 4 runs all the offline processes including Forge and Dgidix.



The advantage of this scenario is that overall throughput and redundancy is high. Each MDEX Engine runs on a dedicated server, so the servers do not need to share resources for Forge processing and indexing. Also, this scenario employs two load balancers to reduce potential offline time if one balancer fails.

The disadvantage of this scenario is that the implementation operates at reduced throughput if any MDEX Engine server fails. However, a single server failure has less effect on the implementation than the previous examples because the MDEX Engine has been replicated more times than in previous examples.

Planning and configuring server topology

Assess the requirements of your staging and production environments and adjust the `AppConfig.xml` file in the Deployment Template to point to the correct host names for your servers.

This topic provides high-level steps. For additional information on customizing the Deployment Template, see *Configuring an EAC Application* on page 115.

To configure your server topology:

1. Analyze your goals for the performance of your Guided Search application. Look at the projected size of the data set and other characteristics. For detailed information on hardware benchmarking and performance, see the *Oracle Commerce MDEX Engine Performance Tuning Guide*.
2. Formulate the requirements of your staging and production environments, based on your goals for the expected performance.

   For example, you will need to have an estimate of how many servers must be provisioned in the staging and production environments, and how many MDEX Engines you will need to run on each of the MDEX Engine servers. In the staging environment, you may want to provision a full duplicate of your expected production environment, or it may be sufficient to provision a subset of your production servers in a staging environment.

Note: The Workbench server requires a separate thread for each Assembler client. If you configure a large number of Assembler clients (50+) for your application or applications, you must ensure that the Workbench server has adequate memory. This may require more than the recommended 4 GB.

3. Proceed to configure your server topology for both staging and production environments. You do this by adjusting the Deployment Template `AppConfig.xml` file so that it points to the correct host names for your servers.

   You will typically run the Deployment Template `deploy` script in each environment, configuring each environment as a self-contained project.

After you finish adjustments to the servers, you can configure your application using the Deployment Template and start customizing the baseline update script for your project.

Chapter 3

Creating and Duplicating Environments

This section describes the necessary steps for deploying and provisioning an application, and for transferring a defined application from one environment to another (such as copying it directly, or promoting from an authoring environment to a live server).

About a multiple server environment

A multiple server environment is commonly divided into a data procesing server, one or more MDEX Engine servers, a tools server, and optionally one or more application servers.

Typically, you create staging and production environments with dedicated servers. To add servers and additional MDEX Engines, adjust the Deployment Template `AppConfig.xml` file so that it points to the correct host names for the servers in your staging (or production) environment.

A multiple server environment is commonly structured into:

- A data processing server consisting of:
  - The MDEX Engine
  - The Platform Sevices package, including the EAC Central Server and Developer Studio
  - The Content Acquisition System
  - Application data
- One or more MDEX Engine servers consisting of:
  - The MDEX Engine
  - An EAC Agent
- A tools server consisting of:
  - The Tools and Frameworks package, including the Workbench and the Deployment Template
  - An EAC Agent

You can also have a dedicated application server to host the front-end application, or use one of the existing servers for this purpose. In staging and production environments it is advisable to configure one or more dedicated application servers.

Staging and production environments

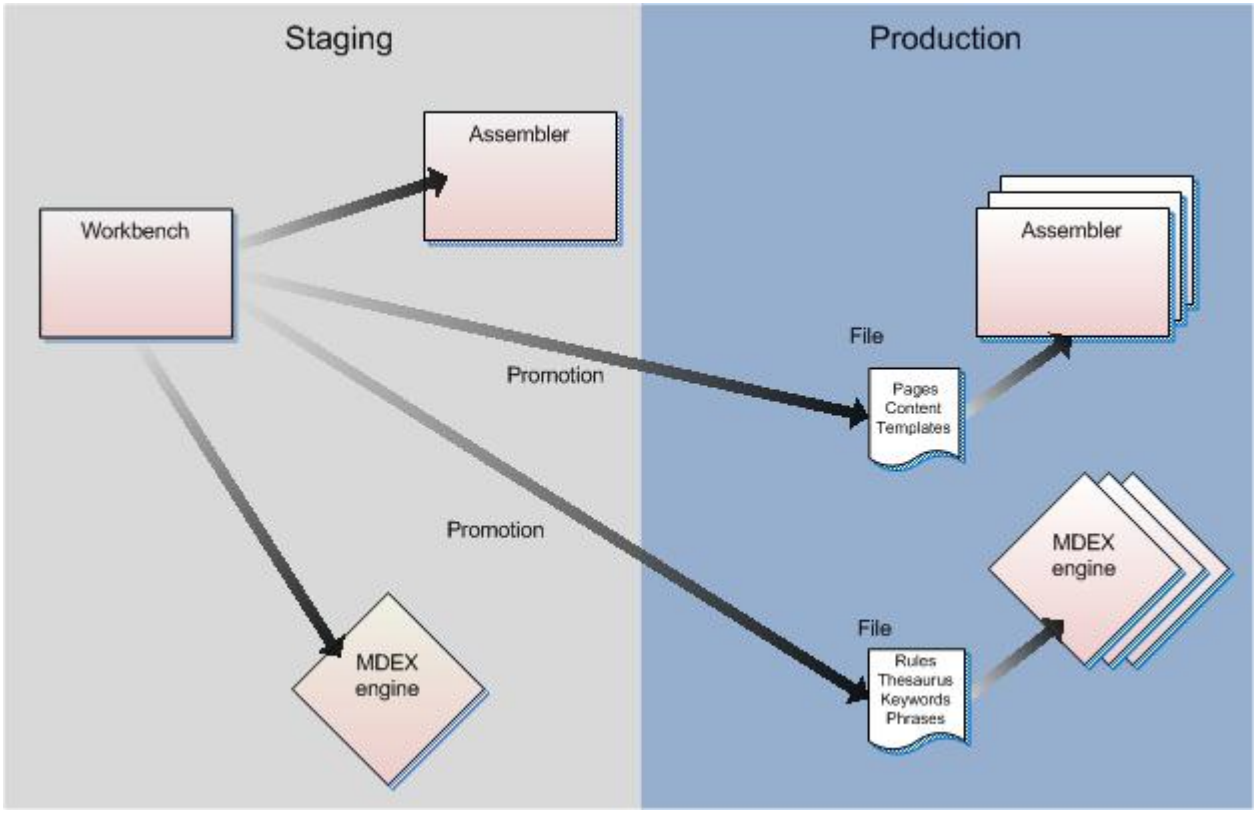
This section provides an architecture overview of staging and production environments.

Endeca applications go through a life cycle of authoring, previewing, and live modes. You author and preview in a staging environment, while your live mode is your production environment. The ability to easily push or promote application updates from a staging environment to your production environment lets you manage the application life cycle with confidence.

When you promote application updates to your production environment the following elements are promoted:

Element	Description
Content	The content that is stored and triggered based on rules. Content is published to the Assembler.
Pages	Content that is stored and delivered based on URLs. Pages are published to the Assembler.
Templates	Templates define the structure and configuration for content items and the associated editors with which the business user can configure them. Templates are published to the Assembler.
Phrases	Muliti-word search terms created with the automatic phrasing tool. Phrases are published to the MDEX engine.
Rules	Rules define the triggering configuration for delivering content: navigation state, time, or user segment. Rules are published to the MDEX engine.
Thesaurus	Thesaurus word and phrase mappings are published to the MDEX engine.
Keyword Redirects	Keywords redirected to URLs are published to the MDEX engine.

The following diagram shows the workflow for publishing these elements from the Workbench to the MDEX engine and Assembler.



The left side of the diagram shows the workflow within a staging environment. When you change content in Workbench in your staging environment, two actions occur: the Assembler pulls updated pages, content, and templates, while rules, thesaurus mappings, phrase entries, and keyword redirects are published to the MDEX engine. These actions ensure that the Workbench always has the most recent content available for authoring and previewing purposes.

The right side of the diagram shows the production environment. Once you have completed your authoring and testing, you can promote your changes to your production environment. It does not matter if your staging and production environments are on separate networks or on the same network, you can promote between environments through file transfers. The advantage of using file transfers is that the production environment has no dependency on the staging environment or even Workbench. For example you might want to keep this information in source control. File transfers let you preserve all your application content and configuration separately from your Workbench installation. Note that Workbench can be installed in the production environment, but this is not required. Requests made against the live Assembler do not require any contact with Workbench, but rather access to the Assembler 's own internal store of content and the production MDEX engine.

In this scenario, elements are moved between environments.

- Pages, content, phrases, and templates are extracted from the Workbench to a content zip file. They are designated a version number so you can maintain different exports copies of content.
- A promotion script copies the content zip file to a shared location in your production environment designated by the Assembler configuration.

  In isolated networks, you can retrieve the zip file from the staging Workbench and manually move it to a shared location in the production environment.

- All Assemblers in the production environment read the zip file from the shared location.
- Rules, thesaurus and phrase entries, and keyword redirects are extracted from the Workbench to a search configuration zip file.
- All MDEX engines read the latest version of these elements from the search configuration zip file.

  In isolated networks, you can retrieve the search configuration zip file from the staging environment and manually move it to a shared location in the production environment. All MDEX engines in the production environment read the file from shared location.

Deploying and initializing an application

This section describes how to deploy and initialize an application using the Deployment Template. You must have installed Tools and Frameworks on the machine running the EAC Central Server (part of the Platform Services package) and set environment variables used by the Guided Search software (including `ENDE¬CA_ROOT`).

About deploying applications

The Deployment Template (`deploy`) script is available for both Windows and UNIX platforms. The prompts for the `deploy.sh` script are exactly the same as the `deploy.bat` script.

In every deployment environment, one server serves as the primary control machine and hosts the EAC Central Server, while all other servers act as agents to the primary server and host EAC Agent processes that receive instructions from the Central Server.

Both the EAC Central Server and the EAC Agent run as applications inside the Endeca HTTP Service. (As mentioned in the prequisites, Tools and Frameworks only needs to be installed on the machine that hosts the EAC Central Server).

Note: Mixed-platform deployments may require customization of the default Deployment Template scripts and components. For example, paths are handled differently on Windows and on UNIX, so paths and working directories are likely to require customization if a deployment includes servers running both of these operating systems.

Deploying and initializing an application

The `deploy` script in the `deployment_template\bin` directory creates, configures, and distributes the EAC application files into the deployment directory structure.

To deploy an EAC application on Windows:

1. Start a command prompt (on Windows) or a shell (on UNIX).
2. Navigate to `ToolsAndFrameworks\<version>\deployment_template\bin`.
3. Run the `deploy` script.
   For example, on Windows:
   ```
   C:\Endeca\ToolsAndFrameworks\<version>\deployment_template\bin>deploy.bat
   ```
4. If the path to the Platform Services installation is correct, press `Enter`.
   The template identifies the location and version of your Platform Services installation based on the `ENDECA_ROOT` environment variable. If the information presented by the installer does not match the version or location of the software you plan to use for the deployment, stop the installation, reset your `ENDECA_ROOT` environment variable, and start again.

   Note: The installer may not be able to parse the Platform Services version from the `ENDECA_ROOT` path if it is installed in a non-standard directory structure. It is not necessary for the installer to parse the version number, so if you are certain that the `ENDECA_ROOT` path points to the correct location, proceed with the installation.

5. Specify a short name for the application.
   The name should consist of lower- or uppercase letters, or digits between zero and nine.
6. Specify the full path into which your application should be deployed.
   This directory must already exist. The `deploy` script creates a folder inside of the deployment directory with the name of your application and the application directory structure.
   For example, if your application name is `MyApp`, and you specify the deployment directory as `C:\Endeca\apps`, the `deploy` script installs the template for your application into `C:\Endeca\apps\MyApp`.
7. Specify the port number of the EAC Central Server, or press **Enter** to accept the default of 8888.
8. Specify the port number of Oracle Commerce Workbench, or press **Enter** to accept the default of 8006.
9. Specify the port number of the Live Dgraph, or press **Enter** to accept the default of 15000.
10. Specify the port number of the Authoring Dgraph, or press **Enter** to accept the default of 15002.
11. Specify the port number of the Log Server, or press **Enter** to accept the default of 15010.
    If the application directory already exists, the `deploy` script time stamps and archives the existing directory to avoid accidental loss of data.

12. Specify the path to the Oracle Wallet configuration file, `jps-config.xml`, or press **Enter** to accept the default path of `../../server/workspace/credential_store/jps-config.xml`.
13. Specify the path to the location where you wish to export application content, or press **Enter** to accept the default path of `../../server/workspace/state/repository`.
14. Navigate to the `control` directory of the newly deployed application.

    This is located under your EAC application directory. For example: `C:\Endeca\apps\<app dir>\control`.

15. From the `control` directory, run the `initialize_services` script.

    The script initializes each server in the deployment environment with the directories and configuration required to host your application. This script stops all application components and removes any existing provisioning associated with this application in the EAC, then adds the hosts and components in your application configuration file to the EAC and restarts them. Use caution when running this script, as it may lead to service interruption if executed on a live environment.

Once deployed, an EAC application includes all of the scripts and configuration files required to create an index and start an MDEX Engine.

If no script customization is required, the application is now ready for use. You can proceed with running a baseline update as well as any necessary CAS crawls.

However, if you need to configure an EAC applicationto reflect unique environment and data processing requirements, proceed to *Configuring an EAC Application* on page 115 before indexing and starting the MDEX Engine.

Directories created by the Deployment Template

The Deployment Template creates the directory structure described below.

The Deployment Template is designed to support operations with the MDEX Engine in the production environment. This means it must support a variety of possible configurations and their modifications. The `AppConfig.xml` file contains all possible configuration and directories required on a production server.

The Deployment Template has separate directories to ensure that the MDEX Engine operations access the information they need. It creates additional directories for any added Dgraphs.

Directory	Contents
`config/lib`	Subdirectories to store custom scripts or code for your Deployment Template project.
`config/pipeline`	The Developer Studio pipeline file and XML configuration files.
`config/report_templates`	Templates used to generate application reports.
`config/script`	The `AppConfig.xml` file and related Deployment Template scripts responsible for defining the baseline update workflow and communication from application components to the EAC Central Server.
`control`	Shell (UNIX) or batch (Windows) scripts responsible for running different operations defined within `AppConfig.xml`.
`data/incoming`	The premodified incoming data files that are ready acquisition by the Endeca pipeline and should be processed.

Directory	Contents
data/processing	Temporary data and configuration files created and stored during the baseline update process.
data/forge_output	The data and configuration files that are output from the Forge process to the Dgidx process.
data/dgidx_output	The index files that are output from the Dgidx process.
data/dgraphs	The copy of the index files used by an instance of the MDEX Engine.
data/state	Autogenerated dimensions files.
logs	Various log files within subdirectories, such as Dgidx logs.
reports	Generated reports.

Configuring automated, file-based deployment

The Deployment Template supports a file-based configuration option to simplify the deployment of an EAC Application. This automation may be especially useful during development, when the same deployment process must be repeated many times.

You can create a deployment configuration file that contains name/values that satisfy the `deploy` script prompts, so you do not have to respond to the prompts manually. You specify the deployment configuration file as an argument to the `--install-config` flag when you run the `deploy` script.

The deployment configuration file should specify the application name, deployment path, deployment type, and all ports. The following example specifies the installation of a Dgraph deployment named `Discover`:

```
<install app-name="Discover">
  <deployment-path>/localdisk/endeca/apps</deployment-path>
  <base-module type="dgraph" />
  <options>
    <option name="eac-port">8888</option>
    <option name="workbench-port">8006</option>
    <option name="dgraph1Port">15000</option>
    <option name="authoringDgraphPort">15002</option>
    <option name="logserverPort">15010</option>
    <option name="jps-config-location">ToolsAndFrameworks/<version>/serv¬
er/workspace/credential_store/jps-config.xml</option>
  </options>
</install>
```

To configure automated/file-based deployment:

1. Start a text editor, create a new text file, and copy/paste the example above.
2. If necessary, modify the default port values for the EAC Central Server, Workbench, Live Dgraph, Authoring Dgraph, and the Log Server to new values.
3. Save and close the file.
4. Run the `deploy` script and specify the `--install-config` flag and the location of the deployment configuration file.

The following example specifies the deployment descriptor (`deploy.xml`) for a version of the Discover Electronics reference application, then the `--install-config` flag with an argument to the deployment configuration file (`app-install-config.xml`):

```
./deploy.sh --app /localdisk/endeca/ToolsAndFrameworks/*/reference/discover-
data/deploy.xml --install-config /localdisk/infrontSetupScripts/config/app-in¬
stall-config.xml --no-prompt
```

When a configuration file is specified for the Deployment Template, the deployment attempts to retrieve and validate required information from the document before proceeding. If any information is missing or invalid, the Deployment Template prompts for that information, as described in previous sections. To truly automate the install process, the `--no-prompt` flag may be passed to the installer, instructing it to fail (with error messages) if any information is missing and to bypass interactive verification of the Guided Search version.

Running a baseline update with test data

A deployed application includes test data that you can process with baseline update scripts, baseline test data, and a baseline Forge pipeline. Because this task describes test data, not production data, you use the `load_baseline_test_data` script to simulate the data extraction process (or to set the data readiness signal, in the case of an application that uses a non-extract data source).

The `load_baseline_test_data` script loads the test data stored in `<app dir>/test_data/baseline` and runs the `set_baseline_data_ready_flag` script which sets a flag in the EAC indicating that data has been extracted and is ready for baseline update processing.

Note: This script is not required in applications that use CAS to produce MDEX-compatible output.

When you are done familiarizing yourself with the data processing steps and the test data, see *Running a baseline update with production data* on page 36. Processing production data requires the following changes to an application' s configuration:

- Replace the step to run `load_baseline_test_data` with a data extraction process that delivers production data into the `<app dir>/test_data/baseline` directory. Delete the `data.txt` file from `<app dir>/test_data/baseline`. This step is not necessary if your application does not use data extracts: for example, if your application retrieves data directly from a database via ODBC or JDBC or from a CAS crawl.
- Set the `baseline_data_ready` flag in the EAC. You set the `baseline_data_ready` flag by making a Web service call to the EAC or by running the `set_baseline_data_ready_flag` script.

To run a baseline update with test data:

1. Ensure that the Endeca HTTP Service is running on each server in the deployment environment and that you have already deployed and initialized an application.
2. Start a command prompt (on Windows) or a shell (on UNIX).
3. Navigate to the `control` directory of deployed application.

   This is located under your application directory. For example: `C:\Endeca\apps\<app dir>\control`.

4. Run the `load_baseline_test_data` script.
   - On Windows:
     ```
     <app dir>\control\load_baseline_test_data.bat
     ```
   - On UNIX:
     ```
     <app dir>/control/load_baseline_test_data.sh
     ```

5. Run the `baseline_update` script.
   - On Windows:
     ```
     <app dir>\control\baseline_update.bat
     ```
   - On UNIX:
     ```
     <app dir>/control/baseline_update.sh
     ```
6. Examine the indexed data in a Guided Search front-end application.
   For example, start a Web browser and open the JSP reference application at `http://localhost:8006/endeca_jspref`.
   You should see 10 records.

Running a baseline update with production data

You run the `baseline_update` script to process production data and distribute the resulting index files to one or more Dgraphs. Production data may come from any number of sources including data extracts, CAS crawls, or direct calls to a database via ODBC or JDBC.

To run a baseline update with production data:

1. Ensure that the Endeca HTTP Service is running on each server in the deployment environment and that you have already deployed and initialized an application.
2. Replace the default Forge pipeline (Developer Studio configuration files) in `<app dir>/config/pipeline` with the Developer Studio configuration files for your application. For details, see *Replacing the Default Forge Pipeline* on page 134.
3. Replace the baseline test data stored in `<app dir>/test_data/baseline` with production data for the application. This step varies depending on your application requirements. It can include any of the following approaches:
   - Add a data extract file to the `<app dir>/test_data/baseline` and delete the test data extract.
   - Set up a CAS crawl to run as part of the `baseline_update` script.
   - Make a direct call to a database via ODBC or JDBC.
4. Start a command prompt (on Windows) or a shell (on UNIX).
5. Navigate to the `control` directory of deployed application.

   This is located under your application directory. For example: `C:\Endeca\apps\<app dir>\control`.
6. Set the `baseline_data_ready` flag in the EAC by running the `set_baseline_data_ready_flag` script.
   - On Windows:
     ```
     <app dir>\control\set_baseline_data_ready_flag.bat
     ```
   - On UNIX:
     ```
     <app dir>/control/set_baseline_data_ready_flag.sh
     ```

   Note: This script is not required in applications that use CAS to produce MDEX-compatible output.
7. Run the `baseline_update` script.

- On Windows:
  ```
  <app dir>\control\baseline_update.bat
  ```
- On UNIX:
  ```
  <app dir>/control/baseline_update.sh
  ```

8. Examine the indexed data in a Guided Search front-end application.
   For example, start a Web browser and open the JSP reference application at `http://localhost:8006/endeca_jspref`.

Running a partial update with production data

You run the `partial_update` script to process *incremental changes* in production data and distribute the resulting index files to one or more Dgraphs. Production data may come from any number of sources including data extracts, CAS crawls, or direct calls to a database via ODBC or JDBC.

For more information on partial updates, see the *MDEX Engine Partial Updates Guide*.

To run a partial update with production data:

1. Ensure that the Endeca HTTP Service is running on each server in the deployment environment and that you have already deployed and initialized an application.
2. Replace the default Forge pipeline (Developer Studio configuration files) in `<app dir>/config/pipeline` with the Developer Studio configuration files for your application. For details, see *Replacing the Default Forge Pipeline* on page 134.
3. Provide the partial data (incremental data changes since the last baseline update). This step varies depending on the application requirements. It can include any of the following approaches:
   - Add a data extract file to the `<app dir>/test_data/partial`.
   - Set up a CAS crawl to run as part of the `baseline_update` script.
   - Make a direct call to a database via ODBC or JDBC.
4. Start a command prompt (on Windows) or a shell (on UNIX).
5. Navigate to the `control` directory of deployed application.

   This is located under your application directory. For example: `C:\Endeca\apps\<app dir>\control`.
6. Set the `partial_data_ready` flag in the EAC by running the `set_partial_data_ready_flag` script.
   - On Windows:
     ```
     <app dir>\control\set_partial_data_ready_flag.bat
     ```
   - On UNIX:
     ```
     <app dir>/control/set_partial_data_ready_flag.sh
     ```
7. Run the `partial_update` script.
   - On Windows:
     ```
     <app dir>\control\partial_update.bat
     ```
   - On UNIX:
     ```
     <app dir>/control/partial_update.sh
     ```

8. Examine the indexed data in a Guided Search front-end application.
   For example, start a Web browser and open the JSP reference application at `http://localhost:8006/endeca_jspref`.

Running CAS crawls

In your `DataIngest.xml` code, you can run baseline or partial updates that include CAS crawls using the methods available in `ContentAcquisitionServerComponent`.

For details about `ContentAcquisitionServerComponent`, see the *EAC Component API Reference for CAS Server (Javadoc)* installed in `CAS\<version>\doc\cas-dt-javadoc` and see the CAS examples in *Deployment Template Script Reference* on page 255.

Provisioning an application with the Endeca Application Controller

Provisioning is the task of defining an application within the Endeca Application Controller. This process registers the application with the EAC and defines the various components (such as Forge, the Dgidx, and any Dgraphs) and their associated scripts and configuration files.

To use the Deployment Template to provision an application with the EAC, you must first deploy the application on disk.

You can provision an application through the Deployment Template by running the `initialize_services` script.

To provision an application:

1. Navigate to the `control` directory of your deployed application on disk.
   For the default Discover Electronics reference application on Windows this is `C:\Endeca\apps\Discover\control`.
2. Run the `initialize_services` batch or shell script.
   This creates an application definition within the EAC.

Updating application configuration for a provisioned application

When you change your application configuration by modifying the `AppConfig.xml` file, you must also push the changes to the EAC. You can update the application definition without running a baseline update.

To update application configuration for a provisioned application:

1. Navigate to the `config\script` directory of your deployed application.
   For the default Discover Electronics reference application on Windows, this is `C:\Endeca\apps\Discover\config\script`.
2. Update the component definition in `AppConfig.xml`.
3. Navigate to the `control` directory of your deployed application.
4. Upload the updated application definition to the EAC by running `runcommand` with the `--update-definition` flag:

For example:

```
C:\Endeca\apps\Discover\control> runcommand.bat --update-definition
```

This updates application provisioning in the EAC.

Incremental provisioning updates existing components, but does not remove components that have been deleted from `AppConfig.xml`. To remove a provisioned component, see *Removing a component from an application* on page 39.

Removing a component from an application

You can run the `runcommand` script with the `stop` and `removeDefinition`commands to stop and remove an application component.

To remove a component from an application:

1. Navigate to the `control` directory of your deployed application.
   For the default Discover Electronics reference application on Windows, this is `C:\Endeca\apps\Discover\control`.
2. Stop the component by running `runcommand <component name> stop`:
   For example:
   ```
   C:\Endeca\apps\Discover\control> runcommand.bat AuthoringDgraph stop
   ```
3. Remove the component by running `runcommand <component name> removeDefinition`:
   For example:
   ```
   C:\Endeca\apps\Discover\control> runcommand.bat AuthoringDgraph removeDefinition
   ```
4. Navigate to the `config\script` directory of your deployed application.
5. Remove the component definition from `AppConfig.xml`.

Backing up application provisioning with the `eaccmd` utility

You can create a copy of your application provisioning information by running the `eaccmd describe-app` command and redirecting its output to a file. This is a useful technique for testing, since it allows you to easily revert your application provisioning if you make unwanted changes.

The output of the `eaccmd describe-app` command is in the same XML format used by the `eaccmd define-app` command.

For more information on `eaccmd` commands, see the *Oracle Endeca Application Controller Guide*.

To back up application provisioning with the `eaccmd` utility:

Run the following command:

```
eaccmd describe-app --app [application name] --canonical >application.xml
```

The `describe-app` command generates XML for the provisioned application specified by the `--app` flag. The `--canonical` flag forces the application's description to use absolute paths for every entry, and to resolve references such as `.` and `..` to produce straightforward paths. The `>application.xml` modifier redirects the output to the specified file in the current working directory.

The output, `application.xml`, can later be passed in to the `define-app` command to restore the provisioned application definition in the EAC.

Configuring the promotion method for your application

There are two methods of promotion: direct and file-based. Direct promotion is typically used *within* your staging environment, while file-based promotion is used *between* your staging and production environments.

You specify the method to use in your Assembler configuration. The Assembler implementation included with Tools and Frameworks is configured through Spring. This guide assumes an application based around the included Assembler implementations. You can provide your own implementation if you wish to use an alternate means of configuring the Assembler. In the reference implementations, you specify which method of promotion to use in the `assembler.properties` file by specifying a store type, and you define the properties of both stores in the `assembler-context.xml` file.

1. Navigate to the Assembler properties file for your application, `WEB-INF/assembler.properties`.
2. Open `assembler.properties` in a text editor.
3. Edit the `store.factory` setting.

   Store factories retrieve and store content and configuration specific to the application. There are two store types, but only one store would be used in any Assembler configuration.

   - ecrStoreFactory - store for direct promotion used in a staging environment
   - fileStoreFactory - store for file-based promotion used in a production environment.

   For example, in the production environment for the Discovery Electronics application, the store setting is the following:

   ```
   preview.enabled=false
   store.factory=fileStoreFactory
   user.state.ref=liveUserState
   media.sources.ref=liveMediaSources
   repository.configuration.path=./repository/${workbench.app.name}
   ```

   The `repository.configuration.path` was set when you ran the deployment template to create the application. This is the location where content is extracted to when you promote content.
4. Save and close the file.
5. Navigate to the Assembler context file for your application, `WEB-INF/assembler-context.xml`
6. Open `assembler-context.xml` and find the Administration services section of the file.

   For example, in the production environment for the Discovery Electronics application, the section looks like the following:

   ```
   <!--
          ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

           ~ Administration Service.
       -->
      <bean id="adminService" class="com.endeca.infront.assembler.servlet.admin.Ad¬
   ministrationService">
           <property name="storeFactory" ref="${store.factory}"/>
       </bean>
   ```
7. If you are using the file-based method of promotion, scroll down, and edit the properties in the `fileStoreFac¬tory` bean with values that are appropriate for your site.

For example, the `configurationPath` should be the the location where content is extracted to when you promote content.

```
<bean id="fileStoreFactory" class="com.endeca.infront.content.source.FileStoreFac¬
tory"
          init-method="init">
        <property name="configurationPath" value="${repository.configura¬
tion.path}"/>
    </bean>
```

8. If you are using the direct method of promotion, edit the `ecrStoreFactory` bean with values appropriate for your application:

```
<bean id="ecrStoreFactory" class="com.endeca.infront.content.source.EcrStoreFac¬
tory"
          init-method="init" destroy-method="destroy">
        <property name="isAuthoring" value="true"/>
        <property name="appName" value="${workbench.app.name}" />
        <property name="host" value="${workbench.host}" />
       <property name="clientPort" value="${workbench.publishing.clientPort}"/>

       <property name="serverPort" value="${workbench.publishing.serverPort}"/>

    </bean>
```

9. Save and close the `assembler-context.xml` file.
10. Restart Endeca Tools Services.

About promoting content

The **promote_content** script promotes content and search configuration from the staging environment to the production environment.

The promote content script uses the following components in the `<app dir>\config\script\WorkbenchConfig.xml` file to promote content:

- `IFCR.exportConfigSnapshot(LiveDgraphCluster)` exports search configuration from Workbench in your staging environment.
- `IFCR.exportApplication()` exports content and content configuration from Workbench in your staging environment .
- `LiveDgraphCluster.applyConfigSnapshot();` imports search configuration to the MDEX engines in your production environment.
- `AssemblerUpdate.updateAssemblers()` is a utility component for programmatically calling all your assemblers. It tells all the Assemblers in your production environment to update content and content configuration from disk.

The following table provides more information on these components.

Component	Description
`IFCR.exportConfigSnapshot(LiveD¬graphCluster)`	Exports a snapshot of the current dgraph configuration for the Live dgraph cluster. The command Writes the configuration into a single zip file. The zip is written to the local config directory for the live dgraph cluster. A snapshot of the current rules, thesaurus entries, and keyword redirects is extracted from the staging Dgraph to a search configuration zip file, *<app-name>*`.mdex.`*<date-stamp>*`_`*<time-stamp>*`.zip`. The zIp

Component	Description
	file is written to the local configuration directory *<app dir>*`./data/dgraphcluster/LiveDgraphCluster/config_snapshots` directory. This directory is specified in the `LiveDgraphCluster.xml` file. A key file is also stored in this location that contains the latest version of the file.
`IFCR exportApplication ()`	Pages, content, and templates are extracted from Workbench to a content export zip file, *<app-name>.<date-stamp>_<time-stamp>*`.zip`, in the `ToolsAndFrameworks\`*<version>*`\server\workspace\state\repository\` by default. You specify this directory when you initially run the `deploy` script to create your application. You can only export nodes that represent content relevant to this application. The command creates an application specific subdirectory.
`LiveDgraphCluster.applyConfigSnapshot()`	Updates the Dgraph servers configured as part of the cluster using a zip file tagged as the current file. The `LiveDgraphCluster` component specifies the Dgraphs used in the live environment and a script that pushes configuration from Workbench to each Dgraph in the live cluster. If the name of the cluster is different or there are multiple clusters, add a line for each cluster. The file tagged as the current file is used to define which zip file in the directory is used to update the Dgraphs.
`AssemblerUpdate updateAssemblers()`	Updates all the Assemblers configured for your application. The `AssemblerUpdate` component can take a list of Assembler clusters from the `LiveAppServerCluster.xml` file and build URLs and POST requests for each Assembler. Upon this request, the Assembler attempts to read the content from disk if new content is available. If it is the same content, it does not update.

Promoting content from staging to production environments

After you have completed your authoring and testing in a staging environment, you can promote your changes to your production environment.

You can promote content to Assemblers and MDEX engines in your production environment by running the `promote_content` script.

1. Navigate to the `<app dir>\config\script` directory on Windows (`<app dir>/config/script on UNIX`).
2. Open the `WorkbenchConfig.xml` file in an text editor and update any components as necessary.
   See *About promoting content* on page 41 for more information on the components in the script.
3. Save and close the file.
4. Navigate to the `<app dir>\control` directory on Windows (`<app dir>/control on UNIX`).
5. From the command prompt, run `promote_content`.

Promoting content from staging to production environments in different networks

You can promote content from staging to production environments in isolated networks, but you must manually move the promoted content to a location in the production environment where Assemblers and MDEX engines can access it.

The promotion procedure for promoting content between isolated environments consists of running the promote content script first in your staging environment and then in your production environment. Before you run the scripts, edit the `WorkbenchConfig.xml` files in both environments. The promote content script in your staging environment uses the `WorkbenchConfig.xml` file to export content to zip files. After you move the zip files to a shared location in your production environment, you run the promote content script in your production environment. This script uses the `WorkbenchConfig.xml` file in the production environment to update your Assemblers and MDEX engines with the content from the zip files.

1. Navigate to the `<app dir>\config\script` directory on Windows (`<app dir>/config/script on UNIX`) in your staging environment.
2. Open the `WorkbenchConfig.xml` file in an text editor and delete or comment out the following lines:

   ```
   LiveDgraphCluster.applyConfigSnapshot();
   ```

   ```
   AssemblerUpdate.updateAssemblers();
   ```

   These components are for importing content and you want to use the promote content script in your staging environment only for exporting content.
3. Update the following lines as necessary to reflect the properties of your specific staging environment:

   ```
   IFCR.exportConfigSnapshot(LiveDgraphCluster);
   ```

   ```
   IFCR.exportApplication();
   ```

   See *About promoting content* on page 41 for more information on the script.
4. Navigate to the `<app dir>\config\script` directory on Windows (`<app dir>/config/script on UNIX`) in your production environment.
5. Open the `WorkbenchConfig.xml` file in an text editor and delete or comment out the following lines:

   ```
   IFCR.exportConfigSnapshot(LiveDgraphCluster);
   ```

   ```
   IFCR.exportApplication();
   ```

   These components are for exporting content and you want to use this promote content script in your production environment only for importng content to your production environment.
6. Update the following lines as necessary to reflect the properties of your specific production environment:

   ```
   LiveDgraphCluster.applyConfigSnapshot();
   ```

   ```
   AssemblerUpdate.updateAssemblers();
   ```

7. Navigate to the `<app dir>\control` directory on Windows (`<app dir>/control on UNIX`) in your staging environment.
8. Run `promote_content`.

   The command exports content to zip files in your staging environment.
9. Move the two exported zip files to the shared locations in your production environment that you specified for your production environment.
10. Navigate to the `<app dir>\control` directory on Windows (`<app dir>/control on UNIX`) in your production environment.

11. Run `promote_content`.

    The command imports content from the zip files to your production environment.

Content export file formats

This section describes the file formats that Workbench content in the ECR gets exported to when you promote content.

Workbench content is stored in the Endeca Configuration Repository (ECR). The ECR is a Web application that runs in the Endeca Tools Service. The ECR uses a JSR-170-compliant Java Content Repository to store the content and configuration related to Endeca applications.

When you promote content, the ECR extracts content information to a zip file containing JSON files to represent nodes. The ECR node tree structure is represented as a file structure. Content and content configuration information is located in these nodes:

- content - rule-based content
- pages - static, URL-based content
- templates - structure and content configuration
- services - node is only used internally for accessing any merchandising content configured in Rule Manager.

All data properties on the nodes are represented in the JSON files in the zip.

Each repository node type has a corresponding export format. Content items, pages, and templates have specific export formats with specific ECR types. Services nodes are only used for Workbench deployments that contain Rule Manager instead of Experience Manager.

Content item export format

Content Items JSON files have the preserved state, and a specified ecr:type. Triggers are all consolidated into the same JSON file.

Property	Description
workflowState	Determines whether or not the content item is active or inactive.
priorty	Determines the likelihood that content displays if there are other content items with overlapping trigger criteria.
triggers	Criteria that cause the content to display
ecr:type	Identifies the node as a content-item

Here is an example of the export JSON file for the Brand - Canon content item in the Discover application.

```
{
    "ecr:type": "content-item",
    "workflowState": "ACTIVE",
    "priority": 20,
    "triggers": [
        {
            "searchTerms": "canon",
            "matchmode": "MATCHPHRASE",
            "exactLocation": false
        },
        {
```

```
            "dvalIDs": ["4294967266"],
            "exactLocation": true
        }
    ]
}
```

Page export format

Page JSON files have an ecr:type of endeca:page.

Property	Description
contentType	Identifies the content type.
ecr:type	Identifies the node as a page.

Here is an example of the export JSON file for the Detail page in the Discover application.

```
{
    "contentType": "Page",
    "ecr:type": "page"
}
```

Template export format

Template JSON files have an ecr:type of template.

Property	Description
ecr:type	Identifies the node as a template

Here is an example of the export JSON file for the Three Column Page template in the Discover application.

```
{
    "ecr:type": "template"
}
```

Search configuration file formats

This section describes the file formats that the Workbench search configuration gets exported to when you promote content.

Just like Workbench content information, search configuration information is stored in nodes in the Endeca Configuration Repository (ECR). When you promote content, the ECR exports the search configuration to a zip file containing XML files to represent nodes instead of JSON files as in the case of content. Exporting search configuration information results in a zip file with the following XML files:

- <*app_name*>.merch.xml -- Contains merch rule information.
- <*app_name*>.merch_rule_group.content.<*category*>.xml -- Contains the rule groups that logically organize merch rules into categories.
- <*app_name*>.merchzones.xml -- Contains the zones: collections of merch rules that ensure promoted records are always produced.
- <*app_name*>.phrases.xml -- Contains phrases that were created using the automatic phrasing tool.
- <*app_name*>.redirects.xml -- Contains keyword redirects.
- <*app_name*>.thesaurus.xml -- Contains thesaurus entries.

XML file format example

Here is an example of a thesaurus XML file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<THESAURUS>
  <THESAURUS_ENTRY_ONEWAY>
    <THESAURUS_FORM_FROM>huge</THESAURUS_FORM_FROM>
    <THESAURUS_FORM_TO>extra large</THESAURUS_FORM_TO>
    <THESAURUS_FORM_TO>big</THESAURUS_FORM_TO>
    <THESAURUS_FORM_TO>large</THESAURUS_FORM_TO>
  </THESAURUS_ENTRY_ONEWAY>
</THESAURUS>
```

For more information on XML file formats, see the *Oracle Commerce Guided Search Platform Services XML Reference Guide* on the Oracle Technology Network.

Promoting content using the direct method of promotion

Oracle recommends that you promote content between the staging and production environments using the file-based method described in the previous topics. Alternatively, you can still promote content using the direct method if necessary.

To promote content using the direct method, you must edit the application's `assembler-context.xml file` and `promote_content` script.

1. Navigate to the Assembler context file for you application, `WEB-INF/assembler-context.xml`.
2. Open `assembler-context.xml` in a text editor.
3. Find the `ecrStoreFactory` bean and change the value of the `isAuthoring` property to *false*.
   ```
   <property name="isAuthoring" value="false"/>
   ```
4. Navigate to the `<app dir>\config\scriptdirectory` on Windows (`<app dir>/config/script` on UNIX), and open `WorkbenchConfig.xml`
5. Uncomment the following line:
   ```
   // IFCR.promoteFromAuthoringToLive();
   ```
6. Delete or comment out the following lines:
   ```
   IFCR.exportConfigSnapshot(LiveDgraphCluster);
   ....
   IFCR.exportApplication();
   ....
   LiveDgraphCluster.applyConfigSnapshot();
   ....
   AssemblerUpdate.updateAssemblers();
   ```
7. Save and close the file.
8. Navigate to the `<app dir>\control` directory on Windows (`<app dir>/control on UNIX`).
9. From the command prompt, run `promote_content`.

Configuring the application on multiple servers

To configure an application on multiple servers, you specify each server in your environment by modifying the application configuration files stored in `<app dir>/config/script.`

In a multiple server environment, the application configuration files typically describe the following:

- One or more MDEX Engine servers
- A Data Processing (ITL) server
- A Tools server

To configure the application on multiple servers:

1. You should have already deployed an application using the Deployment Template.
2. Modify the application configuration files located in `<app dir>/config/script` according to the following topics.

Adding MDEX Engine servers and Dgraphs to the servers

When you initially deploy a new application using the Deployment Template, it sets up an environment in which two Dgraphs are running on the same MDEX Engine server. One Dgraph is for the authoring application and the other Dgraph is for the live application. You can optionally change this configuration and add one or more MDEX Engine servers, with one or more Dgraphs, to either the authoring or live application.

This task describes how to add an MDEX Engine Server, and Dgraphs on that server, to a *live* application by modifying settings in the `LiveDgraphCluster.xml` file. However, the task is essentially the same as adding an MDEX Engine server to an *authoring* application. In an authoring application, you modify `AuthoringDgraphCluster.xml` instead of `LiveDgraphCluster.xml`. Within each configuration file, the setting are the same to define `host`, `dgraph`, and `dgraph-cluster` options.

To add MDEX Engine servers and Dgraphs to the servers:

1. Navigate to `<app dir>/config/script` and open `LiveDgraphCluster.xml` in a text editor.
2. Locate the `Live MDEX Hosts` section of the file and add one or more `host` elements to represent an MDEX Engine server. Specify host name and EAC Agent port information for the new server.
   This example adds a server named `LiveServerB.CompanyName.com`:

```
 <!--
   ###################################################################
   # Live MDEX Hosts - The machines used to host all MDEX processes
   # for the 'live environment' MDEX cluster.
   #
 -->
 <host id="LiveMDEXHostA" hostName="LiveServerA.CompanyName.com" port="8888"
/>
 <host id="LiveMDEXHostB" hostName="LiveServerB.CompanyName.com" port="8888"
/>
```

3. Add the Dgraphs that run on the new server to the `Dgraph Cluster` block.
   This example adds `DgraphB1` and `DgraphB2` to the new `LiveServerB.CompanyName.com` server:

```
 <!--
   ###################################################################
   # Live Dgraph Cluster - The 'live environment' MDEX cluster.
   #
 -->
 <dgraph-cluster id="LiveDgraphCluster" getDataInParallel="true" enabled="true">

   <dgraph ref="DgraphA1" />
   <dgraph ref="DgraphA2" />
   <dgraph ref="DgraphB1" />
   <dgraph ref="DgraphB2" />
 </dgraph-cluster>
```

4. Add a Dgraph process definition for each Dgraph on the new server.
   This example adds `DgraphB1` and `DgraphB2` to the new `LiveServerB.CompanyName.com` server:

```
...
  <dgraph id="DgraphB1" host-id="LiveMDEXHostB" port="15010"
          post-startup-script="LiveDgraphPostStartup">
    <properties>
      <property name="restartGroup" value="1" />
      <property name="DgraphContentGroup" value="Live" />
    </properties>
    <log-dir>./logs/dgraphs/DgraphB1</log-dir>
    <input-dir>./data/dgraphs/DgraphB1/dgraph_input</input-dir>
    <update-dir>./data/dgraphs/DgraphB1/dgraph_input/updates</update-dir>
  </dgraph>

  <dgraph id="DgraphB2" host-id="LiveMDEXHostB" port="15010"
          post-startup-script="LiveDgraphPostStartup">
    <properties>
      <property name="restartGroup" value="2" />
      <property name="DgraphContentGroup" value="Live" />
    </properties>
    <log-dir>./logs/dgraphs/DgraphB2</log-dir>
    <input-dir>./data/dgraphs/DgraphB2/dgraph_input</input-dir>
    <update-dir>./data/dgraphs/DgraphB2/dgraph_input/updates</update-dir>
  </dgraph>
...
```

5. Save and close `LiveDgraphCluster.xml`.

Additional customization tasks

After you adjust the Deployment Template workflow to reflect the location of your incoming data and the topology of the servers in your environment, you can start working on the baseline update script for your project. You can later run update scripts within the Deployment Template.

Running the baseline update script on your project's data is similar to running it on the sample data. For more information on Deployment Template customization and capabilities, see *About configuring an EAC application* on page 115.

Replicating application definitions across environments

Endeca applications typically go through a life cycle of development, test, use and modification. The ability to replicate application definitions across heterogeneous environments can greatly simplify the management of this life cycle, as well as helping with backup and recovery.

About replicating application definitions

Creating a common application definition can simplify administrative operations such as deployment, provisioning, backup, and recovery.

The same application definition can apply across varying hardware and software environments:

- A primary development environment, where the application configuration and files are created and maintained.

- A staging and test environment, where a content administrator creates the dynamic content for an application.
- A production environment, where the configured application is available to end users.
- One or more secondary development environments, where an application definition can be recycled as a template for new applications.
- A fallback environment, where the application can be redeployed and restarted in the event of a problem with the main production environment.

These environments may include variations in the numbers and types of servers, operating systems, and network architecture . For example, a development environment might be a single Unix workstation, a test environment might be a set of virtual machines running on a Windows server, and a production environment might include a dedicated subnet connecting an MDEX Engine server and multiple application servers and log servers.

The steps to replicating application definitions across environments are:

- Identify the artifacts that make up your application definition.
- Create custom files for environment-specific settings.
- Control paths to ensure interoperability across environments.
- Automate file collection.
- Replicate application definitions across environments.
- Select an approach to avoiding synchronization conflicts.

Part of the solution is to develop with the Guided Search Deployment Template. The Deployment Template utilizes the EAC to manage application definitions across servers in each environment. Although each environment may contain multiple machines, including the MDEX Engine Server, the ITL Server, and application servers, the application definition is stored in only one of them: the EAC Central Server. The Deployment Template generates application control scripts that keep the other servers updated with the definition stored on the EAC Central Server.

A typical situation is illustrated in the following diagram. In this example, an administrator maintains three environments, one for development, a second for staging, testing and adjusting applications, and a third for running the application in production.

The administrator wants to synchronize the application across the three environments, so that changes made in the development environment can be moved quickly to the staging environment, and so that adjustments made in the staging environment can be deployed easily both to the production environment, and back to the development environment (where they can be reflected in new versions of the application under development).

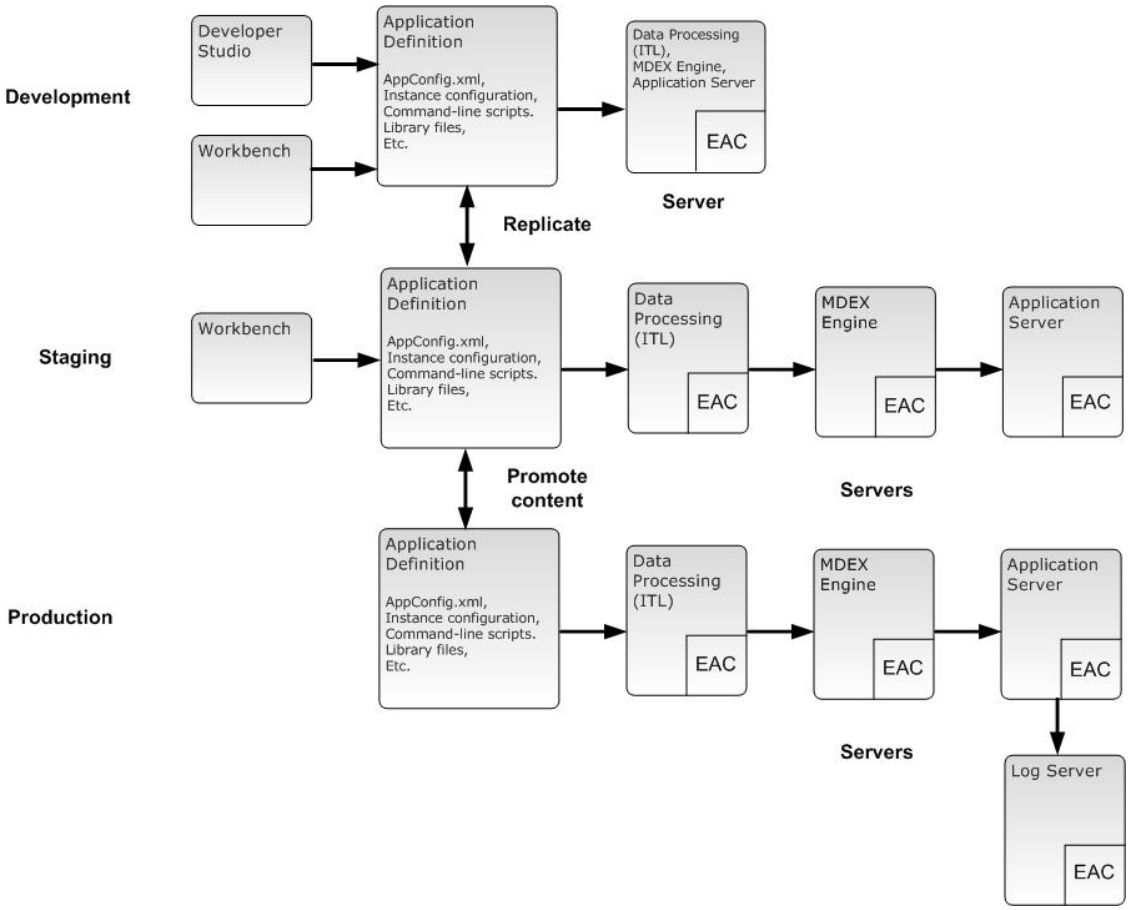
The administrator is developing with the Deployment Template, so when a new or modified application definition is moved to the EAC Central Server in each environment, the EAC takes care of propagating the necessary parts of the application definition across the servers in that environment.

But to simplify the replication process as much as possible the administrator must also:

- Create application definitions that can operate with little or no modifications in all environments.
- Automate the process of replicating the elements of the application definition between environments.

Similar considerations would apply if the administrator wanted to:

- Replicate the application definition to a secondary development environment and use it as a basis of a new application.
- Create a copy of the application that can be stored in a safe location, and then quickly replicated to a backup environment or disaster recovery site, even if the characteristics of that site differ from the original environment.



Collecting the files that make up an application

After you modify an application configuration, you collect all of the files and artifacts for the application definition and store them in the application's <app dir> directory, so they can be replicated to the other environments. This requires collecting files from various locations, including the Deployment Template's<app dir> directory, the Workbench, and arbitrary locations used by the application's scripts.

1. Use your editor of choice and specify all the Workbench parts of the application definition.
2. In the same file, combine the Workbench parts with the remaining application definition stored in the Deployment Template's<app dir>directory.

Running the script results in collecting and storing all of the artifacts in subdirectories under <app dir>, such as /config, /control, /test-data, data/state, and data/incoming.

A sample collection script, collect-app.bat, is shown below. You can write a similar script to fit your Endeca environment. This particular sample is provided as reference only.

```
set app=wine
set wbench=localhost:8006
call %~dp0..\config\script\set_environment.bat
call %~dp0runcommand.bat ConfigManager updateWsConfig
xcopy /y %~dp0\..\data\complete_index_config %~dp0\..\config\pipeline
set templ=%~dp0\..\config\templates
mkdir %templ%
```

```
emgr_update --app_name %app% --host %wbench% --action get_templates --dir
%~dp0\..\config\templates
```

You can also create a script to upload relevant parts of the updated application definition to the Workbench from the<app dir>directory. A sample script for upload.bat is shown as follows:

```
set app=new-wine
set wbench=localhost:8006
call %~dp0..\config\script\set_environment.bat
emgr_update --app_name %app% --host %wbench% --action update_mgr_settings --dir
%~dp0\..\config\pipeline --prefix %ENDECA_PROJECT_NAME%
emgr_update --app_name %app% --host %wbench% --action set_templates --dir
%~dp0\..\config\templates
```

Note: Be careful when running the collect_app.bat and upload.bat scripts. These scripts can overwrite settings in the Developer Studio and the Workbench.

Changing the IP address for the EAC Central Server machine

This topic describes how to change the IP address of an EAC Central Server machine in an environment featuring remote MDEX host machines, where the AppConfig.xml files use host names rather than IP addresses for the machines.

To change the IP address for an EAC Central Server machine:

1. Stop the Endeca HTTP service on both the EAC Central Server and remote MDEX host.
2. Change the networkaddress.cache.ttl in the java.security file to 0 on both machines.

   By default, this file is located in %ENDECA_ROOT%\j2sdk\jre\lib\security.
3. Change the IP address of the EAC Central Server, making sure that the MDEX host operating system can resolve the EAC Central Server at the new IP address.
4. Restart the Endeca HTTP service on both machines.
5. Revert the changes to the java.security file mentioned in Step 2 above by changing networkaddress.cache.ttl back to -1, and restart the Endeca HTTP service on both machines.

   This prevents subsequent DNS spoof attacks.

Chapter 4

Working with Multiple Sites

This section describes how to deploy additional sites in your application, and how to configure and maintain the sites in your application.

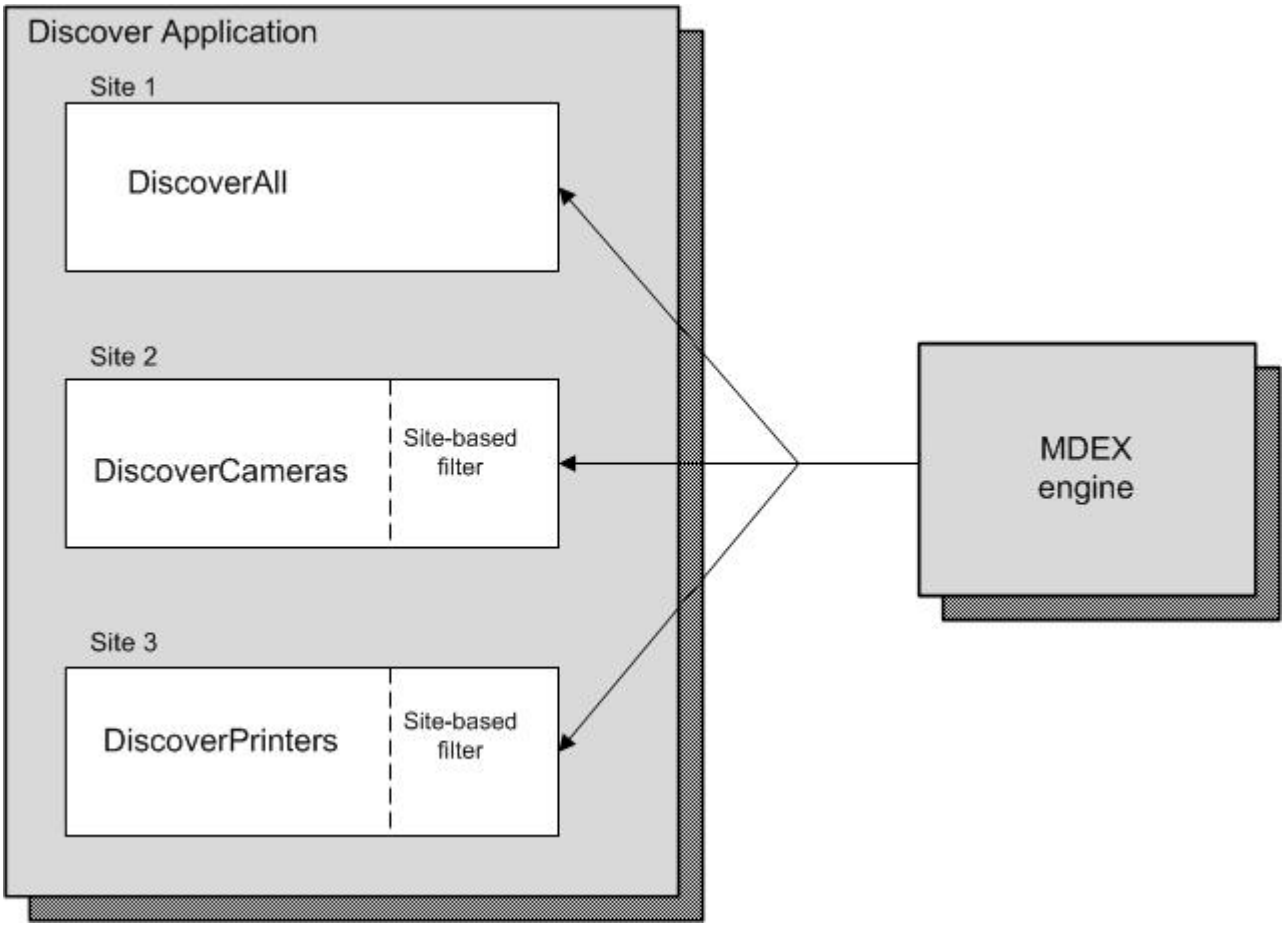
About sites

Oracle Commerce Experience Manager supports applications that can contain multiple web sites using a single MDEX Engine. Each web site in the application can have its own unique set of pages and data. These sites are sometimes known as storefronts or webshops.

Within an application, you can create sites. These sites are collections of content and features that an end user experiences through a unique base URL. This URL might contain a unique domain or a unique path within a domain. Site filters can be used to limit the display of data to a subset of the index for a particular site. Business users can create and manage specific pages for a site with Experience Manager while also sharing elements of the experience.

Note: If you are using Oracle Commerce Guided Search, applications have one site by default, and Oracle does not support adding more sites.

The following diagram shows an application that contains three web sites. Two of the sites, DiscoverCameras and DiscoverPrinters, use site filters that let the site display only those records that are relevant for the site.



Site definitions

An application in Oracle Commerce Experience Manager must have at least one site, and every site in the application must have its own site definition.

In a multiple site application, the properties in a site definition determine site-specific behavior. The site definition can be a JSON file located in the `/<app_name>/config/import/pages/<site_ID>` directory for your application. This JSON representation of a site definition is what you see in a deployment template or if you were to export the site after it is deployed. You can modify a site definition in this form and import it to update the JCR's site definition node. For example, a Discover Electronics application might have a site definition in the `/Discover/config/import/pages/DiscoverElectronics` directory.

```
{
    "ecr:type":"site-home",
    "urlPattern" : "",
    "displayName":"DiscoverElectronics",
    "description" : ""
}
```

Note that all properties in JSON files including the site definition must be in UTF-8 character format. Convert all Unicode characters to JSON encoded values. The properties of this file are described in the next section.

Site definition properties

Property	Description
urlPattern	The pattern used to match the request URL to this site. A pattern can be a domain, like example.com, or a URI, like /DiscoverPrinters. Specify multiple patterns delimited by commas. For example `/DiscoverPrint¬ers, /Printers, example.com`. See the following section for more information about URL pattern matching.
displayName	The site name that appears in Experience Manager. This is the only required property.
description	The site description that appears in Experience Manager.
previewUrl	The fully qualified preview URL for the site.
linkServiceUrl	The fully qualified URL for the site's preview link service.

URL pattern matching in the site definition.

The URL pattern is the pattern used to match the request URL to this site definition. The value defined for this property is based on the scheme used by the application to encode sites into the URL. When attempting to identify a site from a request, this pattern is matched against the URL. For example, you might specify "/DiscoverCamera" for the Discover Camera site definition.

If you enter a domain as a value in the `urlPattern` property, follow these rules:

- Domains are not case-sensitive.
- Wildcard characters are not supported.
- The first character in the domain cannot be a slash (/), for example, `example.com` is a valid domain.
- Domains do not include the schema, so pattern matching begins after the schema: `http://` or `https://`.
- If you do not use a standard port, then a port number is required. The standard port for *http* is *80*; and the standard port for *https* is *443*.

If you enter a URI as a value in the `urlPattern` property, the value matches the pattern between the context path and the page's content URI of the incoming request. URL values must follow these rules:

- The URI is case-sensitive.
- Wildcard characters are not supported.
- The first character of the pattern *must* be a slash (/), for example, `/DiscoverPrinters`.
- The pattern must match the full and exact string in the request.

A domain pattern always has precedence over a URI pattern. For example, if you have two sites where Site A uses a domain pattern and Site B uses a URI pattern, Site A always displays if that domain pattern is entered in the address.

Site-based filters

Site-based filters allow you to segregate sites that share the same MDEX engine. If only a subset of records within the MDEX engine are relevant for a specific site in your application, then you can specify site-based

filters. If you don't use filters for your sites then all sites share an identical dataset and only the presentation is different.

The site-based filter is applied to all queries performed on the site. The filter that is applied is determined by the current site. For example, the Discover Electronics application could have two additional sites: DiscoverCameras and DiscoverPrinters. DiscoverCameras is a site where end users can buy cameras, while DiscoverPrinters is a site where end users can buy printers. You can configure site-based filters that determine which records are appropriate for each site. Even though the shared index has both cameras and printers, a shopper only sees products that are relevant for the site they are visiting.

Site-based filters are defined in the `filterState.xml` file. Each site that requires site-based filtering should have a `filterState.xml`. You can define record filters in the `filterState.xml` file. Record filter syntax is described in the *MDEX Engine Development Guide*. The Discover Electronics reference application could have a site-based filter in the following location: `<app_name>/config/import/pages/<site_ID>/filterState.xml`. Site filtering does not allow configuring navigation filters and EQL filters.

Example of filterState.xml file

The following `filterState.xml` file shows a configuration with a record filter that only includes records in the camera category.

```
<Item class="com.endeca.infront.navigation.model.FilterState" xmlns="http://endeca.com/schema/xavia/2010">
   <Property name=recordFilters">
       <List>
         <String>product.category:cameras</String>
       </List>
   </Property>
</Item>
```

Adding a site to an existing application

You can add a site to an application that you have already deployed.

To add a site to your application, you must export the application, add the new site, and then import the updated application. Every site must have its own site definition, and if you plan on filtering records for the new site, you must configure a site-based filter as well.

1. Export the application to which you want to add a site.
   a) Navigate to the `<app dir>\control\` directory on Windows (`<app dir>/control/` directory on Unix).
   b) From the command line, export the application by entering the following command: `runcommand.<bat|sh> IFCR exportApplication <destination>`
   c) Navigate to the destination directory that you specified.
   d) Unzip the `<app name><date&time>.zip` file to an empty directory, for example: `\myexports\discover`.
2. Add the default site ID, site node and site definition for you new site.
   a) Navigate to the pages node of your unzipped exported application, for example, `\myexports\discover\pages`.
   b) Open the `_.json` file in an editor and add a default site ID.
      For example:

      ```
      {
          "defaultSiteId": "/DiscoverElectronics",
      ```

```
    "ecr:type": "page-root"
}
```

This is the site that the Assembler uses when the request cannot be matched to any site using the URL patterns.

Note: All properties in JSON files must be in UTF-8 character format. Convert all Unicode characters to JSON encoded values.

c) Navigate to the `\myexports\discover\pages` directory. Note that the application already has an existing site. For example, the Discover application has an existing site with *DiscoverElectronics* as the `site_ID`.

d) Create a folder for your new site parallel to the existing site. The name that you provide for the folder becomes the `site_ID`, for example `DiscoverCameras`.

e) Navigate to the existing site in your application, for example `pages\DiscoverElectronics`.

f) Copy the site definition file, `_.json`, from the folder and paste it into your new site folder, for example `pages\DiscoverCameras\_.json`.

g) Use a text editor to update the site definition with unique information appropriate for your new site. For example, the following new site has been updated with a unique URL pattern, display name, and description.

```
{
   "ecr:type": "site-home",
   "urlPattern" : "/DiscoverCameras",
   "displayName" : "Discover Cameras",
   "description" : "This site shows you all the Cameras available at Discov¬
er.",
}
```

h) If you want the site to have its own unique icon to identify it in Experience Manager, you can replace the default site icon  with one of your own by copying an image to the same directory as the site's JSON file.

The image must be 16 by 16 pixels, and be named `siteIcon.<extension>`, for example `pages\DiscoverCameras\siteIcon.png`. The supported formats are JPG, JPEG, PNG, GIF, and TIF. Oracle recommends using the PNG format.

i) Save and close the file.

3. If you want your site to filter a subset of relevant records, you need to add a site-based filter to your site.

a) Navigate to the new site directory `pages/<new site_ID>`, for example `pages/DiscoverCameras`.

b) Create an XML file and name it `filterState.xml`. The following file filters records so that only those records in the cameras category can appear in the site.

```
<Item class="com.endeca.infront.navigation.model.FilterState"
xmlns="http://ende¬
ca.com/schema/xavia/2010">
  <Property name="recordFilters">
    <List>
      <String>product.category:cameras</String>
    </List>
  </Property>
</Item>
```

c) Save and close the file.

4. A site needs at least one page in order to display so copy and paste pages that you want from your original site to your new site.

For example, you could copy the browse folder from `pages/DiscoverElectronics` to `pages/DiscoverCameras`.

5. Import the application with the new site.
   a) Navigate to the *<app dir>*`\control\` directory on Windows (*<app dir>*`/control/` directory on Unix).
   b) From the command line, import the updated application by entering the following command:
   `runcommand.`*<bat/sh>* `IFCR importApplication <path to source>`
   For example:
   ```
   runcommand.bat IFCR importApplication c:\myexports\discover
   ```

After you have added a site to your application, you must complete the following tasks:

- If your application was built using Spring and `ActionPathProvider` was implemented for your application, then you might need to update the action paths in `assembler-context.xml` file. See *Configuring site awareness in action links* on page 58.
- Update permission settings for business users that have access to Experience Manager. You can limit the site pages that business users can modify. See *Modifying users and groups* on page 82

Configuring site awareness in action links

Action links are returned in responses such as result records, refinements, and breadcrumbs. If you add a site to a Spring-built application that has implemented the `ActionPathProvider`, you might need to update the action links in the `ActionPathProvider` of the `assembler-context.xml` file to make the action links site-specific.

The `ActionPathProvider` determines navigation and record detail action paths. The content paths that prefix navigation and record states are configured as sets of key-value pair mappings. The key-value pairs shown in the Discover reference application support multiple sites -- provided that the sites use the same path for the browse and detail pages. In a multiple site application, the `ActionPathProvider` can also provide different mappings for pages on different sites.

If your sites define the navigation and record detail pages in the same location you do not have to define a path for each site, and you can skip the following steps. However, if the sites have different paths for the browse and detail pages you can define those mappings for each site by following these steps.

1. Navigate to the Assembler context file for your application, `WEB-INF/assembler-context.xml`.
2. Open `assembler-context.xml` in an editor, and find the `ActionPathProvider` section of the file.
3. Edit the `navigationActionUriMap` mapping section for the site that you are adding to your application in the `ActionPathProvider` mappings.
   In the follow example, there is a new mapping for the DiscoverCameras site that has been added to `navi¬gationActionUriMap`.

```
<!-- navigationActionUriMap -->
     <constructor-arg index="2">
         <map>
           <entry key="^/pages/DiscoverCameras/.*$" value="/pages/DiscoverCam¬
eras/camerasbrowse" />
            <entry key="^/pages/[^/]*/mobile/detail$" value="/mobile/browse"
/>
           <entry key="^/pages/[^/]*/services/recorddetails/.*$" value="/ser¬
vices/guidedsearch" />
            <entry key="^/pages/[^/]*/detail$" value="/browse" />
```

```
            <entry key="^/services/.*$" value="/services/guidedsearch" />
        </map>
    </constructor-arg>
```

4. Edit the `recordActionUriMap` mapping section for the site that you are adding to your application in the `ActionPathProvider` mappings.

   In the follow example, there is a new mapping for the DiscoverCameras site that has been added to `recordActionUriMap`:

```
<!-- recordActionUriMap -->
    <constructor-arg index="3">
        <map>
          <entry key="^/pages/DiscoverCameras/.*$" value="/pages/DiscoverCam¬
eras/camerasdetail" />
           <entry key="^/pages/[^/]*/mobile/.*$" value="/mobile/detail" />
          <entry key="^/pages/[^/]*/services/.*$" value="/services/recordde¬
tails" />
           <entry key="^/pages/[^/]*/.*$" value="/detail" />
           <entry key="^/services/.*$" value="/recorddetails" />
        </map>
    </constructor-arg>
```

5. Save and close the file.
6. Restart the Endeca Tools Service.

In the examples in the previous steps, the `camerasdetail` page is used as the record detail page for any page on the DiscoverCameras site. Other sites in the application would use the `/mobile/detail page` for `/mobile/*` pages and the `/detail` page for all other pages.

Updating a site definition

This section describes how to update the site definition.

You can update a site definition by changing the URL pattern, the display name, or description. You can also modify the site-based filter. To update a site, you must export the application, edit the site, and then import the updated application. To remove the site altogether, you can delete the site definition, or to stop record filtering on a site, you can delete the `filterState.xml` file.

1. Export the application with the site that you want to update.
   a) Navigate to the `<app dir>\control\` directory on Windows (`<app dir>/control/` directory on UNIX).
   b) From the command line, export the application by entering the following command: `runcommand.<bat|sh> IFCR exportApplication <destination>`
   c) Navigate to the destination directory that you specified.
   d) Unzip the `<app name><date&time>.zip` file to an empty directory, for example: `\myexports\discover`.
2. Update the site definition for the site.
   a) Navigate to the `/pages/<site ID>` directory for the site in your unzipped exported application, for example `pages/DiscoverCameras`.
   b) Use an editor to modify the site definition JSON file with updated information for your site.
   c) Save and close the file.
3. If the site has a site-based filter that needs to be updated, edit the `filterState.xml` file.

4. Import the application with the modified site.
   a) Navigate to the <app dir>\control\ directory on Windows (<app dir>/control/ directory on UNIX).
   b) From the command line, import the updated application by entering the following command:
   runcommand.<*bat/sh*> IFCR importApplication <path to source>
   For example:
   ```
   runcommand.bat IFCR importApplication c:\myexports\discover
   ```

Removing a site

This section describes how to remove a site.

To remove a site from an application, you can delete the site node.

1. Export the application with the site that you want to remove.
   a) Navigate to the <app dir>\control\ directory on Windows (<app dir>/control/ directory on UNIX).
   b) From the command line, export the application by entering the following command:
   runcommand.<*bat/sh*> IFCR exportApplication <destination>
   c) Navigate to the destination directory that you specified.
   d) Unzip the <app name><date&time>.zip file to an empty directory, for example:
   c:\myexports\discover.
2. Delete the site node for the site.
   a) In your unzipped exported application, navigate to the /pages/ directory.
   b) Delete the site node, for example DiscoverCameras.
   If you delete the default site, you must update the defaultSiteID, so that it contains the site ID of one of your remaining sites. For more information on updating the defaultSiteID, see *Adding a site to an existing application* on page 56.
3. Import the application.
   a) Navigate to the <app dir>\control\ directory on Windows (<app dir>/control/ directory on UNIX).
   b) From the command line, import the updated application by entering the following command:
   runcommand.<*bat/sh*> IFCR importContent pages <path to source>
   For example:
   ```
   runcommand.bat IFCR importContent pages c:\myexports\discover\pages.
   ```

Part 2

Configuring Application Components

Chapter 5

Configuring Communication Between Components

This section discusses configuring communication between application components.

About Workbench interaction with the Endeca Configuration Repository

The Endeca Configuration Repository is a Web application that runs in the Endeca Tools Service.

The Endeca Configuration Repository uses a JSR-170-compliant Java Content Repository to store configuration related to Endeca applications. It also hosts several tools that are accessed via Workbench, including Experience Manager and the Thesaurus.

Specifying Workbench authentication credentials for the Endeca Configuration Repository

By default, Workbench authenticates itself to the Endeca Configuration Repository using the JCR repository's administrator credentials. These are the `admin` user credentials that you initially specify when you install Tools and Frameworks.

Workbench uses Oracle Wallet, a file-based credential store, to safely store administrator authentication credentials. The credential store is represented as `cwallet.sso` and is located at `ToolsAndFrameworks\<version>\server\workspace\credential_store`. The administrator credential in the store is composed of a user name and password that is uniquely identified by a map name and a credentials key combination. Therefore, one set of credentials containing a map name and a credentials key maps to one set of a username and a password pair. This map name and credentials key for the administrator user are stored as properties in the `WorkbenchConfig.xml` file.

```
<custom-bean id="csfManager" class="com.endeca.soleng.eac.toolkit.util.CSFManager">

      <property name="jpsConfigPath" value="@@JPSCONFIG_LOCATION@@" />
      <property name="mapName" value="endecaToolsAndFrameworks"/>
    </custom-bean>

    <custom-component id="IFCR" host-id="ITLHost" class="com.ende¬
ca.soleng.eac.toolkit.component.IFCRComponent">
      <properties>
        <property name="repositoryUrl" value="http://@@HOST@@:@@WORK¬
BENCH_PORT@@/ifcr" />
        <property name="numExportBackups" value="3" />
```

```
        <property name="credentialsKey" value="ifcr"/>
      </properties>
      <custom-bean ref="csfManager"/>
    </custom-component>
```

To specify the Workbench authentication credentials for the Endeca Configuration Repository:

1. Update the administrator credentials used by Deployment Template scripts to connect to the Endeca Configuration Repository. Workbench stores these credentials in the Oracle Wallet credentials store.
   a) Navigate to the *<installation path>*\ToolsAndFrameworks*<version>*\credential_store\bin directory.
   b) Run the manage_credentials script using this format

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

Flags	Description
config	Path to the jps-config.xml. This file defines this instance of the credentials store. The default value is ..\..\server\workspace\credential_store\jps-config.xml.
mapName	Must match the mapName specified in the config\script\WorkbenchConfig.xml file. The default value is *endecaToolsAndFrameworks*.
user	The administrator user name. The default value is *admin*.
key	The key name that can be of your choosing. The value must match the credentialsKey value specified in the config\script\WorkbenchConfig.xml file.
type	*Password* is the default.

   For example,

   ```
   manage_credentials.bat add --user admin --key ifcr
   ```

   The config, mapName, and type are not included because they are the default values.

   c) Since the credential already exists, you are prompted to replace it. Type yes.
   d) When you are prompted for your new password, enter and confirm the password that you want to use. The credentials store is updated with the new administrator password. You still need to update it in the Workbench. See *Changing the administrator's passwordFollow these steps to change the password for the predefined admin user.*

2. Change the administrator password in the Endeca Configuration Repository by submitting a POST request to http://<WorkbenchHost>:8006/ifcr/system/userManager/user/admin.changePassword.html with the following parameters:

Parameter	Value
oldPwd	The current password.
newPwd	The value for the new password.
newPwdConfirm	The value for the new password.

The following is an example using the curl tool:

```
curl -FoldPwd=admin -FnewPwd=newpassword -FnewPwdConfirm=newpassword \
http://admin:admin@localhost:8006/ifcr/system/userManager/user/admin.changePass¬
word.html
```

3. Update the credentials that Workbench uses to connect to the Endeca Configuration Repository.
   a) Stop the Endeca Tools Service.
   b) Navigate to `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF\conf` (on UNIX).
   c) Open the `webstudio.properties` file.
   d) Locate the `ifcr.password` property, for example:

```
# The password Workbench uses to authenticate as the admin user for the Endeca
 Configuration Repository
#ifcr.password=admin
```

   e) Uncomment the line and set the value of the property to the new password, for example:

```
ifcr.password=newpassword
```

   f) Save and close the `webstudio.properties` file.
   g) Start the Endeca Tools Service.

For further information about Sling user management, consult the Apache documentation.

About Workbench interaction with the MDEX Engine

Workbench both queries the MDEX Engine for information and publishes configuration to it.

Experience Manager queries the MDEX Engine for record and dimension information that a content administrator can use to configure dynamic content. Examples include:

- specifying a navigation state as part of a location trigger
- selecting records or a navigation state for content spotlighting
- selecting records or dimension values for boost and bury

Workbench publishes configuration to the MDEX Engine, including:

- Rules
- Phrases
- Keyword redirects
- Thesaurus entries

When you initially deploy a new application, an environment is created with two Dgraphs running on the same MDEX Engine server. One Dgraph is for the authoring application and the other Dgraph is for the live application. You can optionally change this configuration and add one or more MDEX Engine servers, with one or more Dgraphs, to either the authoring or live application. For more information on changing this configuration, see *Adding MDEX Engine servers and Dgraphs to the servers* on page 47.

About accessing files on remote servers

Experience Manager occasionally needs to access files hosted on a different server. Certain security issues may apply.

Experience Manager makes an anonymous request to the file server to fetch resources. That is, even though content administrators are authenticated when they log in to Experience Manager, the tool does not use their credentials when requesting images, editors, or other files.

About cross-domain policy files

Experience Manager respects the cross-domain policy file of a server hosting external resources. This file enables access to the server from a specified IP address or domain, or from any domain. If the policy file does not allow access from the Experience Manager server, a security error similar to the following displays when Experience Manager attempts to load the resource:

```
Error #2044: Unhandled securityError:. text=Error #2048: Security sandbox viola¬
tion: http://pagebuilder.mycompany.com/tmgr/tmgr.swf cannot load data from
http://www.example.com/images/3column.gif.
```

The following example `crossdomain.xml` file enables access to files hosted on www.example.com, from any domain:

```
<?xml version="1.0"?>
<!-- http://www.example.com/crossdomain.xml -->
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
```

You can also restrict access to specific domains or IP addresses; for instance, for the server on which Experience Manager is running. Wildcards are allowed in domain names, but not IP addresses.

The following example shows a policy file for www.example.com that allows access from anywhere in the example.com domain, www.customer.com, and 105.216.0.40. It includes a `by-content-type` meta-policy that allows policy files with a `Content-Type` of exactly `text/x-cross-domain-policy`:

```
<?xml version="1.0"?>
<!-- http://www.example.com/crossdomain.xml -->
<cross-domain-policy>
  <site-control permitted-cross-domain-policies="by-content-type"/>
  <allow-access-from domain="*.example.com" />
  <allow-access-from domain="www.customer.com" />
  <allow-access-from domain="105.216.0.40" />
</cross-domain-policy>
```

Important: In addition to cross-domain policy files, you must set up a meta-policy for each server. These are configuration settings that manage what cross-domain policies are allowed on a server. A meta-policy file is required with the use of Flash 10.

For more information about meta-policies and cross-domain policy files, see the Adobe Flash documentation.

Setting up a cross-domain policy file

By default, MDEX Web services are accessible from Experience Manager and other Workbench tools only if the MDEX Engine and Workbench are hosted on the same domain.

For example, if Workbench is hosted on `apps.example.com`, the MDEX must also be accessible at `apps.example.com`, and Experience Manager must be configured to access the MDEX Engine at `"apps.example.com"`. Using the host's IP address or an alias hostname, such as `"localhost"` causes a "Security Error" alert box to appear in Experience Manager when an editor attempts to access the MDEX Engine.

If the MDEX Engine is hosted on a different domain from Workbench, you must set up a cross-domain policy file on the MDEX Engine server. These steps apply to any Flex client application that communicates with an MDEX Engine via Web services.

To configure cross-domain access to MDEX Web services from a Flex client:

1. Navigate to the `/conf/dtd/xform` directory of your MDEX Engine installation, for example:
   `C:\Endeca\MDEX\6.4.0\conf\dtd\xform`
2. Create an Adobe Flash cross-domain policy file, `crossdomain.xml`.
3. Configure your `crossdomain.xml` file to grant access to all domains hosting instances of Workbench.
   An example is provided below:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.macromedia.com/xml/dtds/cross-
domain-policy.dtd">
<cross-domain-policy>
  <allow-access-from domain="*.example.com" />
  <allow-http-request-headers-from domain="*" headers="SOAPAction" />
</cross-domain-policy>
```

   - The `<allow-access-from>` element grants access to the local MDEX Web service from a set of domains. The `domain` attribute may be specific, or may include a wildcard, as shown above. You can include any number of `<allow-access-from>` elements, each for a different domain.
   - The `<allow-http-request-headers-from>` element as specified above is required. It enables Flash clients to communicate with the MDEX using the SOAP protocol.

For a complete specification of the cross-domain policy file format, please see the Adobe documentation at *http://www.adobe.com/devnet/articles/crossdomain_policy_file_spec.html*.

Chapter 6

Configuring the Assembler

The Assembler implementation provided with Tools and Frameworks is confgured through Spring. While most configuration is left to the Application Developer, administrative users may choose to add configuration that enables the Assembler Administrative Servlet for an associated application. For a full overview of configurable properties in the default Assembler implementation, see "Assembler Configuration" in the *Assembler Application Developer's Guide*.

Configuring the Assembler administrative servlet

The administrative servlet classes are integrated with the Assembler. To access the servlet in an application, you must configure the servlet path and class in the application's `web.xml` deployment descriptor file and define the service in the Assembler context configuration file.

To configure the Assembler administrative servlet:

1. Stop your Web application container.
   For Guided Search reference applications, this is controlled through the Endeca Tools Service.
2. Navigate to the `WEB-INF\web.xml` file of your application.
3. Open the file in a text editor.
4. Define the admin servlet, referencing the `com.endeca.infront.assembler.servlet.spring.admin.SpringAdminServlet` class and setting a value for the `adminServiceId`:

```
<servlet>
    <servlet-name>admin</servlet-name>
   <servlet-class>com.endeca.infront.assembler.servlet.spring.admin.SpringAdminServlet</servlet-class>
    <init-param>
        <param-name>adminServiceId</param-name>
        <param-value>adminService</param-value>
    </init-param>
</servlet>
```

   Note: The value of the `adminServiceId` parameter corresponds to the JavaBean ID in the Assembler context file.

5. Create the servlet mapping, using the servlet you defined in Step 4:

```
<servlet-mapping>
    <servlet-name>admin</servlet-name>
```

```
    <url-pattern>/servlet/admin/*</url-pattern>
</servlet-mapping>
```

6. Optionally, configure the HTTP request as the root event for performance logging.

   Map a `com.endeca.infront.assembler.perf.PerfEventFilter` filter to the HTTP request:

```
<!-- The PerfFilter must go after the RequestContextFilter so it can have
access to the request -->
<filter>
    <filter-name>PerfFilter</filter-name>
    <filter-class>com.endeca.infront.assembler.perf.PerfEventFilter</filter-
class>
</filter>
<filter-mapping>
    <filter-name>PerfFilter</filter-name>
    <url-pattern>*</url-pattern>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

   This causes the performance statistics page to report the latency for all operations on the page, instead of just those included in an `assemble()` call.

7. Save and close the file.
8. Open the Assembler context file for your application.
   For example,
   `C:\Endeca\ToolsAndFrameworks\<version>\reference\discover-electronics\WEB-INF\assembler-context.xml`
9. Define a administrative service bean of class `com.endeca.infront.assembler.servlet.admin.AdministrationService`:
   a) Set the `id` attribute to the value of the `adminServiceId` parameter set in Step 4:

```
<bean id="adminService" class="com.endeca.infront.assembler.servlet.admin.Ad¬
ministrationService">
</bean>
```

   b) Set the `storeFactory` `<property>` as shown below:

```
<bean id="adminService" class="com.endeca.infront.assembler.servlet.admin.Ad¬
ministrationService">
    <property name="storeFactory" ref="${store.factory}"/>
</bean>
```

      This property is used to retrieve the registered `FileStoreFactory` for the servlet's `op=update` operation.

10. Save and close the file.
11. Restart your Web application container.

Chapter 7

Configuring Workbench

This section describes how to perform basic configuration and administrative tasks for Workbench, including managing users, configuring locales and extensions, and integrating Workbench with LDAP, SSL, and Single Sign-On.

The Workbench session cookie

Oracle Commerce Workbench uses a `ORA_OC_WBSESSION` cookie to maintain a user's session.

In rare cases it is possible for the cookie name to collide with a cookie of the same name on the same application server. This conflict can occur if you are running your application on an application server on the same host as Workbench and using `ORA_OC_WBSESSION` for two purposes. In this situation, a user may have their session unexpectedly terminated. To resolve this issue, you can either run the application on another host (that is, a host other than the one Workbench is on), or customize your application server to use a different cookie name (other than `ORA_OC_WBSESSION`) through custom directives on the specific application server.

Configuring Workbench locales

The section describe how to configure Workbench customizations and reports for locales.

About configuring Workbench for locales

By configuring the locale, you can change the language and country settings for custom Workbench menus and extensions, as well as reports.

You can configure Workbench to display in the following supported locales:

Locale	ISO Language Code
Chinese (Simplified)	zh_CN
Chinese (Traditional)	zh_TW
Czech	cs
Danish	da
Dutch (The Netherlands)	nl

Locale	ISO Language Code
English (United States)	en_US
English (United Kingdom)	en_UK
English (Australia)	en_AU
English (Canada)	en_CA
French (France)	fr
Finnish	fi
German	de
Greek	el
Hungarian	hu
Italian	it
Japanese	ja
Korean	ko
Polish	pl
Portuguese (Portugal)	pt_PT
Portuguese (Brazil)	pt_BR
Russian	ru
Spanish	es
Swedish	sv
Thai	th
Turkish	tr

Locales for custom extensions and menus

If your implementation supports multiple locales, you can localize your custom Workbench extensions and menus.

Workbench provides resource property files for each locale for storing localized strings. These files are located in `%ENDECA_TOOLS_CONF%\conf\locale` (on Windows) or `$ENDECA_TOOLS_CONF/conf/locale` (on UNIX). Each locale has a uniquely named resource property file: `Resources_<locale>.properties` where `<locale>` is the ISO language code. For example `Resources_fr.properties` indicates that French values are stored in it.

You can store localized values for the following attributes :

- Extension names
- Extension descriptions
- Extension icons
- Menu nodes
- Menu node descriptions
- Menu node icons

Values that do not change for locale (URLs or product names for example) are specified in the single `Resources.properties` file.

Localizing Workbench extensions

You localize custom Workbench extensions by adding localized strings into the resource property file for each locale that you support..

Before you begin, add the custom extension to the `ws-extensions.xml`. The `defaultName`, and `defaultDescription` attributes are not required if the names and descriptions are included in the localized resource files.

```
<?xml version="1.0" encoding="UTF-8"?>
<extensions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="extensions.xsd">
  <extension id="productHome"
       url="http://www.example.com" />
  </extensions>
```

Follow these steps to add localized properties to the resource property file for each locale :

1. Stop the Endeca Tools Service.
2. Navigate to %ENDECA_TOOLS_CONF%\conf\locales\ (on Windows) or $ENDECA_TOOLS_CONF/conf/locales/ (on UNIX).
3. Open `Resources_<locale>.properties` in a text editor for the locale in which you want to add localized values. For example, `Resources_en_US.properties` is the file name for English (US) values.
4. Add the following attributes and localized strings for your custom extension. The extension ID is the same one used in the `ws_extensions.xml` file.

Attribute	Value
`tool.<extension_ID>.name =`	The localized display name for this extension that appears in the navigation menu and launch page in Oracle Commerce Workbench.
`tool.<extension_ID>.description =`	A brief localized description of this extension that appears on the launch page in Oracle Commerce Workbench.
`tool.<extension_ID>.icon =`	An absolute or relative URL to a custom image for this extension's entry on the launch page. Relative URLs are relative to **<**hostname**>**:8006. This attribute is only required if the icon is unique for each locale, otherwise, add this attribute to the `Resources.properties` file or specify a value for the `defaultIcon` in the `ws_extensions.xml` file.

For example:

```
# Product Home Extension
tool.productHome.name = Product Home Page
tool.productHome.description = Start the product here.
```

5. Save and close the file.
6. Start the Endeca Tools Service

Localizing menu nodes

If you customize a menu for multiple locales in Workbench, you can optionally specify localized titles for custom menu nodes in a resources property file.

Before you begin, add the custom menu node to the `ws-mainMenu.xml`. The `defaultName` attribute is not required if the name is included in the localized resource files.

```
<?xml version="1.0" encoding="UTF-8"?>
<mainmenu xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mainMenu.xsd">
<menunode id="myextensions">
        <menuitem id="productHome"/>
</menunode>
</mainmenu>
```

Follow these steps to add localized properties to the resource property file for each locale:

1. Stop the Endeca Tools Service.
2. Navigate to `%ENDECA_TOOLS_CONF%\conf\locales\` (on Windows) or `$ENDECA_TOOLS_CONF/conf/locales/` (on UNIX).
3. Open `Resources_<locale>.properties` in a text editor for the locale in which you want to add localized values. For example, `Resources_en_US.properties` is the file name for English (US) values.
4. Add the following attributes and localized strings for your custom menu node. The menu node ID is the same one used in the `ws_mainMenu.xml` file.

Attribute	Value
menu.<menunode_ID>.name =	The localized display name for this node that appears in the navigation menu and launch page. This attribute is required for all custom nodes.
menu.<menunode_ID>.description =	The localized description for this node that appears in the navigation menu and launch page. This attribute is not required.
menu.<menunode_ID>.icon =	An absolute or relative URL to a localized custom image for this menu node on the launch page. Relative URLs are relative to `<hostname>:8006`. This attribute is only required if the icon is unique for each locale, otherwise, add this attribute to the `Resources.properties` file or specify a value for the `defaultIcon` in the `ws_mainMenu.xml` file.

   For example:

```
#My Extensions Tools Menu Properties
menu.myextensions.name = Custom Tools
menu.myextensions.description = Launch your custom tools.
```

5. Save and close the file.
6. Start the Endeca Tools Service.

Configuring the Report Scheduler for locales

If you want to use the Report Scheduler to run reports on use patterns at your site, you must configure it to display your daily and weekly reports in the language for your locale. The default locale of the Report Scheduler is the locale of the JVM on which the reports run.

Follow these steps to configure a reporting locale.

1. Navigate to `C:\Endeca\Apps\<appname>\config\script` (on Windows) or `/Endeca/Apps/<appname>/confige/script` (on UNIX).
2. Open `ReportGeneration.xml` in a text editor.
3. Find the `Workbench XML Report Generators` section.
4. Add the following element as a child of the **<**java options**>** for each report generator:

   `<java-option>-Duser.language=`*xx*`</java-option>`

   where *xx* is the language code for that locale.

   The following example specifies `fr` for French.

```
<java-options>
            <java-option>-Xmx1G</java-option>
            <java-option>-Duser.language=fr</java-option>
</java-options>
```

5. Save and close the file.
6. In Workbench, open the EAC Admin Console.
7. Click the **Scripts** tab.
8. Run the reports that you have updated with a new locale.

Configuring Workbench logs

You can control the Workbench log level, the maximum file size, and the number of files in the log rotation. You can also optionally direct the output of any Workbench logger to the console or to another file.

Both the Workbench system log and audit log are configured in the `webstudio.log4j.properties` file, located in `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF/conf` (on UNIX).

To configure Workbench logs:

1. Stop the Endeca Tools Service.
2. Navigate to `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF/conf` (on UNIX).
3. Open the `webstudio.log4j.properties` file.
4. Modify the configuration file as needed.

   For more information, see the comments in `webstudio.log4j.properties` and the log4j documentation at *http://logging.apache.org/log4j/*.
5. Save and close the file.
6. Start the Endeca Tools Service.

Managing Users in Workbench

This section describes the Workbench users and permissions model, and how to manage users and groups within Workbench.

About users and permissions in Workbench

Workbench users and permissions are defined by a Workbench administrator.

Workbench users log in with standard user name and password authentication, and permissions dictate which Workbench tools and content within an application are available to them. Workbench administrators create accounts for users that define this authentication and permission information. Administrators can also create groups and then add users to the groups. Creating groups is the preferred method of managing user permissions. For example, you can place all users that need access to a specific content collection into a group. By granting access to the group rather than to each user, you avoid the need to assign permissions to each user one-by-one. You can also make groups members of other groups which further aids in assigning permissions.

Once groups and users are added to Workbench, their names and passwords are associated with all applications across Workbench. Permissions, on the other hand, are associated with single applications, and must be specified for each application in Workbench.

Within an application, administrators provide permissions at the tool level or at a more granular content level. For example, you might provide a group with permissions to access the Experience Manager tool. This high-level access provides group members with write access to all the content within the Experience Manager. The administrator can also limit access to specific content folders within Experience Manager.

User management and LDAP

LDAP, Lightweight Directory Access Protocol, is a centralized directory used by programs to look up user information. Using LDAP, one password for a user can be shared between many services. If you have Workbench configured to use LDAP for user authentication, an administrator can create a member profile where the password and identity information is stored and managed in an LDAP directory. LDAP integration also allows you to assign permissions across an entire LDAP group rather than configuring each user individually. For more information about configuring Workbench with LDAP, see *Integrating LDAP with Oracle Commerce Workbench*.

User management and Oracle Commerce Single Sign-On

Commerce SSO integrates authentication for Workbench and Oracle Commerce Business Control Center, allowing a user to switch between tools without encountering additional login screens. If your deployment has enabled Commerce SSO, then an administrator can create a member or group profile where the password and identity information is stored and managed in the Oracle Commerce Platform internal profile repository. For more information about configuring Workbench with Commerce SSO, see *Integrating Workbench with Oracle Commerce Single Sign-On* on page 106

Special characters

User and group names cannot contain the following characters: `/ \ : [ ] | * ? " < >`.

Logging in to Workbench as an administrator

After installation, Workbench has a predefined administrator user with full administration privileges.

An administrator is granted all permissions in the system. The predefined Workbench administrator uses the username `admin` with a password that you specify during installation. After logging in to Workbench you can modify the password, but not the user name.

The `admin` user can create additional users and administrators in Workbench (only an administrator can create other administrators). An administrator can also delete other administrators, but not the predefined administrator. If you have LDAP authentication enabled, see the section *About granting administrator privileges in Workbench to LDAP users and groups* for additional information on creating Workbench users.

To log in to Workbench as an administrator:

1. In a Web browser, navigate to the Workbench login page.
   By default, this is `http://localhost:8006`.
2. Specify the username and password.
3. Click **Log In**.

After your initial login, you can change the password of the predefined `admin` user or create and manage users and administrators.

Changing the administrator's password

Follow these steps to change the password for the predefined `admin` user.

You must also change the administrator's password in the Workbench and the Oracle Wallet for the new password to take effect. You cannot use this procedure to change the password of any Oracle Workbench administrators that are in the Commerce or LDAP repositories.

1. In the upper right corner of Oracle Commerce Workbench, click the down arrow next to **admin** and then click **Change Password**.
2. Enter the current admin password in the **Old Password** field.
   If this is the first time that you are changing the admin password, enter the password that you specified when you installed Tools and Frameworks.
3. Enter your new password in the **New Password** and **Confirm Password** fields.
4. Click **Save**.
5. Change the administrator password stored on disk. See *Specifying Workbench authentication credentials for the Endeca Configuration Repository* on page 63.

Best practices

Oracle recommends the following best practices for managing users in the Workbench.

- Consider adding all users to groups to make managing permissions simpler.
- Do not share the predefined admin user account. This makes it difficult to track who has made changes to Workbench. Create an account for each user or group that administers Workbench.
- Do not let business users share accounts. Again, this makes it difficult to track changes in Workbench.
- Create administrators in addition to the predefined admin. If one administrator loses a password, another administrator needs to reset it.
- If you use LDAP, consider creating an LDAP group of administrators to add to the Administrators group in Workbench.

Adding administrators to Workbench

Administrators can configure other administrators on the **User Management** page.

You add administrators to the Workbench by adding a users or groups to the administrators group.

1. From Administrative Tools, click **User Management**.
2. Click the **Users** or **Group** tab, and then click **Add User** or **Add Group**.
3. Select a **Source**. If your site uses Commerce SSO, then **Commerce** is already selected.
4. Enter the name of the user or group that you want to add in the **User ID** or **Group ID** field.
5. If your **Source** is LDAP, click **Validate** to determine if you have entered a valid LDAP user name.

    **Note: Validate** does not display if your **Source** is Workbench or if you have Commerce SSO enabled.

   If you entered a valid LDAP user or group name, then Workbench retrieves available information and populates the name and email fields. This information as well as the user password is not editable.
6. If your **Source** is Workbench, complete the following fields.

   For a single user:

   - First Name
   - Last Name
   - Email
   - Password
   - Confirm Password

   For a group:

   - Name
   - Email
7. If your **Source** is Commerce, click **Validate** to determine if you have entered a valid Commerce user name. If you entered a valid Commerce user or group name, then Workbench retrieves available information from the Oracle Commerce Platform internal user repository and populates the name, email, and locale fields. Only the locale field is editable.
8. Select **Administrators** from the **Select a group** drop-down, and click the **Include in Group** button  . You do not need to give this new administrator any additional permission since administrators have all available permissions already.
9. Click **OK**.

Adding groups to Workbench

Administrators and administrative users with permissions can configure groups on the **User Management** page.

You can add a group in one of these ways:

- Create a group and assign permissions to the group.
- Add a group that is stored in LDAP and assign permissions to the group.
- Add a group that is stored in the Oracle Commerce Platform internal profile repository if Commerce SSO is enabled.

The LDAP options are only available if you have configured Oracle Workbench to use LDAP for user authentication. For more information about using Workbench with LDAP, see the *Integrating LDAP with Oracle Commerce Workbench*.

To add a group to Workbench:

1. From Administrative Tools, click **User Management**.
2. Click the **Groups** tab, and then click **Add Group**.
   The **Create Group** dialog displays.
3. Select a **Source**. If your site uses LDAP, then **LDAP** is already selected; if not, there is no **Source** to select.
4. Enter a name in the **Group ID** field.
5. If your **Source** is LDAP, click **Validate** to determine if you have entered a valid LDAP group name.

   
   Note: Validate does not display if your **Source** is Workbench.

   If you entered a valid LDAP group name, then Workbench retrieves available group information and populates the name and email fields. This information is not editable.
6. If your **Source** is Workbench, complete the following optional fields:
   - Name
   - Email
7. If your **Source** is Commerce, click **Validate** to determine if you have entered a valid Commerce organization name. Organization names are the Oracle Commerce Platform equivalents of Workbench group IDs.
   If you entered a valid Commerce organization name, then Workbench retrieves available information from the Oracle Commerce Platform internal user repository and populates email field. The email field is not editable. You can enter an optional name in the **Name** field. This name does not synch with any fields in the internal user repository.
8. If you want to make this group a member of another group, select a group in which to add the group from the **Select a group** drop-down, and click the **Include in Group** button  . Repeat this for each group in which you want to add the group as a subgroup.
   The group is granted any permissions of the group in which it is a subgroup.

   Permissions are cumulative. For example, if the new group belongs to two groups: group A and group B, and group A has permission to use a tool and group B does not, then the new group has access to the tool due to his or her membership in group A.
9. To populate this group with users and groups, click the **Membership** tab.
   a) Select a user or group from the **Include a user or group...** drop-down list, and then click the **Include in Group** button  .
      The user or group displays in the list.
   b) Repeat the first step until you have added all the members that you want to the group.
10. If you want to give the group additional permission to use tools or to access content that group membership does not provide, then follow these steps. You cannot use this procedure to remove permissions that a group is granted by membership in another group.
    a) Click the **Permissions** tab.
    b) Select the application in which the group will be working.
       Any tools that the group's parent group grants permissions for are already selected.
    c) Select the additional tools to which you want to give the group access.
       Giving the group access gives it full access: the group members can read, write and edit.
    d) If you selected Experience Manager in the previous step, or if it was already selected, you can update the content that the group can access. Expand **Experience Manager**.
       Any content that the group's parent group grants permissions for are already selected. The group's members have read access to any unselected content.

e) If you want to limit the sites within an application that a group can access, then deselect the **Site Page** nodes for those sites.
f) Select or deselect the pages, folders, and content folders to which you want to update group access.

Giving the group access gives it full access: the group members can read, write, and edit. Content permissions are inherited, so a user has access to any child folders of a configured content folder.

Note: Deselecting sites, pages, folders or content collections removes write access. The group still has read access.

11. Click **Save** .

The new group profile displays on the **Groups** tab of the **User Management** page.

Adding business users to Workbench

Administrators and administrative users with permissions can configure users on the **User Management** page.

You can add a user in one of these ways:

- Add a user manually in Oracle Commerce Workbench and assign permissions.
- Add a user that is stored in LDAP and assign permissions.

  If a user is a member of an LDAP that has been added to Workbench, the user is automatically added the first time that he or she logs in to Workbench.

- Add a user that is stored in the Oracle Commerce Platform internal user repository if Commerce SSO is enabled.

  If a user is a member of an Oracle Commerce Platform organization that has been added to Workbench as a group, then the user is automatically added the first time that he or she logs in to Workbench.

The LDAP options are only available if you have configured Oracle Workbench to use LDAP for user authentication. For more information about using Oracle Commerce Workbench with LDAP, see the *Integrating LDAP with Oracle Commerce Workbench*.

To add a user to Oracle Commerce Workbench:

1. From Administrative Tools, click **User Management**.
2. Click the **Users** tab, and then click **Add User**.
   The **Create User** dialog displays.
3. Select a **Source**. If your site uses Commerce SSO, then **Commerce** is already selected.
4. Enter a name in the **User ID** field.
5. If your **Source** is LDAP, click **Validate** to determine if you have entered a valid LDAP user name.

   Note: Validate does not display if your **Source** is Workbench or if you have Commerce SSO enabled.

   If you entered a valid LDAP user name, then Workbench retrieves available user information and populates the name and email fields. This information and the user password is not editable.
6. If your **Source** is Workbench, complete the following fields:

   - First Name
   - Last Name
   - Email
   - Password

- Confirm Password

7. If your **Source** is Commerce, click **Validate** to determine if you have entered a valid Commerce user name. If you entered a valid Commerce user or group name, then Workbench retrieves available information from the Oracle Commerce Platform internal user repository and populates the name, email, and locale fields. Only the locale is editable.
8. If you want to change the locale from the one retrieved from the Oracle Commerce Platform internal user repository, select a different one for the user from the **Locale** drop-down.

   If you select **System default**, then Workbench checks for a locale that Workbench supports in this order:

   1. If the user has specified a supported preferred locale in his or her browser, then that locale is used.
   2. If no preferred locale is supported, then Workbench uses the locale specified in the `webstudio.properties` file for this user.

9. Select a group in which to add the user from the **Select a group** drop-down, and click the **Include in Group** button . Repeat this for each group in which you want to add the user.

   Note: You cannot add an LDAP user to an LDAP group.

   The user inherits the permissions of a group in which it is a member.

   Permissions are cumulative. A user who is a member of groups defined in Workbench is assigned the broadest permission associated with any of the groups to which that user belongs. For example, if the user belongs to two groups: group A has permission to use Experience Manager and group B does not, then the user has access to Experience Manager due to his or her membership in group A.

10. If you want to give the user additional permission to use tools or to access content in your application that group membership does not provide, or if the user is not a member of any group, then follow these steps. You cannot use this procedure to remove permissions that a user is granted by membership in a group because permissions are additive. These permissions display in grey to indicate they cannot be removed.
    a) Click the **Permissions** tab.
    b) Select the application in which the user will be working.
    Any tools for which the user's group membership grants permissions are already selected.
    c) Select the additional tools to which you want to give the user access.
    d) If you selected **Experience Manager** in the previous step, or if it was already selected, you can update the content that the user can access. Expand **Experience Manager**.
    e) If you want to limit the sites within an application that a user can access, then deselect the **Site Page** nodes for those sites.
    f) Select or deselect the site pages, folders, and content folders to which you want to update user access.

    Giving users access gives them full access: the user can read, write, and edit. Content permissions are inherited, so a content collection has the same permissions as the folder to which it belongs. You can also add permissions explicitly if the folder has no permissions.

    Note: Deselecting sites, pages, folders or content collections removes write access. The user still has read access.

11. Click **Save** .

The new profile displays on the **Users** tab of the **User Management** page.

Modifying users and groups

Administrators and administrative users with permission can modify aspects of a user or group including password, identity information, and permissions. You cannot change password and identity information for Commerce or LDAP users and groups.

You cannot change the user or group name. To change a user or group name, create a new member with the new name and the same permissions, then delete the existing user or group.

See *Changing a group's membership* to add or remove members from a group.

To modify a user or group:

1. From the **User** tab or **Group** tab on the **User Management** page, click the user or group name of the user or group whose profile you want to modify.
2. Modify any first or last name, group name, email, and password information that is required.
   You cannot update this information for Commerce or LDAP users and groups.
3. Optionally, select a group to which you want to add the user or group and then click the **Include in Group** button .
4. Click the **Permissions** tab and update user or group access to tools and content.
   You cannot remove permissions if the user or group inherits the permission from a parent group. You can, however, add permissions.
5. Click **Save**.

If you want to add or remove members from a group, see *Changing a group's membership*.

Changing a group's membership

You can add and remove members from a group with the following exceptions: you cannot add and remove LDAP members from an LDAP group or Commerce members from a Commerce group.

When you add members to an existing group, the members are granted all the permissions that the group has. If a member belongs to multiple groups that have different permissions, then the member has the broadest permissions. If you remove members from a group, be sure that the members have all the necessary permissions that they need either as an individual user or as a member of another group.

To add and remove members:

1. From Administrative Tools, click **User Management**.
2. Click the **Groups** tab, and then click the group in which you want to add or remove members.
3. Click the **Membership** tab.
4. From the **Include a user or group...** drop-down list, select the user or group that you want to add to the current group, and click the **Add to group** button .
5. To remove a member of group from this group:
   a) Enter the name in the **Find** field or locate the name in the list that displays in the dialog.
   b) Click the **X** icon in the **Remove** column for each user or group that you want to remove.
6. Click **Save**.

The changes take affect the next time a user or group member attempts to use a tool.

Deleting users from Workbench

An administrator can delete users and groups from Oracle Endeca Workbench.

There are restrictions on deleting users:

- You cannot delete yourself.
- You cannot delete the predefined `admin` user.
- You cannot delete the Administrators group.

If your site has deployed LDAP or Commerce SSO, deleting users does not delete them from the LDAP or Commerce systems.

To delete a user from Workbench:

1. On the **User Management** page, click the **Delete** icon for the user or group that you want to remove.
2. Click **Delete** in the confirmation dialog.

Configuring the user inactivity logout

You can configure how much time elapses before an inactive user is logged out of Oracle Commerce Workbench in the `webstudio.properties` file.

Follow these steps to set how much time that Workbench users must be inactive before they are automatically logged out. You can also specify how much time elapses before a timeout warning message appears to inactive users.

1. Stop the Endeca Tools Service.
2. Navigate to `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF/conf` (on UNIX).
3. Open the `webstudio.properties` file, and locate the `com.endeca.webstudio.timeout.warning` property.

```
# The warning for impending auto-logout
com.endeca.webstudio.timeout.warning=3300
```

4. Change the value to the number of seconds of inactivity that you want to elapse before an impending automatic logout warning appears to an inactive user.
5. Locate the `com.endeca.webstudio.timeout.logout` property.

```
# The time where a user will be automatically logged out due to inactivity
com.endeca.webstudio.timeout.logout=3600
```

6. Change the value to the number of seconds of inactivity that you want to elapse before an inactive user is logged out of Workbench.
7. Save and close the file.
8. Start the Endeca Tools Service.

Integrating LDAP with Oracle Commerce Workbench

This section describes how to configure Workbench to use LDAP for user authentication.

About LDAP integration with Workbench

LDAP integration allows you to share user identity information and passwords defined in LDAP with Workbench. You can also assign permissions for LDAP users or across an entire LDAP group rather than configuring Workbench users individually.

By configuring Workbench to use LDAP for user authentication, you enable administrators to create Workbench user profiles that are associated with users in an LDAP directory.

Workbench does not write data to the LDAP directory. Passwords and identity information, such as names and e-mail addresses, are maintained in the LDAP directory. Permissions assigned to an LDAP user or group profile in Workbench are stored in the Endeca Configuration Repository.

LDAP user and group profiles can be used in combination with non-LDAP Workbench user and groups that are manually configured by an administrator. Users can authenticate using either method on the same instance of Workbench.

Optionally, you can enable SSL for communication between Workbench and your LDAP server.

Supported versions of LDAP

Workbench supports integration with all LDAP servers that comply with LDAP version 3.

User authentication in Workbench with LDAP enabled

Once you have integrated LDAP with your Workbench installation, you can authenticate users either through LDAP or by configuring them manually in Workbench.

Workbench does the following when a user attempts to log in:

1. Workbench checks whether the user name matches the name of any manually configured Workbench user in the current application. If such a user exists, Workbench attempts to authenticate the user against the password stored for that user.
2. If no manually configured user exists for the name entered, Workbench attempts to authenticate the user against the LDAP directory.
3. If the user is configured for LDAP authentication in Workbench, any associated permissions are applied.
4. If the individual user is not configured for LDAP authentication in Workbench, Workbench checks the LDAP directory for any groups of which the user is a member.

   If a profile exists in Workbench for any of these groups, the user is automatically made a member of each matching group. They inherit the permission of these groups. For more information about inheritance of LDAP group permissions, see “Permissions for LDAP users and groups.”

Permissions for LDAP users and groups

The User Management tool in Workbench allows you to assign permissions to an LDAP user or group.

A user that exists in the LDAP directory but is not associated with a Workbench user profile, either individually or as a member of an LDAP group, cannot log in to Workbench.

A user who authenticates via LDAP is assigned the union of all permissions associated with all groups of which that user is a member. A user who is a member of multiple LDAP groups defined in Workbench is assigned the broadest permission associated with any of the LDAP groups to which that user belongs. Every time users log in to Workbench, their group membership is synchronized with LDAP so that their permissions are current with any group membership changes.

If you create an LDAP user profile in Workbench for an individual who is also a member of one or more LDAP groups defined in Workbench, that user is assigned any permissions that you specify in the User Management tool in addition to any permissions that the user inherits from membership in LDAP groups. If you specify permissions for an LDAP user who is also a member of an LDAP group, then the user is assigned either the

permission specified in the User Management tool or the broadest permission associated with any of the user's LDAP groups, whichever is broader.

About granting administrator privileges in Workbench to LDAP users and groups
With LDAP enabled, you can create user profiles for both LDAP users and LDAP groups that grant administrator privileges in Workbench.

Note: For administrators, the same precedence rules apply when logging in to Workbench as for non-administrators, so that if a manually configured user profile exists in Workbench, a user will not be able to log in via LDAP with the same user name.

Administrators can delete other administrators, but they cannot delete the predefined admin user or the Administrators group. This restriction ensures that changing LDAP permissions or disabling LDAP authentication for Workbench does not disable all administrator logins.

Enabling LDAP authentication in Workbench

LDAP authentication in Workbench is disabled by default.

Because LDAP configuration is unique to each LDAP server and directory, enabling LDAP authentication in Workbench is a manual process.

To enable LDAP authentication in Workbench:

1. Stop the Endeca Tools Service.
2. Navigate to `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF/conf` (on UNIX).
3. Open the `webstudio.properties` file, and locate the `com.endeca.webstudio.useLdap` property:
   ```
   # LDAP Authentication
   com.endeca.webstudio.useLdap=false
   ```
4. Change the value of the property to `true`:
   ```
   com.endeca.webstudio.useLdap=true
   ```
5. Save and close the file.
6. Open the `Login.conf` file.

   This file contains a sample configuration for LDAP authentication.

   Note: By default, Workbench uses the authentication profile in this location. You can specify an alternate configuration file. For more information, see *Specifying the location of the LDAP login configuration file* on page 91.

7. Uncomment and modify the login profile according to your LDAP configuration.

   For details about profile parameters, see *About configuration parameters for the LDAP login profile* on page 86.
8. Save and close the file.
9. Start the Endeca Tools Service.

About disabling LDAP authentication in Workbench

If you disable LDAP authentication in Workbench by setting the property `com.endeca.webstudio.useL¬dap=false` in the `webstudio.properties` file, the options to create a user profile for an LDAP user or an LDAP group do not display in Workbench.

All new user profiles you create must be manually configured in Workbench. Any users who were configured as LDAP users or as members of an LDAP group lose access to Workbench. Existing user profiles for LDAP users or LDAP groups remain in Workbench in an inactive state, and can be edited by an administrator.

About the LDAP login configuration file

Workbench uses the Java Authentication and Authorization Service (JAAS) to authenticate users against an LDAP directory.

Workbench stores LDAP login configuration information in the `%ENDECA_TOOLS_CONF%\conf\Login.conf` file. A sample profile is included in this location by default, but you should modify its parameters as needed for your LDAP configuration. You can also specify an alternate location for the configuration file.

If you want to configure JAAS authentication for other applications running in the Endeca Tools Service (for example, your own Endeca application or Workbench extensions), you can create additional profiles with unique names in the `Login.conf` file.

Templates used in the LDAP login profile

Workbench allows templates to be supplied for certain configuration parameters in the LDAP login profile.

These templates, indicated by `%{}` escapes, allow values from the authentication operation (such as a user or group name entered in Workbench or specific values from the user or group objects in LDAP) to be substituted into the parameter value. Templates also allow you to extract information from the LDAP user or group object (such as the exact user or group name as specified in the LDAP directory) or identity information that is stored in LDAP. The `%{}` escapes are expanded as follows:

Escape	Description
`%{#username}`	The name of the LDAP user as entered in the User Settings tool in Workbench, or the user name entered by a user at the Workbench login page.
`%{#groupname}`	The name of the LDAP group as entered in the User Settings tool in Workbench.
`%{#dn}`	The distinguished name of the user or group object in the LDAP directory.
`%{#dn:n}`	The value of the path field at index n in the distinguished name of the user or group object in LDAP. For example, if the value in the `%{#dn}` field is `cn=joe,ou=Peo¬ple,dc=foo,dc=com`, then the value "`People`" will be substituted for `%{#dn:1}`, while "`joe`" will be substituted for `%{#dn:0}`. Note that unlike the value of `%{#dn}`, which is the raw value returned from the LDAP server, the values returned by this template are not LDAP escaped.
`%{#fieldname}`	The value in the specified field of the user object (or group object when used in the `groupTemplate` or `findGroupTemplate` parameter) under consideration.

About configuration parameters for the LDAP login profile

You specify the values of configuration parameters for LDAP authentication as quoted strings.

If there are any quotation marks (") or backslashes (\) in the string, they must be escaped. For example, if you have the following string:

```
"A string with an "embedded quote" and a \backslash"
```

In the profile, it should be specified as follows:

```
"A string with an \"embedded quote\" and a \\backslash"
```

For most parameter values, single quotation marks (') do not need to be escaped and the values you specify for the parameters can include non-ASCII UTF-8 characters. For additional restrictions on the `userPath`, `groupPath`, and `findGroupPath` parameters, see “LDAP path parameters."

For a full list of the parameters that can be specified in the profile, see the section "Configuration parameters for the LDAP login profile."

Configuration parameters for the LDAP login profile

This section provides a reference of parameters that can be specified in the LDAP login profile.

The following is a full list of the parameters that can be specified in the profile:

Parameter	Description
`serverInfo`	A URL specifying the name and port of the LDAP server to be used for authentication. You can specify multiple LDAP servers. Note that the protocol portion of the URL (that is, `ldap://`) must be in all-lowercase.
`userPath`	The query that is passed to the LDAP server to find an individual user. You can use the `%{#username}` template to insert the name entered in the User Settings tool or the name entered in the Workbench login page into the query. Be sure to set the appropriate objectClass. For example: `userPath="/ou=users,dc=example,dc=com??sub?¬ (&(objectClass=person)(uid=%{#username}))"`
`userTemplate`	A template that specifies how to produce the username from the user object returned by the `userPath` query. This template allows Workbench to automatically correct the case (capital or lowercase) of the username to match the name exactly as specified in the LDAP directory. The correction occurs when you add an LDAP user to Workbench. Therefore, the value returned by this template should match the name entered in the User Settings tool, except for possible differences in case.
`groupPath`	The query that is passed to the LDAP server to find all the groups of which a user is a member. This query is executed when a user logs in to Workbench after looking up the user with the `userPath` query. Thus, you can use templates to insert any information from the user object that is returned by the previous query, such as the distinguished name of the user or any other LDAP attributes, into the `groupPath` query. You can specify multiple values for `groupPath`.

Parameter	Description
groupTemplate	A template that specifies how to produce individual group names from the set of groups returned by the groupPath query. The value returned by this template should match the name of the LDAP group as defined in the Workbench user profile. You can specify multiple values for groupTemplate.
findGroupPath	The query that is passed to the LDAP server to find a specific group. You can use the %{#groupname} template to insert the name of the group as entered in the User Settings tool into the query. Be sure to set the appropriate objectClass. For example: findGroupPath="/ou=groups,dc=example,dc=com¬ ??sub?(&(objectClass=group)(cn=%{#groupname}))"
findGroupTemplate	A template that specifies how to produce the group name from the group object returned by the findGroupPath query. Like the userTemplate, this template is used to correct the case of a group name when you add LDAP group profiles in Workbench. Therefore, the value returned by this template should match the name entered in the User Settings tool, except for possible differences in case.
serviceUsername	The user name of an administrator login to the LDAP server specified in the serverInfo parameter. For example: "Manager@example.com" or "cn=Manager,dc=example,dc=com". If no value is specified for this option, Workbench attempts to authenticate anonymously.
servicePassword	The password to use in conjunction with the serviceUsername value.
serviceAuthentication	Specifies the method of authentication that should be used in connecting to the LDAP server as the administrator account. The permitted values are none, simple, or EXTERNAL.
authentication	Specifies the method of authentication that should be used in binding to the LDAP server as a user account. The permitted values are none, simple, or EXTERNAL
ldapBindAuthentication	*Not supported in Workbench 3.1.1* Optional. By default this is set to true, and Workbench authenticates users by rebinding as the user to the LDAP system, thereby employing the LDAP system's own authentication mechanism.
loginName	Optional. A template login name that will be used to bind to the LDAP server. Default value is %{dn}.

Parameter	Description
passwordAttribute	*Not supported in Workbench 3.1.1* Optional. The name of the attribute on the user object that contains the user's password. Used only if ldapBindAuthentication is set to false. The field specified must contain the user's password in clear text. By default this is set to userPassword.
checkPasswords	*Not supported in Workbench 3.1.1* Optional. Determines whether Workbench checks passwords during logins. Default value is true. If set to false, Workbench uses only the user name to authenticate from the LDAP directory.
useSSL	Optional. Default value is false. If set to true, Workbench attempts to make mutually authenticated SSL connections to the LDAP server. If you set the parameter, ensure that you have configured the LDAP server to use SSL and that the value of serverInfo has the protocol specified as ldaps:// with an SSL port.
keyStoreLocation	Used only if useSSL=true. The location of the Java keystore, which stores keys and certificates. The keystore is where Java gets the certificates to be presented for authentication. The location of the keystore is OS-dependant, but is often stored in a file named .keystore in the user's home directory.  **Note:** Even if this location is on a Windows system, the path uses forward slashes, (/) not backslashes (\).
keyStorePassphrase	Used only if useSSL=true. The passphrase used to open the keystore file.

Configuration parameters for identity information stored in LDAP

The LDAP configuration profile allows you to specify templates to extract identity information from LDAP user or group objects.

Workbench does not store any identity information such as first name, last name, or email address for LDAP users or groups. Instead, Workbench looks up this information in the LDAP directory when needed. The LDAP configuration profile allows you to specify templates to extract identity information from LDAP user or group objects, but they are not required for authentication via LDAP.

Workbench looks up the identity information for a user or group when you use the Check Name function on the **Add User** page to confirm that you are adding the correct LDAP user or group. If you do not specify templates for retrieving identity information, the fields are not filled in when you use Check Name .

Parameter	Description
firstNameTemplate	A template that specifies how to produce the user's first name from the user object, for example, %{#firstNameAttribute}.

Parameter	Description
lastNameTemplate	A template that specifies how to produce the user's last name from the user object, for example, %{#lastNameAttribute}.
emailTemplate	A template that specifies how to produce the user's email address from the user object, for example, %{#emailAttribute}, or %{usernameField}@companydomain.com.
findGroupEmailTemplate	A template that specifies how to produce the email address associated with a group in LDAP from the group object.

LDAP path parameters

The userPath, groupPath, and findGroupPath parameters, when appended to the URL in the serverInfo parameter, must conform to RFC 2255.

This means that certain characters must be encoded in order for the path parameters to form a valid LDAP URL when appended to the value of the serverInfo parameter. Both LDAP and URL encoding may apply to these strings depending on your data. If possible, verify the URL by passing it to your LDAP server before specifying it in the configuration for Workbench.

LDAP encoding affects reserved characters such as the comma (,), equals sign (=), and question mark (?). These characters must be escaped by prepending a backslash (\) when they are not used for their reserved purpose, for example if they appear within a common name or organizational unit.

URL encoding affects characters that are invalid for URLs, such as non-ASCII characters and any unsafe characters as defined in RFC 1738. This includes reserved LDAP characters when they are not used for their reserved purpose. These characters must be replaced with the % sign followed by the appropriate hex code.

For example, if you have the following string as part of your userPath:

```
ou=Endeca Technologies, Inc.
```

Applying LDAP encoding produces the following result:

```
ou=Endeca Technologies\, Inc.
```

Applying URL encoding to the LDAP-encoded string produces:

```
ou=Endeca%20Technologies%5C%2C%20Inc.
```

Any non-ASCII characters or any other characters that are not valid in an LDAP URL must also be properly encoded in the string that you specify in the LDAP login profile.

About specifying multiple values for parameters in the LDAP login profile

You can specify multiple LDAP servers and multiple values for the groupPath element.

If you specify multiple LDAP servers, the servers are assumed to be equivalent. The choice of which LDAP server to contact is made randomly. If an LDAP server cannot be reached, the LoginModule plug-in proceeds through the remaining servers in order of configuration, wrapping if necessary. For example, if five servers are configured and Server 3 is the first to be contacted, the remaining order of contact is Server 4, Server 5, Server 1, and finally Server 2.

You can specify multiple LDAP servers with multiple instances of the serverInfo parameter, by using the format:

```
serverInfo.n = ”ldap://server_url:port_number”
```

For example:

```
serverInfo.0="ldaps://globalcatalog.corp.example.com:3269"
serverInfo.1="ldap://globalcatalog.us.example.com:3009"
```

You can also specify multiple values for the groupPath attribute by using the same format, for example:

```
groupPath.0="/ou=groups,dc=example,dc=com??sub?(member=%{#dn})"
groupPath.1="/dc=example,dc=com?memberOf?sub?(AccountName=%{#username})"
groupPath.2="/dc=example,dc=com?memberOf?sub?(CN=%{#dn})"
```

If you specify more than one groupPath, Workbench sends all the queries to the LDAP server to discover the groups of which a user is a member.

You can specify corresponding values for groupTemplate for each groupPath. In this case, the value for groupTemplate.0 is applied to the results of the groupPath.0 query, groupTemplate.1 is applied to the results of groupPath.1, and so on.

For example:

```
groupTemplate.0="%{#dn:0}"
groupTemplate.1="%{#memberOf:0}"
groupTemplate.2="%{#memberOf:0}"
```

Specifying the location of the LDAP login configuration file

By default, Workbench uses %ENDECA_TOOLS_CONF%\conf\Login.conf (on Windows) or $ENDE¬CA_TOOLS_CONF/conf/Login.conf (on UNIX) as the LDAP login configuration file.

If you are running the Endeca Tools Service as a Windows service, see the section "Specifying the location of the LDAP login configuration file using Windows Services."

You can substitute any configuration file that includes a LDAP login profile named Webstudio. The file does not have to be named Login.conf, but it must be saved in UTF-8 format.

If you want to store the configuration file in a different location, you can pass this location to the Java JVM. How you specify the location depends on how you run the Endeca Tools Service.

If you are running the Endeca Tools Service on Windows from the command line or on UNIX:

1. Navigate to %ENDECA_TOOLS_ROOT%\server\bin (on Windows) or $ENDECA_TOOLS_ROOT/server/bin (on UNIX).
2. Open the setenv.bat or setenv.sh file.
3. Locate the line that sets JAVA_OPTS:

   ```
   set JAVA_OPTS=-Xmx1024m -XX:MaxPermSize=128m -Djava.security.auth.login.con¬
   fig=%ENDECA_TOOLS_CONF%/conf/Login.conf
   ```

4. Change the -Djava.security.auth.login.config parameter to point to the location of your configuration file on the file system.

Specifying the location of the LDAP login configuration file using Windows Services

By default, Workbench uses %ENDECA_TOOLS_CONF%\conf\Login.conf (on Windows) or $ENDE¬CA_TOOLS_CONF/conf/Login.conf (on UNIX) as the LDAP login configuration file.

If you are running the Endeca Tools Service on UNIX or on Windows from a command line, see "Specifying the location of the LDAP login configuration file."

You can substitute any configuration file that includes a LDAP login profile named Webstudio. The file does not have to be named Login.conf, but it must be saved in UTF-8 format.

If you want to store the configuration file in a different location, you can pass this location to the Java JVM. How you specify the location depends on how you run the Endeca Tools Service.

If you are running the Endeca Tools Service on Windows from the command line or on UNIX:

1. Open the Registry Editor.
2. Navigate to the `HKEY_LOCAL_MACHINE\SOFTWARE\Apache Software Foundation\ Procrun 2.0\EndecaToolsService\Parameters\Java\Options` key.
3. Right click **Options** in the right pane and select **Modify**.
   The **Edit Multi-String** dialog box displays.
4. Locate the following parameter:
   ```
   -Djava.security.auth.login.config=%ENDECA_TOOLS_CONF%/conf/Login.conf
   ```
5. Change the path to point to the location of your configuration file.
6. Click **OK**.

Troubleshooting user authentication in Workbench with LDAP enabled

If a user cannot log in to Workbench, one of several error messages displays.

Incorrect Username or Password

If the user is entering the correct LDAP user name and password, there may be a manually configured Workbench user in the same application with the same user name or a Workbench administrator with the same user name.

A user with a manually configured profile always takes precedence over a user authenticating via LDAP. For more details about the behavior of users with the same name, see “User profiles for LDAP users and groups.”

An error occurred while trying to validate your credentials

This error displays when any error occurs other than a user name-password mismatch or an absence of permissions. It can indicate anything from a connectivity issue with the LDAP server to a mistake in the configuration in the LDAP login configuration file located in `%ENDECA_TOOLS_CONF%\conf\Login.conf`. For more information about the login profile, see “Configuration parameters for the LDAP login profile.”

Check the Workbench log, located in `%ENDECA_TOOLS_CONF%\logs\webstudio.#.log` for more information about the causes of authentication failures. In most cases, the solution is to adjust the LDAP query strings to return the desired results. If possible, test the query URLs against your LDAP server using an independent tool in order to confirm that they behave as expected and that each query for a user or group that exists in the directory returns a unique user or group object.

Configuring SSL for Oracle Commerce Workbench

This section describes how to configure your Workbench installation to use SSL for Web browser connections. Workbench does not support SSL communication with Guided Search components (such as the EAC and MDEX Engine).

About configuring SSL in Workbench

SSL is disabled by default for Workbench as a server.

To enable SSL security between Workbench and its clients, you must do the following:

- Enable the SSL version of Workbench.

- Set up a certificate for the Workbench server. For details, see the *Oracle Commerce Guided Search Security Guide*. The server certificate for Workbench must be issued to the fully qualified domain name of the server.
- Modify the `server.xml` file for the Endeca Tools Service to enable the HTTPS connector and point to the new keystore.

Clients can make secure connections to Workbench either by taking advantage of a redirect from the non-SSL port or, if you have disabled the non-SSL port or do not wish to use the redirect, by making an HTTPS connection directly to the SSL port.

Workbench supports version 3.0 of the Secure Sockets Layer (SSL) protocol for its communication endpoints.

Enabling the SSL version of Oracle Commerce Workbench

The non-SSL version of Oracle Commerce Workbench is installed by default.

To enable the SSL version of Workbench:

1. Stop the Endeca Tools Service.
2. Navigate to `%ENDECA_TOOLS_CONF%\conf\Standalone\localhost` (on Windows) or `$ENDE¬ CA_TOOLS_CONF/conf/Standalone/localhost` (on UNIX).
3. Open the `ROOT.xml` file.
4. Locate the line in which the `docBase` is defined.
   For example:
   ```
   docBase="${catalina.base}/../webapps/workbench-legacy-tools-3.1.1.war"
   ```
   Note: The file name in the example may not match the one in your installation.
5. Change this to point to the SSL version of the WAR by adding -ssl to the filename.
   For example:
   ```
   docBase="${catalina.base}/../workbench-legacy-tools-3.1.1-ssl.war"
   ```
6. Save and close the file.
7. Start the Endeca Tools Service.

If you want to restore the non-SSL version at a later date, you can reverse the process by editing the `ROOT.xml` file accordingly.

Modifying the server.xml for the Endeca Tools Service

Before you can use SSL with Workbench, you must edit its `server.xml` file as described.

This procedure assumes you have already generated server certificates for Workbench as described in the *Oracle Commerce Guided Search Security Guide* and uploaded them to the Endeca Workbench server.

To enable the HTTPS connector:

1. Stop the Endeca Tools Service.
2. Navigate to `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF/conf` (on UNIX).
3. Open the `server.xml` file.
4. Locate and remove the comments around the Connector element for port 8446 as follows:
   ```
   <Connector port="8446" SSLEnabled="true"
   protocol="org.apache.coyote.http11.Http11Protocol"
   maxPostSize="0"
   maxThreads="150" scheme="https" secure="true"
   ```

```
clientAuth="false" sslProtocol="TLS"
keystoreFile="conf/eac.ks" keystorePass="eacpass"
truststoreFile="conf/ca.ks" truststorePass="eacpass"
/>
```

5. Optionally, change the port number to something other than 8446 if you do not want to use that default.

   If you do not use the default port, update the `redirectPort` attribute on the non-SSL HTTP connector to point to the new port as in the following example:

```
<!-- Define a non-SSL HTTP/1.1 Connector on port 8006 -->
 <Connector port="8006" maxHttpHeaderSize="8192"
  maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
  enableLookups="true" redirectPort="8446" acceptCount="10"
  connectionTimeout="60000" disableUploadTimeout="true" debug="0"
  URIEncoding="UTF-8"/>
```

6. If you want to disable the redirect from the non-secure port to the secure port, comment out the non-SSL connector in the `server.xml` file. By default, the redirect is enabled.

   Caution: If you choose to disable the non-SSL connector, the Deployment Template cannot communicate with Oracle Commerce Workbench and you must manually update your application's instance configuration.

7. Update the `keystoreFile`, `keystorePass`, `truststoreFile`, and `truststorePass` with the appropriate values for your certificates.

   The `keystoreFile` and `truststoreFile` values should be the paths to the location where you uploaded your keystore and truststore files. These paths can be specified as absolute paths, or paths relative to `EN¬DECA_TOOLS_CONF`, although the files themselves can be located anywhere on the server.

8. Save and close the file.
9. Start the Endeca Tools Service.

Customizing Oracle Commerce Workbench

This section describes how to customize the Oracle Commerce Workbench interface and how to add extensions to Oracle Commerce Workbench.

The navigation menu and launch page

You can configure the items in the navigation menu on the left and on the launch page of Oracle Commerce Workbench by modifying the `ws-mainMenu.xml` file in `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF/conf` (on UNIX).

By editing `ws-mainMenu.xml`, you can do any of the following:

- Add a new menu item.
- Remove an item from the menu.
- Specify the order in which the menu items display.
- Specify whether an item is in the top-level menu or in a submenu.
- Specify whether a menu item displays on the launch page.

Navigation menu nodes

A menu item is either a leaf or a node. A node is a top-level menu item that does not link directly to any pages.

Instead it has children that are leaf items and are displayed in a submenu. Each node is defined in a <menun¬ode> element in ws-mainMenu.xml that takes the following attributes.

Note: The defaultName, defaultDescription, and defaultIcon values are only used if equivalent attributes are not specified in the resource property files in the conf/locales folder. For example, if you do not specify a menu node name for French locales in those resource property files, then the defaultName value is used. For more information on locales, see *Localizing menu nodes* on page 74.

Attribute name	Attribute value	Required
id	The id of a predefined node in Oracle Endeca Workbench or a unique string identifying a custom node. For more information on predefined nodes, see “Predefined menu nodes in Oracle Commerce Workbench.”	yes
defaultName	The display name for this node that appears in the navigation menu.	no
defaultTitle	*Deprecated in release 4.0.0.* Use defaultName instead.	
defaultDescription	A brief description of this node that appears on the launch page in Oracle Commerce Workbench.	no
defaultIcon	An absolute or relative URL to a custom image for this node’s entry on the launch page. (Relative URLs are relative to <hostname>:8006.	no

A menunode element requires one or more child menuitem elements.

This example of a ws-mainMenu.xml file defines a custom menu node with extensions as its child items.

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

<mainmenu xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="mainMenu.xsd">
 <menunode id="myextensions"
       defaultName="My Extensions"
       defaultDescription="These are my custom extensions.">
  <menuitem id="extensionA"/>
  <menuitem id="extensionB"/>
 </menunode>
</mainmenu>
```

Predefined menu nodes in Oracle Commerce Workbench

There are several predefined menu nodes in Oracle Endeca Workbench. You can specify the placement of the predefined nodes in the menu and what items display under them, but you cannot modify the titles or specify localized titles.

The predefined nodes in Oracle Commerce Workbench are as follows:

Node id	Node description
`reports`	Reports
`settings`	Application Settings
`administration`	Administration

About navigation menu leaf items

A leaf is a menu item that links to a page, and also has an entry on the launch page.

A leaf can be either in the top-level menu or in a submenu as the child of a node. Leaf items cannot have child items. Menu items display in the order in which they are listed in `ws-mainMenu.xml`.

Each leaf in the menu is defined in a `menuitem` element in `ws-mainMenu.xml` that takes the following attributes:

Attribute name	Attribute value	Required?
`id`	The `id` of a predefined page in Workbench or the `id` of an extension as defined in `ws-extensions.xml`. For more information about extensions, see “Workbench extensions."	yes
`onLaunchPage`	*Deprecated and ignored in release 3.1.1.* If set to true, the menu item displays on the launch page in the order in which it is listed in `ws-main¬Menu.xml`. Default value is `false`.	no

Note: For a full list of predefined pages and their corresponding ids, see "Predefined `menuitem` elements."

This example of a `ws-mainMenu.xml` file defines a menu that shows top-level leaf items, items nested within a predefined node.

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

<mainmenu xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mainMenu.xsd">

    <menuitem id="xmgr"/>
    <menuitem id="thesaurus"/>
    <menunode id="reports">
        <menuitem id="reports.today"/>
        <menuitem id="reports.daily"/>
        <menuitem id="reports.weekly"/>
    </menunode>
    <menunode id="settings">
        <menuitem id="user-segments"/>
        <menuitem id="preview-settings"/>
        <menuitem id="report-settings"/>
    </menunode>
```

```
    <menunode id="administration">
        <menuitem id="locks"/>
        <menuitem id="user-management"/>
        <menuitem id="eac-settings"/>
    </menunode>
    <menuitem id="eac-admin-console"/>
</mainmenu>
```

Predefined menuitem elements

This section is a reference table listing all of the predefined pages and corresponding ids available in the `ws-mainMenu.xml` file.

The predefined pages and their corresponding ids are as follows:

Workbench page	Menu item id
Experience Manager  **Note:** This menu item is only available in installations of Oracle Commerce Guided Search that include Experience Manager.	`xmgr`
Rule Manager  **Note:** This menu item is only available in the Oracle Commerce Guided Search package.	`rmgr`
Thesaurus	`thesaurus`
Today's Reports	`reports.today`
Daily Reports	`reports.daily`
Weekly Reports	`reports.weekly`
Report Scheduler	`report-settings`
Preview Settings  **Note:** This menu item is only available in installations of Oracle Commerce Guided Search that include Experience Manager.	`preview-settings`
EAC Admin Console	`eac-admin-console`
EAC Connection Settings	`eac-settings`
User Segments	`user-segments`
User Management	`user-management`

Updating the Oracle Commerce Workbench menu and launch page

The menu items on the launch page of Oracle Commerce Workbench are configurable.

You can configure the items in the menu and on the Workbench launch page by modifying the ws-main¬Menu.xml file in %ENDECA_TOOLS_CONF%\conf (on Windows) or $ENDECA_TOOLS_CONF/conf (on UNIX).

To update the navigation menu and launch page:

1. Stop the Endeca Tools Service.
2. Navigate to %ENDECA_TOOLS_CONF%\conf (on Windows) or $ENDECA_TOOLS_CONF/conf (on UNIX).
3. Open ws-mainMenu.xml in a text editor and add or modify menu items as necessary.
4. Save and close the file.
5. Start the Endeca Tools Service.

Hiding user segment settings in Workbench

If your Guided Search application is integrated with the Oracle Commerce Platform (ATG) and all user segments will be created and managed in Business Control Center, you can prevent Workbench users from creating user segments by hiding these settings in Workbench.

You can remove the user segment settings from the main menu and launch page in Workbench by modifying the ws-mainMenu.xml file in %ENDECA_TOOLS_CONF%\conf (on Windows) or $ENDECA_TOOLS_CONF/conf (on UNIX). You only need to do this if you plan on managing user segments exclusively through the ATG Business Control Center, otherwise you can leave the settings alone and still use both types of user segments.

To update the Workbench main menu and launch page:

1. Stop the Endeca Tools Service.
2. Navigate to %ENDECA_TOOLS_CONF%\conf (on Windows) or $ENDECA_TOOLS_CONF/conf (on UNIX).
3. Open ws-mainMenu.xml in a text editor.
4. Locate the **settings** menu node, and remove **<menuitem id="user-segments"/>** .

```
<menunode id="settings">
    <menuitem id="user-segments"/>
    <menuitem id="preview-settings"/>
    <menuitem id="report-settings"/>
</menunode>
```

5. Save and close the file.
6. Start the Endeca Tools Service.

Workbench extensions

Extensions enable you to incorporate Web applications related to your Guided Search implementation as plug-ins to Oracle Commerce Workbench.

An extension can be as simple as a static Web page or it can provide sophisticated functionality to control, monitor, and configure your Endeca applications. Extensions can be hosted on the same server as Workbench or on another server.

About configuring extensions in Oracle Commerce Workbench

Extensions are defined in the ws-extensions.xml file in %ENDECA_TOOLS_CONF%\conf (on Windows) or $ENDECA_TOOLS_CONF/conf (on UNIX).

The default `ws-extensions.xml` file is as follows:

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

<extensions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="extensions.xsd">

</extensions>
```

Each extension is defined in an extension element within extensions. You can specify as many additional extensions as you need by adding more extension elements. For a full list of list of required and optional attributes, see "Extension element attributes."

This example of a `ws-extensions.xml` file defines a simple extension that enables a link to the Endeca Web site.

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

<extensions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="extensions.xsd">
 <extension id="endecaHome"
   defaultName="Endeca home page"
   defaultDescription="Visit the Endeca home page"
   url="http://www.endeca.com">
</extension>
</extensions>
```

Extension element attributes

This section provides a reference table of required and optional extension element attributes.

The extension element takes the following attributes.

Note: The `defaultName`, `defaultDescription`, and `defaultIcon` values are only used if equivalent attributes are not specified in the resource property files in the `conf/locales` folder. For example, if you do not specify an extension name for French locales in those resource property files, then the `defaultName` value is used. For more information on locales, see *Localizing Workbench extensions* on page 73.

Attribute name	Attribute value	Required?
`id`	A unique string identifying this extension. Do not define an extension with the same id as one of the predefined Oracle Commerce Workbench pages. For a list of predefined Workbench pages and their ids, see the reference table in “Predefined `menuitem` elements."	yes
`defaultName`	The display name for this extension that appears in the navigation menu and launch page in Oracle Commerce Workbench.	no
`defaultDescription`	A brief description of this extension that appears on the launch page in Oracle Commerce Workbench.	no

Attribute name	Attribute value	Required?
`url`	An absolute or relative URL to this extension. The extension must be a Web application reachable through HTTP or HTTPS, but it does not have to run on the same server as Oracle Commerce Workbench. If the extension is hosted on the same server as Oracle Commerce Workbench, the extension URL can be site relative (omitting the host name and port and beginning with the leading slash).	yes
`defaultIcon`	An absolute or relative URL to a custom image for this extension's entry on the launch page. (Relative URLs are relative to `<workbench host>:<workbench port>`.	no
`launchImageUrl`	*Deprecated in release 11.0.0.* Use `defaultIcon` instead.	
`role`	This attribute is only used when you want an extension to display for administrators on the Workbench Admin page. The only value allowed is "admin." Any other value is ignored.	no
`sharedSecret`	A shared key that Oracle Commerce Workbench uses to calculate the authentication token.	no

Enabling extensions in Oracle Commerce Workbench

You enable Workbench extensions by editing the `ws-extensions.xml` file.

To enable extensions in Oracle Commerce Workbench:

1. Stop the Endeca Tools Service.
2. Navigate to `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF/conf` (on UNIX).
3. Open `ws-extensions.xml` in a text editor and add or modify extensions as necessary.

   Note: In addition to adding an extension to Oracle Commerce Workbench, you must also enable links to the new extension in the navigation menu and the launch page.

4. Save and close the file.
5. Start the Endeca Tools Service.

URL tokens and Workbench extensions

Oracle Commerce Workbench can pass information to an extension through URL tokens in order to enable the extension to authenticate users, connect to the EAC Central Server, and maintain its state if a user navigates away from the extension and back again during the same session.

You use URL tokens by specifying them in the url attribute of the extension definition in `%ENDE¬CA_TOOLS_CONF%\conf\ws-extensions.xml`. The name of the URL parameter does not have to match the id of the token as listed in the preceding table.

For example, the following extension definition creates a URL that passes the EAC host, port, and application to the extension:

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

<extensions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="extensions.xsd">
 <extension id="testExtension"
  defaultName="Test Extension"
  defaultDescription="Demonstrates extensions with tokens."
  url="http://www.example.com:8989/TestExtension/index.jsp?eac-host=
${EAC_HOST}&amp;amp;eac-port=${EAC_PORT}&amp;amp;eac-app=${EAC_APP}">
 </extension>
</extensions>
```

Note the use of the & entity in the url attribute in place of the ampersand in the URL. In general, you should ensure that the `ws-extensions.xml` file validates against the provided schema before updating Oracle Commerce Workbench with the new configuration.

URL token reference

This section provides a complete list of all tokens available to pass to Workbench extensions.

The following tokens are available to pass to extensions:

Token ID	Token description
`${AUTH}`	An SHA-256 hash value used to authenticate users coming from Oracle Commerce Workbench.
`${EAC_APP}`	The name of the application that the Workbench user is logged in to.
`${EAC_HOST}`	The host running the EAC Central Server to which Endeca Workbench is currently connected.
`${EAC_PORT}`	The port on the EAC host through which Oracle Endeca Workbench and the EAC Central Server communicate.
`${EXTENSION_ID}`	The id of the extension as defined in `ws-extensions.xml`.
`${LOCALE}`	The locale of Oracle Commerce Workbench; this is the value of the `com.endeca.webstudio.locale` property in `webstudio.properties`.
`${TS}`	The time, in milliseconds since 00:00:00 UTC January 1, 1970, when the user navigates to the extension.
`${USERNAME}`	The username of the Workbench user accessing the extension.

Token ID	Token description
${WEBSTUDIO_SESSIONID}	The id of the user's current Workbench session. The extension can use this in combination with the ${USERNAME} token to maintain the state of the extension throughout a single Workbench session, for instance by storing the information in a cookie.

Token-based authentication for Workbench extensions

You can enable extensions to authenticate users coming from Endeca Workbench by including an authentication token in the URL.

Oracle Commerce Workbench calculates the value of the token by generating an SHA-256 hash from a portion of the URL and a shared secret. The portion of the URL that is used for the hash consists of everything after the host name and port, including the leading slash, but excluding the value of the AUTH token itself. The shared secret is a string that is specified in `ws-extensions.xml` and is also stored in the extension itself.

For example, the following `ws-extensions.xml` file defines an extension with a URL that uses the AUTH and TS tokens:

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

<extensions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="extensions.xsd">
 <extension id="authExtension"
  defaultName="Authenticated Extension"
  defaultDescription="Demonstrates token-based authentication."
  url="http://localhost:8080/AuthExtension/index.jsp?timestamp=${TS}&amp;
auth=${AUTH}"
  sharedSecret="secret!@#$%^*(987654321" />
</extensions>
```

In this case, the value of the authentication token is the hash of a String similar to the following:

```
/AuthExtension/index.jsp?timestamp=1189702462936&auth=secret!@#$%^*(987654321
```

The extension can verify that a user is coming from Oracle Endeca Workbench by calculating the hash of the same string and comparing the result to the value of the AUTH token. This ensures that the user visiting the extension has logged in to Oracle Commerce Workbench and has the permission (if any) that is required to access the extension.

Because the AUTH token is based in part on the URL, it is recommended that you include the time stamp of the request to introduce some variation in the value of the token. The time stamp can also be used to filter out stale requests and limit the possibility of an eavesdropper reusing the same URL to gain access to the extension.

The following Java code shows how the extension defined in the preceding example can authenticate users from Oracle Commerce Workbench:

```
// These values depend on what you defined in ws-extensions.xml
String extensionSecret="secret!@#$%^*(987654321";
final String authTokenParameterName = "auth";
final String timeStampParameterName = "timestamp";

// Set the tolerance, in milliseconds, before a request is considered too old
int allowedTimeStampSlackInMS = 5 * 60 * 1000;

// Calculate the hash of the substring of the URL and the shared secret
String url = request.getRequestURI() + "?" + request.getQueryString();
```

```
String findAuthToken = "&" + authTokenParameterName + "=";
url = url.substring(0, url.indexOf(findAuthToken) + findAuthToken.length());
String authCode = request.getParameter(authTokenParameterName);

MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] secretDigest = md.digest((urlForHash + extensionSecret).getBytes("UTF-
8"));

StringBuffer hashCode = new StringBuffer();

for (int i : secretDigest) {
  String str = Integer.toHexString(i+128);
  if (str.length() < 2) {
    str = "0" + str;
  }
  hashCode.append(str);
}

// Compare the hash to the value of the AUTH token
if (!hashCode.toString().equals(authCode)) {
 // Authentication fails because AUTH token did not match
}

// Compare the time stamp of the request to the current time stamp
long currentTime = new Date().getTime();
long ts = Long.parseLong(request.getParameter(timeStampParameterName));

if ( Math.abs(ts - currentTime) > allowedTimeStampSlackInMS) {
 // Authentication fails because request is too old
}
```

The example extension places the AUTH token at the end of the URL, making it more convenient to build the substring of the URL for the hash.

However, the AUTH token can be in any position in the URL. For instance, the URL can be defined in `ws-extensions.xml` as follows:

```
url="http://localhost:8080/AuthExtension/index.jsp?auth=${AUTH}&amp;
timestamp=${TS}"
```

This results in a URL similar to this:

```
http://localhost:8080/AuthExtension/in¬
dex.jsp?auth=dc40570f2e7111fbe1af820a854ca817&timestamp=1189702462936
```

The value of the authentication token is the hash of a String similar to the following:

```
/AuthExtension/index.jsp?auth=&timestamp=1189702462936secret!@#$%^*(987654321
```

In this case, the code in the extension to remove the value of the authentication token from the URL is more complex.

Troubleshooting Workbench extensions

This section provides troubleshooting information about Workbench extensions.

If the extension does not have a link in the navigation menu or launch page:

- Stop and restart the Endeca Tools Service. Changes to the XML configuration files for extensions and the navigation menu do not go into effect until the service is restarted.

- Ensure that you have the required Workbench user permission to access the extension.
- Ensure that a menu item for the extension is specified in `ws-mainMenu.xml` and that the id attribute matches the id of the extension as defined in `ws-extensions.xml`. Defining an extension in `ws-extensions.xml` does not automatically add a link to the navigation menu in Oracle Commerce Workbench.
- If you have no applications defined in Oracle Commerce Workbench, the only links that display in the navigation menu are for the EAC Admin Console and EAC Settings. To enable display of the full Workbench menu, you must first provision an application.

If the link displays in the menu but the extension does not display when you click the link:

- Ensure that the URL for the extension specified in `ws-extensions.xml` is a valid HTTP or HTTPS URL. A Workbench extension must be a Web application running in a Web server.

If an error message displays after updating ws-extensions.xml:

There may be a problem with your XML configuration files that prevents Oracle Commerce Workbench from starting up. The error messages in the Oracle Workbench log can help you identify whether one of the following is the case:

- One or more of the XML configuration files is missing. The following files must be present in `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF/conf` (on UNIX):
  - `ws-extensions.xml` and its associated schema, `extensions.xsd`
  - `ws-mainMenu.xml` and its associated schema, `mainMenu.xsd`

  The files are created in this location when you install Endeca. By default, the `ws-extensions.xml` file defines no extensions. The `ws-mainMenu.xml` file controls the display of the navigation menu and launch page.

  If you have deleted one of these files, you can restore the default file by copying it from `%ENDECA_TOOLS_ROOT%\workspace_template\conf` (on Windows) or `$ENDECA_TOOLS_ROOT/workspace_template/conf` (on UNIX).

- One or more of the configuration files contains badly formed or invalid XML.

  Ensure that the configuration files contain well-formed XML. In particular, check that any ampersand that is used within an attribute value is specified as the & entity.

  Use an XML tool to validate any configuration files that you have edited against the associated schema in `%ENDECA_TOOLS_CONF%\conf` (on Windows) and `$ENDECA_TOOLS_CONF/conf` (on UNIX).

Integrating Workbench with the Business Control Center

if your implementation makes use of the Business Control Center (BCC), you can enable single sign-on and add a Workbench extension that adds a link to the BCC within Workbench, facilitating navigation between the two tools.

The steps below assume that you have a BCC instance configured and running.

To integrate Workbench with the Business Control Center:

1. Run the CIM and select **[W] Workbench configuration – OPTIONAL**, configuring the fields as described in the *Oracle Commerce Platform Installation and Configuration Guide*.
2. Copy the `workbench` directory generated by the CIM from your BCC host machine to the machine hosting your Workbench installation.

3. On your Workbench host, stop the Endeca Tools Service.
4. Configure Workbench with the new **BCC Home** and **BCC Access Control** extensions.

   If you have not added any custom extensions of your own, you can overwrite the `ws-mainMenu.xml` and `ws-extensions.xml` files with the versions in the copied `workbench` directory. Otherwise:

   a) Open the `ToolsAndFrameworks\<version>\server\workspace\conf\ws-mainMenu.xml` file in a text editor.
   b) Create a menu item for the BCC by adding a top-level `<menuitem>` element with an `id` attribute of `bcc-home`:

   ```
   <menuitem id="xmgr"/>
   <menuitem id="bcc-home"/>
   ...
   ```

   c) Create a menu item for BCC Access Control by adding a `<menuitem>` element with an `id` attribute of `bcc-access-control` within the `user-access` node:

   ```
   <menuitem id="xmgr"/>
   <menuitem id="bcc-home"/>
   <menunode id="user-access">
     <menuitem id="user-management"/>
     <menuitem id="bcc-access-control"/>
   </menunode>
   ...
   ```

   d) Save and close the file.
   e) Open the `ToolsAndFrameworks\<version>\server\workspace\conf\ws-extensions.xml` file in a texte editor.
   f) Define the `bcc-home` and `bcc-access-control` extensions by adding the following elements:

   ```
   <extension id="bcc-home" defaultName="BCC" defaultDescription ="BCC"
     url="http://myBCCHost.myBCCDomain.com:<bcc port>/atg/bcc"
     externalURL="true"/>
   <extension id="bcc-access-control"
     defaultName="BCC Access Control"
     defaultDescription="BCC Access Control"
     role="admin"
     url="http://myBCCHost.myBCCDomain.com:<bcc port>/ControlCenter/application/

         accesscontrol"
     externalURL="true"/>
   ```

   Where the `url` attribute provides links to the BCC and to the Access Control Panel, respectively.

   g) Save and close the file.
5. Merge the localized Strings in the copied `workbench\locales` directory with those in `ToolsAndFrameworks\<version>\server\workspace\conf\locales`:

   a) Open the `Resources_<language>` file you wish to modify.
   b) Copy in the following properties, with values corresponding to the text you wish to use:

Property	Description
`atg_flags`	ATG flags indicating localization or internationalization.
`tool.bcc-home.name`	The name of the BCC Home link.
`tool.bcc-home.description`	The description for the BCC Home link.
`tool.bcc-home.icon`	The path to the icon file for the BCC Home link.

Property	Description
tool.bcc-access-control.name	The name of the BCC Access Control link.
menu.user-access.name	The name of the User Access menu.
menu.user-access.description	The description for the User Access menu.
menu.user-access.icon	The path to the icon file for the User Access menu.

For example:

```
# These resources should be localized.
atg_flags=i18n,l10n

# BCC extension
tool.bcc-home.name = Business Control Center
tool.bcc-home.description = Create, preview, and deploy site content and
product catalogs.
# @i18n:begin:trans(false)
tool.bcc-home.icon = http://myserver.example.com:8080/atg/images/
  BCC_home/icon_wb_bcc.png
# @i18n:end:trans

# BCC Access Control extension
tool.bcc-access-control.name = BCC Access Control

# User Access menu
menu.user-access.name = User Access
menu.user-access.description = Add and remove Oracle Commerce Workbench and
 Business Control Center Users and modify their tool access and content per¬
missions.
# @i18n:begin:trans(false)
menu.user-access.icon = /ifcr/apps/endeca/workbench-assets/images/home/icon_us¬
er.png
# @i18n:end:trans
```

c) Save and close the file.
d) Repeat the above steps for each language in your Workbench implementation.

6. Start the Endeca Tools Service.

To configure single sign-on with the BCC and Workbench, you can use Oracle Access Management or the Commerce Single Sign-On solution. See *Integrating Workbench with Oracle Access Management Single Sign-On* on page 107 or *Integrating Workbench with Oracle Commerce Single Sign-On* on page 106 for details.

Integrating Workbench with Oracle Commerce Single Sign-On

You can configure Workbench to authenticate users using Oracle Commerce Single Sign-On (SSO).

Before configuring Workbench integration with Commerce Single Sign-On, you must have a Commerce SSO server configured. For details, see the *Oracle Commerce Platform-Guided Search Integration Guide*.

Commerce SSO is provided as an alternative to third-party SSO solutions. It integrates authentication for Workbench and the ATG Business Control Center, allowing a user to switch between tools without encountering additional login screens.

Important: This functionality is independent of Oracle Access Management (OAM) as part of the Oracle Identity Management platform. The two solutions should not be used for the same components. See

Integrating Workbench with Oracle Access Management Single Sign-On on page 107 if you are configuring integration with OAM.

To integrate Workbench with Commerce SSO:

1. Navigate to your `ToolsAndFrameworks\<version>\server\workspace\conf` directory.
2. Open `webstudio.properties` in a text editor.
3. Locate the configuration titled `# Commerce SSO Authentication`.
4. Set `com.endeca.webstudio.useSSO` to `true`:

```
# Commerce SSO Authentication
com.endeca.webstudio.useSSO=true
```

5. Uncommment the following properties:
   - `com.endeca.webstudio.sso.loginURL`
   - `com.endeca.webstudio.sso.keepAliveFrequency`
6. Set the uncommented properties to their respective values:

Property	Value
`com.endeca.webstu¬ dio.sso.loginURL`	The URL of the Commerce SSO `login` servlet.
`com.endeca.webstu¬ dio.sso.keepAliveFrequency`	The time interval, in seconds, for which to send Keep Alive requests. This indicates that a session is active and should not time out. **Note:** The session timeout limit is configured on the Commerce SSO server.

   For example:

```
# Commerce SSO Authentication
com.endeca.webstudio.useSSO=true
com.endeca.webstudio.sso.loginURL=http(s)://<SSO Host>:<SSO Port>/sso/login
com.endeca.webstudio.sso.keepAliveFrequency=1800
```

   Where the protocol is either `http` or `https` and `<SSO Host>` and `<SSO Port>` are the host machine and port, respectively, for the Commerce SSO server.
7. Save and close the file.

Integrating Workbench with Oracle Access Management Single Sign-On

You can configure Workbench to authenticate users using Oracle Access Management (OAM).

Before configuring Workbench integration with OAM, you must have an OAM server configured. For details, see the *Oracle® Fusion Middleware Installation Guide for Oracle Identity and Access Management* .

To integrate Workbench with OAM SSO:

1. Configure proxy settings for the OAM Oracle HTTP Server:
   a) Log on to the OAM host machine.
   b) Stop the Oracle HTTP Server.
   c) Navigate to the `<IAM Home>\config\OHS\ohs1` directory.

d) Open the mod_wl_ohs.conf file.

e) For each instance of Workbench, add the following within the <IfModule weblogic_module> element:

```
Listen [VirtualHost-Port]
<VirtualHost *:[VirtualHost-Port]>
    <Location>
        SetHandler weblogic-handler
        WebLogicHost [Workbench-Host]
        WebLogicPort [Workbench-Port]
    </Location>
</VirtualHost>
```

Where [VirtualHost-Port] is the port for the virtual host on the Oracle HTTP Server, and [Workbench-Host] and [Workbench-Port] are the Workbench host and port, respectively.

For example, for two Workbench instances:

```
Listen 9999
<VirtualHost *:9999>
    <Location>
        SetHandler weblogic-handler
        WebLogicHost 10.154.137.102
        WebLogicPort 8006
    </Location>
</VirtualHost>
Listen 9998
<VirtualHost *:9998>
    <Location>
        SetHandler weblogic-handler
        WebLogicHost 10.154.137.103
        WebLogicPort 8006
    </Location>
</VirtualHost>
```

f) Save and close the file.

2. Configure a dynamic Preferred Host for Webgate:
   a) On the OAM host machine, navigate to the <IAM Home>\config\OHS\ohs1\webgate\config directory.
   b) Open the Webgate configuration file, ObAcessClient.xml.
   c) Set the preferredHost value to SERVER_NAME.
   d) Save and close the file.
   e) Start the Oracle HTTP Server.
   f) Access the OAM Console from the URL.
   For example, http://<OAM Host>:7001/oamconsole/.
   g) Select the **System Configuration** tab.
   h) Expand **Access Manager** in the left pane.
   i) In the tree view, expand **SSO Agents** and double click **OAM Agents**.
   The **OAM Agents** tab opens in the right pane.
   j) Click the **Search** button in the **Search** panel.
   A list of OAM agents appears.
   k) Select the **webgate** agent in the **Search Results** list.
   l) Verify that the Preferred Host column has a value of SERVER_NAME.
   If not, click the **Edit** button, set the value, and click the **Apply** button in the upper-right to save changes.

3. Add the Oracle HTTP Server to the IAMSuiteAgent host identifier:
   a) Within the OAM Console, select the **Policy Configuration** tab.

b) Double click **Host Identifiers** in the left pane.
The **Host Identifiers** tab opens in the right pane.
c) Click the **Search** button in the **Search** panel.
A list of host identifiers appears.
d) Click the link to select the **IAMSuiteAgent**.
e) Click the **Add** button in the **Host Name Variations** panel.
f) Set the **Host Name** field to the Oracle HTTP Server address and leave the **Port** field blank.
g) Click the **Apply** button in the upper-right.

4. Create a host identifier for each Workbench instance:
   a) Within the OAM Console, return to the **Host Identifiers** tab in the right pane.
   b) Click the **Create** button.
   The **Create Host Identifier** tab opens.
   c) Set a unique **Name**, such as `WB1`.
   d) Click **Apply**.
   e) Click the **Add** button in the **Host Name Variations** panel.
   f) Set the **Host Name** and **Port** fields to the Workbench host and port.
   g) Click **Apply**.
   h) Repeat the steps above for each Workbench instance.
5. Create the application domain:
   a) Within the OAM Console, select the **Policy Configuration** tab.
   b) Double click **Application Domains** in the left pane.
   The **Application Domains** tab opens in the right pane.
   c) Click the **Create** button.
   The **Create Application Domain** tab opens.
   d) Set the **Name** to `ATG/Endeca`.
   e) Click **Apply**.
6. Create an authentication policy for protected resources on the new domain:
   a) Within the **Application Domains** tab in the right pane, select the `ATG/Endeca` domain you created in Step 5.
   An **ATG/Endeca** tab opens in the right pane.
   b) Select the **Authentication Policies** tab.
   c) Click the **Create** button.
   The **Create Authentication Policy** tab opens.
   d) Set the **Name** to `Endeca`.
   e) In the **Authentication Scheme** dropdown, select `LDAPScheme`.
   f) Click **Apply**.
7. Create an open authorization policy on the new domain.
   This allows global access to protected resources once the user is authenticated.
   a) Within the OAM Console, return to the **ATG/Endeca** tab in the right pane.
   b) Select the **Authorization Policies** tab.
   c) Click the **Create** button.
   The **Create Authorization Policy** tab opens.
   d) Set the **Name** to `open`.
   e) Click **Apply**.
   f) Select the **Conditions** tab.
   g) Click the **Add** button.
   The **Add Condiition** dialog appears.

h) Set the **Name** to TRUE.
i) In the **Type** dropdown, select True.
j) Click **Add Selected**.
k) Click **Apply**.
l) Select the **Rules** tab.
m) In the **Allow Rule** section, move the TRUE(true) condition to the **Selected Conditions** list.
n) Click **Apply**.

8. Define a protected resource for each instance of Workbench.

   These are the URLs within the application that trigger an authentication challenge if the request is not accompanied by a valid OAM SSO authentication cookie. For Workbench, this includes all URLs.

   a) Within the OAM Console, return to the **ATG/Endeca** tab in the right pane.
   b) Within the **Application Domains** panel, select the **Resources** tab.
   c) Click the **Create** button.
      The **Create Resource** tab opens.
   d) In the **Type** dropdown, select HTTP.
   e) In the **Host Identifier** dropdown, select the host identifier you created in Step 4.
   f) In the **Resource URL** field, enter /**.
   g) In the **Protection Level** dropdown, select Protected.
   h) In the **Authentication Policy** dropdown, select Endeca policy you created in Step 6.
   i) In the **Authorization Policy** dropdown, select the open policy you created in Step 7.
   j) Click **Apply**.
   k) Repeat the steps above for each Workbench instance.

9. Configure Workbench to authenticate against the OAM LDAP server.

   Modify the %ENDECA_TOOLS_CONF%\conf\Login.conf file to set the serverInfo property and query templates according to OAM LDAP settings. For detailed information on LDAP configuration, see *About configuration parameters for the LDAP login profile* on page 86.

   For example:

```
Webstudio {
com.endeca.workbench.authentication.ldap.WorkbenchLdapLoginModule required
serverInfo="ldap://myOAMHost.mydomain.com:1234"
serviceUsername="cn=myAdminUser"
servicePassword="myAdminPassword"
serviceAuthentication="simple"
authentication="simple"
useSSL="false"
keyStoreLocation="C:/Endeca/MDEXEngine/workspace/conf/webstudio.jks"
keyStorePassphrase="keypass"

// The query used to look up a user in the LDAP directory and
// templates that extract information from the user object
userPath="/cn=users,dc=myOAMHost,dc=mydomain,dc=com??sub?(&(objectClass=per¬
son)(uid=%{#username}))"
userTemplate="%{#uid}"
firstNameTemplate="%{#givenName}"
lastNameTemplate="%{#sn}"
emailTemplate="%{#mail}"

// The query used to look up a group in the LDAP directory and
// templates that extract information from the group object
findGroupPath="/cn=groups,dc=myOAMHost,dc=mydomain,dc=com??sub?(&(object¬
Class=groupofUniqueNames)(cn=%{#groupname}))"
findGroupTemplate="%{#dn:0}"
```

```
groupEmailTemplate="%{#mail}"

// The query and template used to fetch the groups associated
// with a user when the user logs in to Web Studio
groupPath="/cn=groups,dc=myOAMHost,dc=mydomain,dc=com??sub?(uniquemember=%{#dn})"
groupTemplate="%{#dn:0}"
;
};
```

10. Configure Workbench to use OAM:
    a) On your Workbench host, navigate to the `ToolsAndFrameworks\<version>\server\workspace\conf` directory.
    b) Open `webstudio.properties` in a text edtior.
    c) Locate the configuration titled `# OAM Authentication`.
    d) Set `com.endeca.webstudio.useOAM` to `true`:

```
# OAM Authentication
com.endeca.webstudio.useOAM=true
```

    e) Uncomment `com.endeca.webstudio.oam.logoutURL` and set the callback URL to the OAM logout page:

    For example, if the OAM logout URL is `/mydomain/logout.jsp`:

```
#com.endeca.webstudio.oam.keyStore=oamkeystore.ks
#com.endeca.webstudio.oam.keyStoreType=JKS
#com.endeca.webstudio.oam.keyStorePassword=<password>
com.endeca.webstudio.oam.logoutURL=/ifcr/system/sling/logout.html?oam.lo¬
gout.url=/mydomain/logout.jsp%3Fend_url=/ifcr
```

    f) Save and close the file.

11. in Workbench, create a set of LDAP user profiles that correspond to your OAM users.
    For information, see the section on *Integrating LDAP with Oracle Commerce Workbench* on page 83.

Optionally, you can configure Identity Assertion validation in OAM if your environment requires it.

Enabling Identity Assertion validation in OAM

You can enable Identity Assertion validation to confirm that users passed to Workbench through OAM originate from a trusted instance. This comes with a performance cost, since Workbench must verify each incoming authentication or authorization request.

The steps below assume you have already integrated Workbench with Single Sign-On through OAM.

When OAM passes a request to Workbench, it includes `oam_remote_user` information on the request header. Optionally, you can configure Workbench to verify that the header originates from a trusted OAM instance by checking that the certificate in the header matches a certificate in the OAM keystore.

Note: If both Workbench and the OAM server are running inside the same secure firewall, Identity Assertion validation may not be necessary.

To enable Identity Assertion validation with OAM:

1. Export the OAM Identity Assertion X509 certificate:
   a) Log in to the OAM host machine.
   b) Navigate to `$MW_HOME/Oracle_IDM1/common/bin`.
   c) Run `wslt.sh`.

d) Run the `connect()` command to connect to the Weblogic server.
e) Run the `listCred(map="OAM_STORE", key="jks")` command to display the OAM keystore.
f) Navigate to `$base_domain/config/fmwconfig`.
g) Run the `keytool -exportcert` command with the following flags to export the keystore information to an `assertion.cer` file:

```
keytool -exportcert -v -alias assertion-key -storetype JCEKS
-keystore .oamkeystore -file assertion.cer
```

h) Copy the `assertion.cer` output file to the Workbench host machine.

For example, `C:\Endeca\ToolsAndFrameworks\<version>\server\workspace\conf\assertion.cer`.

2. Import the OAM Identity Assertion X509 certificate to the `oamkeystore.ks` file on the Workbench host:
   a) Run the `keytool -importcert` command with the following flags to import the keystore information to the `oamkeystore` file:

```
keytool -importcert -v -alias assertion-key -keypass <password>
-keystore c:\Endeca\ToolsAndFrameworks\<version>\server\workspace\conf\oamkey¬
store.ks
-file  c:\Endeca\ToolsAndFrameworks\<version>\server\workspace\conf\asser¬
tion.cer
```

   Where `<password>` is the desired keystore password.

3. Configure Workbench to use Identity Assertion Validation:
   a) On your Workbench host, navigate to the `ToolsAndFrameworks\<version>\server\workspace\conf` directory.
   b) Open `webstudio.properties` in a text edtior.
   c) Locate the configuration titled `# OAM Authentication`.
   d) Set `com.endeca.webstudio.oam.identityAssertionValidation` to `true`:

```
# OAM Authentication
com.endeca.webstudio.useOAM=true
com.endeca.webstudio.oam.identityAssertionValidation=true
```

   e) Uncomment the following properties:
      - `com.endeca.webstudio.oam.identityAssertionValidation`
      - `com.endeca.webstudio.oam.keyStore`
      - `com.endeca.webstudio.oam.keyStoreType`
      - `com.endeca.webstudio.oam.keyStorePassword`
   f) Set the uncommented properties to their respective values:

Property	Value
`com.endeca.webstudio.oam.iden¬ tityAssertionValidation`	`true` enables Identity Assertion validation.
`com.endeca.webstudio.oam.keyStore`	The absolute path to the `oamkeystore.ks` file.
`com.endeca.webstudio.oam.keyStoreType`	The OAM keystore type (JKS).
`com.endeca.webstudio.oam.keyStorePass¬ word`	The OAM keystore password.

For example:

```
com.endeca.webstudio.oam.keyStore=C:/Endeca/ToolsAndFrameworks/<version>/
server/workspace/conf/oamkeystore.ks
com.endeca.webstudio.oam.keyStoreType=JKS
com.endeca.webstudio.oam.keyStorePassword=<password>
com.endeca.webstudio.oam.logoutURL=/ifcr/system/sling/logout.html?
oam.logout.url=/oamsso/logout.html%3Fend_url=/ifcr
```

Where <password> is the keystore password set in Step 2.

g) Save and close the file.
h) Restart the Endeca Tools Service.

4. Modify the authentication policy in OAM to enable Identity Assertion:
   a) Access the OAM Console from the URL.
   For example, http://<OAM Host>:7001/oamconsole/.
   b) Select the **Policy Configuration** tab.
   c) Double click **Application Domains** in the left pane.
   The **Application Domains** tab opens in the right pane.
   d) Click the **Search** button in the **Search** panel.
   A list of application domains appears.
   e) Select the ATG/Endeca domain.
   An **ATG/Endeca** tab opens in the right pane.
   f) Select the **Authentication Policies** tab.
   g) Click the **Search** button in the **Search** panel.
   A list of authentication policies appears.
   h) Select the Endeca authentication policy.
   An authentication policy tab opens.
   i) Select the **Responses** tab.
   j) Enable the **Identity Assertion** checkbox.
   k) Click **Apply**.
5. Modify the authorization policy in OAM to enable Identity Assertion:
   a) Select the previously opened **ATG/Endeca** tab.
   b) Select the **Authorization Policies** tab.
   c) Click the **Search** button in the **Search** panel.
   A list of authorization policies appears.
   d) Select the open authorization policy.
   An authorization policy tab opens.
   e) Select the **Responses** tab.
   f) Enable the **Identity Assertion** checkbox.
   g) Click **Apply**.

Chapter 8

Configuring an EAC Application

This section provides an overview of the elements defined in AppConfig.xml.

About configuring an EAC application

The standard processing and script operations of the Deployment Template are sufficient to support the operational requirements of most projects. Some applications require customization to enable custom processing steps, script behavior, or even directory structure changes.

Developers are encouraged to use the template as a starting point for customization. The scripts and modules provided with the template incorporate Oracle's best practice recommendations for synchronization, archiving, and update processing. The Deployment Template is intended to provide a set of standards on which development should be founded, while allowing the flexibility to develop custom scripts to meet specific project needs.

There are two ways to configure an EAC application:

- Configure `AppConfig.xml` files. The simplest form of configuration consists of editing the `AppConfig.xml` configuration file and its associated configuration files to change the behavior of components or to add or remove components.

  This type of configuration includes the addition of removal of Dgraphs to the main cluster or even the creation of additional clusters. In addition, this category includes adjustment of process arguments (for example, adding a Java classpath for the Forge process in order to enable the use of a Java Manipulator), custom properties and directories (for example, changing the number of index archives that are stored on the indexing server).

- Change behavior of existing BeanShell scripts. Scripts are written in the Java scripting language BeanShell. Scripts are defined in the `AppConfig.xml` document and are interpreted at runtime by the BeanShell interpreter. This allows developers and system administrators to adjust the behavior of the baseline, partial, and configuration update scripts by simply modifying the configuration document.

About the application configuration files

The application configuration file `<app dir>/config/script/AppConfig.xml` and its associated files define the hosts, components, and scripts that make up an EAC application and the that orchestrate updates by executing the defined components.

The Deployment Template provides a single `AppConfig.xml` file that contains pointers to refer to other files that define distinct parts of an application, separate scripts from component provisioning, and are used for other purposes. The full set of application configuration files are as follows:

- `AssemblerConfig.xml` - Specifies the application server cluster that allows for quick updating of all Assemblers.
- `AuthoringDgraphCluster.xml` - Specifies the Dgraphs used in the authoring environment and a script that pushes configuration from Workbench to each Dgraph in the authoring cluster.
- `DataIngest.xml` - Specifies data processing scripts, including the baseline update script, partial update script, and the components to perform data processing such as CAS or Forge and Dgidx.
- `DgraphDefaults.xml` - Specifies default values that are inherited by all Dgraph components. These values include host IDs, data processing paths, and Dgraph flags.
- `InitialSetup.xml` - Specifies scripts to perform initial setup tasks, such as uploading initial configuration to Workbench.
- `LiveAppServerCluster.xml` - Specifies your application server clusters, the servers within each cluster, and the applications running on a given Assember.
- `LiveDgraphCluster.xml` - Specifies the Dgraphs used in the live environment and a script that pushes configuration from Workbench to each Dgraph in the live cluster.
- `ReportGeneration.xml` - Specifies the hosts used for logging and report generations, and several scripts that produce log files at different time intervals.
- `UsageCollectionConfig.xml` - Specifies the Dgraph clusters and application server clusters from which usage is collected.
- `WorkbenchConfig.xml` - Specifies the Endeca Configuration Repository component, the Workbench Manager component, and a script that promotes content from the authoring environment to the live environment.

In addition to these files, any number of `--app-config` arguments may be specified to the Controller class in the EAC development toolkit. All of the objects in the files will be read and processed and scripts can refer to components, hosts, or other scripts defined in other files.

Configuring the application configuration files

This topic guides you through the process of configuring an EAC application.

To configure the application configuration files:

1. Edit the `AppConfig.xml` file in `<app dir>/config/script` to reflect the details of your environment. Specifically, set the following values:
   - Specify the `eacHost` and `eacPort` attributes of the app element with the correct host and port of the EAC Central Server.
   - Specify the `host` elements with the correct host name or names and EAC ports of all EAC Agents in your environment.
   - Specify the `WorkbenchManager` component with the correct host and port for Oracle Commerce Workbench.
2. Edit the `DataIngest.xml` file in `<app dir>/config/script` to reflect your data processing requirements. Specifically, ensure that the baseline update script and partial update script are correct and that the CAS or Forge and Dgidx components are correctly configured.
3. Edit the `DgraphDefaults.xml` file in `<app dir>/config/script` with the default values that are inherited by all Dgraph components in both the authoring cluster and live cluster.

4. Edit the `AuthoringDgraphCluster.xml` file in `<app dir>/config/script` to ensure the authoring Dgraph, the authoring cluster and post-startup script is correct for your environment.
5. Edit the `LiveDgraphCluster.xml` file in `<app dir>/config/script` to ensure the live Dgraph, the live cluster and post-startup script is correct for your environment.
6. Edit the `LiveAppServerCluster.xml` file in `<app dir>/config/script` to ensure that the application server clusters, the servers within each cluster, and the applications running on the servers are correct for your environment.
   a) For each server cluster, create an `<app-server-cluster>` element with an `id` attribute that corresponds to the cluster name.

      For example:

      ```
      <app-server-cluster id="LiveAppServerCluster">
      </app-server-cluster>
      ```

   b) For each server within the cluster, create an `<app-server>` element with the following attributes:

      - `id` — The name of the server.
      - `hostName` — The DNS name or IP address of the server hosting the Assembler.
      - `port` — The port on which the Assembler Web application is running.

      For example:

      ```
      <app-server id="LiveDiscover" hostName="assemblerHost.example.com"
      port="8006">
      </app-server>
      ```

   c) For each application running on a given Assembler, create a `<web-app>` element with the following attributes:

      - `id` — The name of the Assembler application.
      - `contextPath` — The path to the application relative to the Assembler server.
      - `sslEnabled` — Optionally, whether the application is SSL-enabled.

      For example:

      ```
      <web-app id="DiscoverWebApp" contextPath="/discover" sslEnabled="true" />
      <web-app id="DiscoverAsService" contextPath="/discoverAsService" />
      ```

   d) Add the `<web-app>` elements to their respective `<app-server>`s as referenced elements.

      For example:

      ```
      <app-server id="LiveDiscover" hostName="assemblerHost" port="8006">
          <web-app ref="DiscoverWebApp" />
          <web-app ref="DiscoverAsService" />
      </app-server>
      ```

   e) Add the `<app-server>` elements to their respective `<app-server-cluster>`s as referenced elements.

      For example:

      ```
      <app-server-cluster id="LiveAppServerCluster">
          <app-server ref="LiveDiscover" />
      </app-server-cluster>
      ```

7. Edit the `AssemblerConfig.xml` file in `<app dir>/config/script` to ensure that it references the application server clusters that are correct for your environment.
8. Edit the `WorkbenchConfig.xml` file in `<app dir>/config/script` to ensure the Workbench Manager and IFCR components are correct for your environment.
9. Edit the `UsageCollectionConfig.xml` file in `<app dir>/config/script` to ensure that the Dgraph clusters and application server clusters from which usage is collected are correct for your environment.

10. If necessary, edit the `ReportGeneration.xml` file in `<app dir>/config/script`. This file does not usually require any modifications.

The following topics describe the components that you can define in the application configuration files.

Global configuration

The `AppConfig.xml` file defines global application-level configuration, including the host and port of the EAC Central Server, the application name and whether or not SSL is to be used when communicating with the EAC Central Server.

In addition, a default working and log directory are specified and a default `lockManager` is specified for use by other elements defined in the document. All elements inherit these settings or override them.

```
<!--
  ########################################################################
  # EAC Application Definition
  #
-->
<app appName="MyApp" eacHost="myhost1.company.com" eacPort="8888"
    dataPrefix="MyApp" sslEnabled="false" lockManager="LockManager">
  <working-dir>${ENDECA_PROJECT_DIR}</working-dir>
  <log-dir>./logs</log-dir>
</app>
```

The LockManager

The LockManager component obtains and releases locks and sets or removes flags using the EAC synchronization Web service.

A LockManager object is associated with the elements in the application, allowing multiple objects to test for the existence of locks and flags. If it is configured to release locks on failure, the Deployment Template attempts to release all acquired locks when a script or component invocation fails. Multiple LockManager components may be configured, if it is appropriate for some locks to be released on failure while others remain.

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

Dgraph configuration

You can specify default settings for the Dgraphs in an application, as well as group them into Dgraph clusters in order to easily perform operations on multiple Dgraphs.

Global Dgraph settings

The `dgraph-defaults` element in the `DgraphDefaults.xml` file defines shared settings that are inherited (or overridden) by each Dgraph in an application. These default properties are used in the baseline and partial update scripts to define operational functionality.

Properties specify general Dgraph properties.

- `srcIndexDir` - Location from which a new index will be copied to a local directory on the Dgraph host.
- `srcIndexHostId` - Host from which a new index will be copied to a local directory on the Dgraph host.

- srcPartialsDir - Location from which a new partial update will be copied to a local directory on the Dgraph host.
- srcPartialsHostId - Host from which partial updates will be copied to a local directory on the Dgraph host.
- srcCumulativePartialsDir - Location from which all partial updates accumulated since the last baseline update will be copied to a local directory on the Dgraph host.
- srcCumulativePartialsHostId - Host from which all partial updates accumulated since the last baseline update will be copied to a local directory on the Dgraph host.
- srcDgraphConfigDir - Location from which Dgraph configuration files will be copied to a local directory on the Dgraph host.
- srcDgraphConfigHostId - Host from which Dgraph configuration files will be copied to a local directory on the Dgraph host.
- numLogBackups - Number of log directory backups to store.
- shutdownTimeout - Number of seconds to wait for a component to stop (after receiving a stop command).
- numIdleSecondsAfterStop - Number of seconds to pause/sleep after a component is stopped. Typically, this will be used to ensure that log file locks are release by the component before proceeding.

Directories specify local directories where generated Dgraph files are copied.

- localIndexDir - Local directory to which a single copy of a new index is copied from the source index directory on the source index host.
- localCumulativePartialsDir - Local directory to which partial updates are copied from the source (cumulative) partials directory on the source partials host.
- localDgraphConfigDir - Local directory to which Dgraph configuration files are copied from the source Dgraph config directory on the source Dgraph config host.

Args specify arguments passed in as command-line flags. Flags that require arguments include successive <arg> elements that specify those arguments. Note that these settings are not cumulative. Specifying <args> for an individual Dgraph completely overrides any default settings.

- Dgraph flags are documented in the MDEX Engine documentation.

```
<!--
########################################################################
# Global Dgraph settings, inherited by all dgraphs
#
-->
<dgraph-defaults>
  <properties>
    <property name="srcIndexDir" value="./data/dgidx_output" />
    <property name="srcIndexHostId" value="ITLHost" />
    <property name="srcPartialsDir" value="./data/partials/forge_output" />
    <property name="srcPartialsHostId" value="ITLHost" />
   <property name="srcCumulativePartialsDir" value="./data/partials/cumulative_par¬
tials" />
    <property name="srcCumulativePartialsHostId" value="ITLHost" />
    <property name="srcDgraphConfigDir" value="./data/workbench/dgraph_config"
/>
    <property name="srcDgraphConfigHostId" value="ITLHost" />
    <property name="numLogBackups" value="10" />
    <property name="shutdownTimeout" value="30" />
    <property name="numIdleSecondsAfterStop" value="0" />
  </properties>
  <directories>
    <directory name="localIndexDir">./data/dgraphs/local_dgraph_input</directory>

    <directory name="localCumulativePartialsDir">./data/dgraphs/local_cumula¬
tive_partials</directory>
```

```
    <directory name="localDgraphConfigDir">./data/dgraphs/local_dgraph_config</di¬
rectory>
  </directories>
  <args>
    <arg>--threads</arg>
    <arg>2</arg>
    <arg>--whymatch</arg>
    <arg>--spl</arg>
    <arg>--dym</arg>
    <arg>--dym_hthresh</arg>
    <arg>5</arg>
    <arg>--dym_nsug</arg>
    <arg>3</arg>
    <arg>--stat-abins</arg>
  </args>
  <startup-timeout>120</startup-timeout>
</dgraph-defaults>
```

Individual Dgraph settings

The <dgraph> element defines settings for a specific Dgraph. Individual Dgraphs are typically defined in the same file that defines the Dgraph clusters which contain them.

```
<dgraph id="AuthoringDgraph" host-id="AuthoringMDEXHost" port="15002">
  <properties>
    <property name="DgraphContentGroup" value="Authoring" />
  </properties>
  <log-dir>./logs/dgraphs/AuthoringDgraph</log-dir>
  <input-dir>./data/dgraphs/AuthoringDgraph/dgraph_input</input-dir>
  <update-dir>./data/dgraphs/AuthoringDgraph/dgraph_input/updates</update-dir>
</dgraph>
```

Each Dgraph takes the following attributes:

- `id` - The ID of the specified Dgraph.
- `host-id` - A reference to the host machine, specified elsewhere in the file within a <host> element.
- `port` - The port on which the Dgraph listens.

It also takes the following nested elements:

- <log-dir> - The directory on the specified host machine to store Dgraph log output.
- <input-dir> - The directory on the specified host machine to store input data for baseline updates.
- <update-dir> - The directory on the specified host machine to store input data for partial updates.

Dgraph clusters and hosts

Dgraph clusters apply actions to an entire cluster of Dgraphs, rather than manually iterating over a number of Dgraphs. They contain logic associated with Dgraph restart strategies, and can also be configured to copy data in parallel or serially. This setting applies to copies that are performed to distribute a new index, partial updates or configuration updates to each server that hosts a Dgraph.

You can define multiple clusters, with no restriction around which Dgraphs belong to each cluster or how many clusters a Dgraph belongs to. Typically, clusters are defined with the <dgraph-cluster> element in `AuthoringDgraphCluster` and `LiveDgraphCluster` XML files, with references to all Dgraphs that belong to that cluster. Each file also includes host information for the cluster. Each host must be given a unique ID. The port specified for each host is the port on which the EAC Agent is listening, which is the Endeca HTTP Service port on that server.:

```
<!--
  ########################################################################
```

```
  # Authoring MDEX Hosts - The machines used to host all MDEX processes
  # for the 'authoring environment' MDEX cluster.
  #
-->
<host id="AuthoringMDEXHost" hostName="myhost1.company.com" port="8888" />

<!--
  ##########################################################################
  # Authoring Dgraph Cluster - The 'authoring environment' MDEX cluster.
  #
-->
<dgraph-cluster id="AuthoringDgraphCluster" getDataInParallel="true" enabled="true"

  configSnapshotDir="./data/dgraphcluster/AuthoringDgraphCluster/config_snapshots">

  <dgraph ref="AuthoringDgraph" />
</dgraph-cluster>
```

Each Dgraph cluster takes the following attributes:

- `id` - The ID of the Dgraph cluster.
- `getDataInParallel` - Whether to retrieve data for all Dgraphs in the cluster in parallel.
- `enabled` - Whether the cluster is active.

Restart, update, and content groups

In addition to standard Dgraph configuration and process arguments in a `<dgraph>` element, the following custom properties define restart, update, and promotion behavior:

- `restartGroup`
- `updateGroup`
-

The `restartGroup` property indicates the Dgraph's membership in a restart group. When applying a new index or configuration updates to a cluster of Dgraphs (or when updating a cluster of Dgraphs with a provisioning change such as a new or modified process argument), the Dgraph cluster object applies changes simultaneously to all Dgraphs in a restart group.

Similarly, the `updateGroup` property indicates the Dgraph's membership in an update group. When applying partial updates, the Dgraph cluster object applies changes simultaneously to all Dgraphs in an update group.

This means that a few common restart strategies can be applied as follows:

- To restart/update all Dgraphs at once: specify the same restartGroup/updateGroup value for each Dgraph.
- To restart/update Dgraphs one at a time: specify a unique restartGroup/updateGroup value for each Dgraph, or omit one or both of the custom properties on all Dgraphs (causing the template to assign a unique group to each Dgraph).
- To restart/update Dgraphs on each server simultaneously: specify the same restartGroup/updateGroup value for each Dgraph on a physical server.
- To restart Dgraphs one at a time but apply partial updates to all Dgraphs at once: specify a unique restartGroup value for each Dgraph and specify the same updateGroup value for each Dgraph.

Restart and update group values are arbitrary strings. The DgraphCluster will iterate through the groups in alphabetical order, though non-standard characters may result in groups being updated in an unexpected order.

The `DgraphContentGroup` property indicates whether a Dgraph belongs to a "Live" or "Authoring" environment.

Startup and shutdown scripts

Dgraph components can specify the name of a script to invoke prior to shutdown and the name of a script to invoke after the component is started. These optional attributes must specify the ID of a Script defined in the XML file(s). These BeanShell scripts are executed just before the Dgraph is stopped or just after it is started. The scripts behave identically to other BeanShell scripts, except that they have an additional variable, invokingObject, which holds a reference to the Dgraph that invoked the script. This functionality is typically used to implement calls to a load balancer, adding or removing a Dgraph from the cluster as it is updated.

The following example shows two dummy scripts (which just log a message, but could be extended to call out to a load balancer) provisioned to run pre-shutdown and post-startup for Dgraph1.

```
<dgraph id="Dgraph1" host-id="MDEXHost" port="15000"
    pre-shutdown-script="DgraphPreShutdownScript"
    post-startup-script="DgraphPostStartupScript">
  <properties>
    <property name="restartGroup" value="A" />
  </properties>
  <log-dir>./logs/dgraphs/Dgraph1</log-dir>
  <input-dir>./data/dgraphs/Dgraph1/dgraph_input</input-dir>
  <update-dir>./data/dgraphs/Dgraph1/dgraph_input/updates</update-dir>
</dgraph>

<script id="DgraphPreShutdownScript">
  <bean-shell-script>
    <![CDATA[
    id = invokingObject.getElementId();
    hostname = invokingObject.getHost().getHostName();
    port = invokingObject.getPort();
    log.info("Removing dgraph with id " + id + " (host: " + hostname +
      ", port: " + port + ") from load balancer cluster.");
    ]]>
  </bean-shell-script>
</script>

<script id="DgraphPostStartupScript">
  <bean-shell-script>
    <![CDATA[
    id = invokingObject.getElementId();
    hostname = invokingObject.getHost().getHostName();
    port = invokingObject.getPort();
    log.info("Adding dgraph with id " + id + " (host: " + hostname +
      ", port: " + port + ") to load balancer cluster.");
    ]]>
  </bean-shell-script>
</script>
```

The following log excerpt shows these scripts running when a new index is being applied to the dgraph:

```
[03.10.08 10:03:28] INFO: Applying index to dgraphs in restart group 'A'.
[03.10.08 10:03:28] INFO: [MDEXHost] Starting shell utility 'mkpath_dgraph-input-
new'.
[03.10.08 10:03:30] INFO: [MDEXHost] Starting copy utility 'copy_in¬
dex_to_temp_new_dgraph_input_dir_for_Dgraph1'.
[03.10.08 10:03:35] INFO: Removing dgraph with id Dgraph1 (host: mdex1.mycompa¬
ny.com, port: 15000) from load balancer cluster.
[03.10.08 10:03:35] INFO: Stopping component 'Dgraph1'.
[03.10.08 10:03:37] INFO: [MDEXHost] Starting shell utility 'move_dgraph-in¬
put_to_dgraph-input-old'.
[03.10.08 10:03:39] INFO: [MDEXHost] Starting shell utility 'move_dgraph-input-
new_to_dgraph-input'.
```

```
[03.10.08 10:03:40] INFO: [MDEXHost] Starting backup utility 'back¬
up_log_dir_for_component_Dgraph1'.
[03.10.08 10:03:42] INFO: [MDEXHost] Starting component 'Dgraph1'.
[03.10.08 10:03:45] INFO: Adding dgraph with id Dgraph1 (host: mdex1.mycompany.com,
 port: 15000) to load balancer cluster.
[03.10.08 10:03:45] INFO: [MDEXHost] Starting shell utility 'rmdir_dgraph-input-
old'.
```

Note that the `dgraph-default` element can also specify the use of pre-shutdown and post-startup scripts as attributes, allowing all Dgraphs in an application to execute the same scripts. For example:

```
<dgraph-defaults pre-shutdown-script="DgraphPreShutdownScript"
    post-startup-script="DgraphPostStartupScript">

  ...

</dgraph-defaults>
```

Enabling SSL for a Dgraph

You can configure the Dgraph for SSL by using the following elements to define the certificates to use for SSL:

- `cert-file` specifies the path of the `eneCert.pem` certificate file that is used by the Dgraph to present to any client. This is also the certificate that the Application Controller Agent should present to the Dgraph when trying to talk to the Dgraph.
- `ca-file` specifies the path of the `eneCA.pem` Certificate Authority file that the Dgraph uses to authenticate communications with other Guided Search components.
- `cipher` specifies one or more cryptographic algorithms, one of which Dgraph will use during the SSL negotiation. If you omit this setting, the Dgraph chooses a cryptographic algorithm from its internal list of algorithms. See the *Endeca Commerce Security Guide* for more information

All three elements are first-level children of the `<dgraph-defaults>` element.

The following example shows the three SSL elements being used within the `dgraph-default` element:

```
<dgraph-defaults>
...
   <cert-file>
      C:\Endeca\PlatformServices\workspace\etc\eneCert.pem
   </cert-file>
   <ca-file>
      C:\Endeca\PlatformServices\workspace\etc\eneCA.pem
   </ca-file>
   <cipher>AES128-SHA</cipher>
</dgraph-defaults>
```

Data Ingest configuration with Forge

One or many Forge components are defined for baseline update processing and partial update processing depending on the deployment type you choose.

If necessary, you can define a Forge cluster component to apply actions to an entire cluster of Forges, rather than manually iterating over a number of Forges. You could use this feature to run several instances of Forge in parallel to process large joins.

In addition, the object contains logic associated with executing Forges in parallel based on Forge groups, which are described below. Multiple Forge clusters can be defined, with no restriction around which Forges belong to each cluster or how many clusters a Forge belongs to.

A Forge cluster is configured with references to all Forges that belong to that cluster. In addition, the cluster can be configured to copy data in parallel or serially. This setting applies to copies that are performed to retrieve source data and configuration to each server that hosts a Forge component. By default, the template sets this value to true.

```
<!--
########################################################################
# Forge Cluster
#
-->
<forge-cluster id="ForgeCluster" getDataInParallel="true">
  <forge ref="ForgeServer" />
  <forge ref="ForgeClient1" />
  <forge ref="ForgeClient2" />
</forge-cluster>
```

In addition to standard Forge configuration settings and process arguments, the Deployment Template uses several configurable properties and custom directories during processing:

- `numLogBackups` - Number of log directory backups to store.
- `numStateBackups` - Number of autogen state directory backups to store.
- `numPartialsBackups` - Number of cumulative partials directory backups to store. It is recommended that you increase the default value of 5. The reason is that the files in the updates directory for the Dgraph are automatically deleted after partials are applied to the Dgraph. The number you choose depends on how often you run partial updates and how many copies you want to keep.
- `incomingDataHost` - Host to which source data files are extracted.
- `incomingDataDir` - Directory to which source data files are extracted.
- `incomingDataFileName` - Filename of the source data files that are extracted.
- `configHost` - Host from which configuration files and dimensions are retrieved for Forge to process.
- `configDir` - Directory from which configuration files and dimensions are retrieved for Forge to process.
- `cumulativePartialsDir` - Directory where partial updates are accumulated between baseline updates.
- `wsTempDir` - Temp Oracle Commerce Workbench directory to which post-Forge dimensions are copied to be uploaded to the Workbench.
- `skipTestingForFilesDuringCleanup` - Used for directory-cleaning operations. If set to "true", will skip the directory-contents test and instead proceed directly to cleaning the directory. The default behavior is to test the directory contents and skip cleanup if the directory is not empty.
- The properties documented in the "Fault tolerance and polling interval properties" topic.

This excerpt combines properties from both the baseline and partial update Forge to demonstrate the use of all of these configuration settings.

```
<properties>
  <property name="forgeGroup" value="A" />
  <property name="incomingDataHost">ITLHost</property>
  <property name="incomingDataFileName">project_name-part0-*</property>
  <property name="configHost">ITLHost</property>
  <property name="numStateBackups" value="10" />
  <property name="numLogBackups" value="10" />
  <property name="numPartialsBackups" value="5" />
  <property name="skipTestingForFilesDuringCleanup" value="true" />
</properties>
<directories>
  <directory name="incomingDataDir">./data/partials/incoming</directory>
  <directory name="configDir">./config/pipeline</directory>
  <directory name="cumulativePartialsDir">
    ./data/partials/cumulative_partials
  </directory>
```

```
  <directory name="wsTempDir">./data/web_studio/temp</directory>
</directories>
```

In addition to standard Forge configuration and process arguments, Forge processes add a custom property used to define which Forge processes run in parallel with each other when they belong to a Forge cluster.

`forgeGroup` - Indicates the Forge's membership in a Forge group. When the run method on a Forge cluster is executed, Forge processes within the same Forge group are run in parallel. Forge group values are arbitrary strings. The Forge cluster iterates through the groups in alphabetical order, though non-standard characters may result in groups being updated in an unexpected order.

Defining indexers

If necessary, you can define a Dgidx cluser to apply actions to an entire cluster of Dgidxs, rather than manually iterating over a number of Dgidxs. In addition, the object contains logic associated with executing Dgidxs in parallel based on Dgidx groups, which are described below. Multiple indexing clusters can be defined, with no restriction around which Dgidx belongs to each cluster or how many clusters a Dgidx belongs to.

An indexing cluster is configured with references to all Dgidxs that belong to that cluster. In addition, the cluster can be configured to copy data in parallel or serially. This setting applies to copies that are performed to retrieve source data and configuration to each server that hosts a Dgidx component. By default, the template sets this value to true.

```
<!--
########################################################################
# Indexing Cluster
#
-->
<indexing-cluster id="IndexingCluster" getDataInParallel="true">
  <dgidx ref="Dgidx1" />
  <dgidx ref="Dgidx2" />
</indexing-cluster>
```

In addition to standard Dgidx configuration settings and process arguments, the Deployment Template uses several configurable properties and custom directories during processing:

- `numLogBackups` - Number of log directory backups to store.
- `numIndexbackups` - Number of index backups to store.
- `incomingDataHost` - Host to which source data files are extracted.
- `incomingDataDir` - Directory to which source data files are extracted.
- `incomingDataFileName` - Filename of the source data files that are extracted.
- `configHost` - Host from which configuration files and dimensions are retrieved for Dgidx to process.
- `configDir` - Directory from which configuration files and dimensions are retrieved for Dgidx to process.
- `configFileName` - Filename of the configuration files and dimensions that are retrieved for Dgidx to process.
- `skipTestingForFilesDuringCleanup` - Used for directory-cleaning operations. If set to "true", will skip the directory-contents test and instead proceed directly to cleaning the directory. The default behavior is to test the directory contents and skip cleanup if the directory is not empty.
- The properties documented in the "Fault tolerance and polling interval properties" topic.

In addition to standard Dgidx configuration and process arguments, Dgidx processes add a custom property used to define which Dgidx processes run in parallel with each other when they belong to an indexing cluster.

`dgidxGroup` - Indicates the Dgidx's membership in a Dgidx group. When the run method on an indexing cluster is executed, Dgidx processes within the same Dgidx group are run in parallel. Dgidx group values are arbitrary strings. The indexing cluster iterates through the groups in alphabetical order, though non-standard characters may result in groups being updated in an unexpected order.

CAS crawl configuration

The Deployment Template provides support for running CAS crawls with the CAS Server Component. A CAS Server component is implemented as a `custom-component`. You configure the component according to the output type of a crawl. The sections below describe the common configuration properties, the output-type configuration properties, and then provide examples for each output type including Record Store output, MDEX-compatible output, and record file output.

Note: The Deployment Template cannot create a new CAS crawl. You create a crawl using CAS and run it using the Deployment Template. For details about creating a crawl, see the *CAS Developer's Guide*.

The `custom-component` configuration properties

The `custom-component` configuration properties identify the CAS server in the Servers/hosts section of `AppConfig.xml`. The properties are defined as follows:

- `id` - Assigns a unique ID to a specific CAS Server.
- `host-id` - Points back to the `id` attribute of the `host` global configuration element.
- `class` - Specifies the class that implements the `ContentAcquisitionServerComponent`. Specify class="`com.endeca.eac.toolkit.component.cas.ContentAcquisitionServerComponent`".

Common configuration properties

The common configuration properties describe the host and port running CAS. The properties are defined as follows:

- `casHost` - Host name of the server on which the Content Acquisition System is running.
- `casPort` - Port on which the Endeca CAS Service listens. If the application is running in SSL mode, the `casPort` is the SSL port of the Endeca CAS Service The port number must match the `com.ende¬ca.cas.port` value that is used in the CAS Service configuration script. Or, if the Endeca CAS Service is configured for SSL, then the port number must match `com.endeca.cas.ssl.port` value. The configuration script is in `<install path>\CAS\workspace\conf\jetty.xml`.
- `httpSocketTimeout` is the maximum period of inactivity in milliseconds between two consecutive data packets before http times out.

Configuration properties specific to MDEX-compatible output

The configuration properties for MDEX-compatible output are defined as follows:

- `numPartialsBackups` - Indicates the number of backups to keep for the cumulative partials directory (`cumulativePartialsDir`). If this property is not configured, then no backups are retained.
- `cumulativePartialsDir` - Indicates the directory on the CAS host where partial MDEX output should be accumulated. This allows partial updates to be reapplied in the event of a failure while applying partial updates.
- `numDvalIdMappingsBackups` - Indicates the number of backups to keep for the dimension value ID mappings file. This allows you to restore dimension value ID mappings if the CAS host fails. If this property is not configured, then five backups are retained. If set to zero, then no backing up is performed.
- `dvalIdMappingsArchiveDir` - Indicates the directory where the dimension value ID mappings files are stored. If this property is not configured, then mappings are written to `./data/dvalid_mappings_archive`. However, to provide more secure backups, Oracle recommends that you specify a network drive that is available to CAS but not the same as the CAS host.

Example

This example CAS Server component is configured for MDEX-compatible output:

```
   <!--
    ########################################################################
    # Content Acquisition System Server
    #
    -->
  <custom-component id="CAS" host-id="ITLHost" class="com.endeca.eac.toolkit.com¬
ponent.cas.ContentAcquisitionServerComponent">
    <properties>
      <property name="casHost" value="localhost" />
      <property name="casPort" value="8500" />
      <property name="httpSocketTimeout" value="180000" />
      <property name="numPartialsBackups" value="5" />
      <property name="numDvalIdMappingsBackups" value="5" />
    </properties>
    <directories>
      <directory name="cumulativePartialsDir">./data/partials/cumulative_par¬
tials</directory>
     <directory name="dvalIdMappingsArchiveDir">./data/dvalid_mappings_archive</di¬
rectory>
    </directories>
  </custom-component>
```

Configuration properties specific to Record Store output

There are no additional configuration properties required for crawls that write to a Record Store instance. Only the `custom-component` and common configuration properties are required.

Example

This example CAS Server component is configured for Record Store output:

```
<!--
########################################################################
# Content Acquisition System Server
#
<custom-component id="CAS" host-id="CASHost" class="com.endeca.eac.toolkit.compo¬
nent.cas.ContentAcquisitionServerComponent">
   <properties>
      <property name="casHost" value="localhost" />
      <property name="casPort" value="8500" />
      <property name="httpSocketTimeout" value="180000" />
    </properties>
</custom-component>
-->
```

Configuration properties specific to record file output

The configuration properties are defined as follows:

- `casCrawlFullOutputDestDir` - Indicates the destination directory to which the crawl output file will be copied after a baseline crawl. Note that this is not the directory to which the CAS crawl writes its output; that output directory is set as part of the crawl configuration.
- `casCrawlIncrementalOutputDestDir` - Indicates the destination directory to which the crawl output file will be copied after an incremental crawl. As with the previous property, this is not the directory to which the CAS crawl writes its output. If you run incremental crawls, the default settings assume that the output format will be compressed binary files.

- `casCrawlOutputDestHost` - Indicates the ID of the host on which the destination directories (specified by the previous two properties) reside.

Example

This example CAS Server component is configured for a record file output:

```
<!--
    ###########################################################################
    # Content Acquisition System Server
    #
  -->
  <custom-component id="CAS" host-id="CASHost" class="com.endeca.soleng.eac.toolk¬
it.component.ContentAcquisitionServerComponent">
    <properties>
      <property name="casHost" value="localhost" />
      <property name="casPort" value="8500" />
      <property name="httpSocketTimeout" value="180000" />
      <property name="casCrawlFullOutputDestDir" value="./data/com¬
plete_cas_crawl_output/full" />
      <property name="casCrawlIncrementalOutputDestDir" value="./data/com¬
plete_cas_crawl_output/incremental" />
      <property name="casCrawlOutputDestHost" value="CASHost" />
    </properties>
  </custom-component>
```

Endeca Configuration Repository

The IFCR is a custom component that specifies user information for an Endeca Configuration Repository that is running inside Oracle Endeca Workbench. The deployment template scripts use the information to connect to an Endeca Configuration Repository and move configuration used by Authoring and Live Dgraphs, the media MDEX reference application, and the IFCR Backup Utility.

You define an IFCR component in the `WorkbenchConfig.xml` file which is then referenced by `AppConfig.xml`.

The `custom-component` configuration properties

The `custom-component` configuration properties identify the IFCR in the Data Ingest Hosts section of `DataIngest.xml`.

The properties are defined as follows:

- `id` - Assigns a unique ID to a specific IFCR instance.
- `host-id` - Points back to the `id` attribute of the `host` global configuration element.
- `class` - Specifies the class that implements the `IFCRComponent`. Specify class="`com.ende¬ca.soleng.eac.toolkit.component.IFCRComponent`".

IFCR configuration properties

The configuration properties are defined as follows:

- `repositoryUrl` - Specifies host, port, and ifcr directory as `http://<workbench host>:<port>/ifcr`.
- `username` - Name of the user logging in to Oracle Commerce Workbench where the Endeca Configuration Repository is hosted.
- `password` - Corresponding password for the user name.

- numExportBackups - Indicates the number of backups to keep for exported configuration of the Endeca Configuration Repository. If this property is not configured, then no backups are retained. The default value is 5.

Example

This example shows a typical configuration:

```
  <!--
    ########################################################################
    # IFCR - A component that interfaces with the Workbench repository.
  -->
  <custom-component id="IFCR" host-id="ITLHost" class="com.endeca.soleng.eac.toolk¬
it.component.IFCRComponent">
    <properties>
      <property name="repositoryUrl" value="http://localhost:8006/ifcr" />
      <property name="username" value="admin" />
      <property name="password" value="admin" />
      <property name="numExportBackups" value="3" />
    </properties>
  </custom-component>
```

Workbench Manager

The Workbench Manager is a custom component that specifies connection information for Oracle Commerce Workbench and also a configuration directory for Oracle Commerce Workbench. The deployment template scripts use the information to connect to Workbench and update shared configuration contained in the configuration directory.

You define a Workbench Manager component in the WorkbenchConfig.xml file which is then referenced by AppConfig.xml.

The custom-component configuration properties

The custom-component configuration properties identify the Workbench Manager in the Data Ingest Hosts section of DataIngest.xml.

The properties are defined as follows:

- id - Assigns a unique ID to a specific Workbench instance.
- host-id - Points back to the id attribute of the host global configuration element.
- class - Specifies the class that implements the WorkbenchManagerComponent. Specify class="com.endeca.soleng.eac.toolkit.component.WorkbenchManagerComponent".

Workbench configuration properties

The configuration properties are defined as follows:

- workbenchHost - Host name of the server on which Oracle Commerce Workbench is running.
- workbenchPort - Port on which Workbench listens. This is the port of the Endeca Tools Service on the Oracle Commerce Workbench host. If the application is running in SSL mode, the workbenchPort is the SSL port of Workbench.
- configDir - Directory to which Workbench configuration files are uploaded or downloaded by other components in the implementation.
- workbenchTempDir - Temporary directory used for Workbench interaction. Post-Forge dimensions are uploaded or downloaded from this directory by other components in the implementation.

Example

This example shows a typical configuration:

```
  <!--
    ##############################################################################
    # WorkbenchManager - A component that interfaces with the legacy
    # 'web studio' configuration repository. It is used primarily during
    # data ingest to load post-forge dimensions into Workbench.
  -->
  <custom-component id="WorkbenchManager" host-id="ITLHost" class="com.ende¬
ca.soleng.eac.toolkit.component.WorkbenchManagerComponent">
    <properties>
      <property name="workbenchHost" value="localhost" />
      <property name="workbenchPort" value="8006" />
    </properties>
    <directories>
      <directory name="configDir">./config/pipeline</directory>
      <directory name="workbenchTempDir">./data/workbench/temp</directory>
    </directories>
  </custom-component>
```

Reporting

Oracle Commerce Workbench provides an interface for viewing and analyzing reports produced by the Report Generator.

In order for Oracle Commerce Workbench to display these reports, report files and associated charts need to be created and delivered to a directory in Oracle Commerce Workbench's workspace. Alternatively, a "`web¬studio`" host can be provisioned with a "`webstudio-report-dir`" custom directory, which indicates to Oracle Commerce Workbench where it should read reports for the application. In addition, the files need to be named with a date stamp to conform to Oracle Commerce Workbench's naming convention. The Deployment Template includes report generation scripts that perform these naming and copying steps to deliver reports for Oracle Commerce Workbench to read. Common extension or customization of this functionality may occur when one or more of the components in the reporting lifecycle run in different environments. The `AppConfig.xml` allows components to work independently of each other. Specifically, the LogServer can be configured to deliver files to an arbitrary directory, from where the files can be copied to another environment for report generation. Similarly, the Report Generator's output report can be delivered to an arbitrary target directory, from where the files can be copied to another environment for display in Oracle Commerce Workbench.

Logging and reporting configuration

Four report generator components are defined.

In addition to standard Report Generator configuration settings and process arguments, the Deployment Template uses a configurable property for log archiving, as well as these configurable properties:

- `skipTestingForFilesDuringCleanup` - Used for directory-cleaning operations. If set to "true", will skip the directory-contents test and instead proceed directly to cleaning the directory. The default behavior is to test the directory contents and skip cleanup if the directory is not empty.
- The properties documented in the "Fault tolerance and polling interval properties" topic.

The configuration file includes the name of an output file for each report generator, which defaults to `report.html` or `report.xml`. This file name is never used when the report generation scripts in the `AppConfig.xml` file are used. During execution, the script re-provisions the report generator to output a file named with a date stamp. This means that the provisioning in the file will always be "out of synch" with the

provisioning in the EAC. This will result in the Report Generator's definition changing repeatedly as scripts are executed.

```
<report-generator id="WeeklyReportGenerator" host-id="ITLHost">
    <log-dir>./logs/report_generators/WeeklyReportGenerator</log-dir>
    <input-dir>./reports/input</input-dir>
    <output-file>./reports/weekly/report.xml</output-file>
    <stylesheet-file>
      ./config/report_templates/tools_report_stylesheet.xsl
    </stylesheet-file>
    <settings-file>
      ./config/report_templates/report_settings.xml
    </settings-file>
    <time-range>LastWeek</time-range>
    <time-series>Daily</time-series>
    <charts-enabled>true</charts-enabled>
  </report-generator>
```

Log servers

In addition to standard LogServer configuration settings and process arguments, the Deployment Template uses a configurable property for log archiving.

- `numLogBackups` - Number of log directory backups to store.
- `shutdownTimeout` - Number of seconds to wait for a component to stop (after receiving a stop command).
- `numIdleSecondsAfterStop` - Number of seconds to pause/sleep after a component is stopped. Typically, this will be used to ensure that log file locks are release by the component before proceeding.
- `targetReportGenDir` - Directory to which logs will be copied for report generation.
- `targetReportGenHostId` - Host to which logs will be coped for report generation.
- `skipTestingForFilesDuringCleanup` - Used for directory-cleaning operations. If set to "true", will skip the directory-contents test and instead proceed directly to cleaning the directory. The default behavior is to test the directory contents and skip cleanup if the directory is not empty.
- The properties documented in the "Fault tolerance and polling interval properties" topic.

```
<logserver id="LogServer" host-id="ITLHost" port="15010">
  <properties>
    <property name="numLogBackups" value="10" />
    <property name="targetReportGenDir" value="./reports/input" />
    <property name="targetReportGenHostId" value="ITLHost" />
  </properties>
  <log-dir>./logs/logservers/LogServer</log-dir>
  <output-dir>./logs/logserver_output</output-dir>
  <startup-timeout>120</startup-timeout>
  <gzip>false</gzip>
</logserver>
```

Fault tolerance and polling interval properties

Two sets of configurable properties set the behavior of the Deployment Template fault tolerance mechanism and the frequency of status checks for components.

Fault tolerance property

You can now configure fault tolerance (i.e., retries) for any component (such as Forge, Dgidx, and Dgraph) when invoked through the EAC. This functionality also extends to the CAS server when running a crawl with the CAS component. The name of the fault-tolerance property is `maxMissedStatusQueriesAllowed`.

When components are run, the Deployment Template instructs the EAC to start a component, then polls on a regular interval to check if the component is running, stopped, or failed. If one of these status checks fails, the Deployment Template assumes the component has failed and the script ends. The `maxMissedStatus¬ QueriesAllowed` property allows a configurable number of consecutive failures to be tolerated before the script will end.

The following is an example of a Forge component configured to tolerate a maximum of ten consecutive failures:

```
<forge id="Forge" host-id="ITLHost">
  <properties>
    <property name="numStateBackups" value="10"/>
    <property name="numLogBackups" value="10"/>

    <property name="maxMissedStatusQueriesAllowed" value="10"/>
  </properties>
  ...
</forge>
```

The default number of allowed consecutive failures is 5. Note that these status checks are consecutive, so that every time a status query returns successfully, the counter is reset to zero.

Keep in mind that you can use different fault-tolerance settings for your components. For example, you could set a value of 10 for the Forge component, a value of 8 for Dgidx, and a value of 6 for the Dgraph.

Polling interval properties

As described in the previous section, the Deployment Template polls on a regular interval to check if a started component is running, stopped, or failed. A set of four properties is available to configure each component for how frequently the Deployment Template polls for status while the component is running. Because each property has a default value, you can use only those properties that are important to you.

The polling properties are as follows:

- `minWaitSeconds` specifies the threshold (in seconds) when slow polling switches to standard (regular) polling. The default is **-1** (i.e., no threshold, so the standard polling interval is used from the start).
- `slowPollingIntervalMs` specifies the interval (in milliseconds) that status queries are sent as long as the `minWaitSeconds` time has not elapsed. The default slow polling interval is **60** seconds.
- `standardPollingIntervalMs` (specified in milliseconds) is used after the `minWaitSeconds` time has passed. If no `minWaitSeconds` setting is specified, the `standardPollingIntervalMs` setting is always used. The default standard polling interval is **1** second.
- `maxWaitSeconds` specifies the threshold (in seconds) when the Deployment Template gives up asking for status and assumes that it has failed. The default is **-1** (i.e., no threshold, so the Deployment Template will keep trying indefinitely).

Here is an example configuration for a long-running Forge component that typically takes 8 hours to complete:

```
<forge id="Forge" host-id="ITLHost">
  <properties>
    <property name="numStateBackups" value="10"/>
    <property name="numLogBackups" value="10"/>

    <property name="standardPollingIntervalMs" value="60000"/>
    <property name="slowPollingIntervalMs" value="600000"/>
    <property name="minWaitSeconds" value="28800"/>
    <property name="maxMissedStatusQueriesAllowed" value="10"/>
  </properties>
  ...
</forge>
```

The result of this configuration would be that for the first 8 hours (`minWaitSeconds`=28800), Forge's status would be checked every 10 minutes (`slowPollingIntervalMs`=600000), after which time the status would be checked every minute (`standardPollingIntervalMs`=60000). If a status check fails, a maximum of 10 consecutive retries will be attempted, based on the `standardPollingIntervalMs` setting.

Keep in mind that these values can be set independently for each component.

Fault tolerance and polling interval for utilities

Fault tolerance and polling interval values can also be set for these utilities:

- copy
- shell
- archive
- rollback

You set the new values by adjusting the BeanShell script code that is used to construct and invoke the utility. You adjust the code by using these setter methods from the EAC Toolkit's `Utility` class:

- `Utility.setMinWaitSeconds()`
- `Utility.setMaxWaitSeconds()`
- `Utility.setMaxMissedStatusQueriesAllowed()`
- `Utility.setPollingIntervalMs()`
- `Utility.setSlowPollingIntervalMs()`
- `Utility.setMaxMissedStatusQueriesAllowed()`

If you do not use any of these methods, then the utility will use the default values listed in the two previous sections.

For example, here is a default utility invocation in the CAS crawl scripts:

```
// create the target dir, if it doesn't already exist
mkDirUtil = new CreateDirUtility(CAS.getAppName(),
   CAS.getEacHost(), CAS.getEacPort(), CAS.isSslEnabled());
mkDirUtil.init(Forge.getHostId(), destDir, CAS.getWorkingDir());
mkDirUtil.run();
```

You would then add these methods before calling the `run()` method, so that the code would now look like this:

```
// create the target dir, if it doesn't already exist
mkDirUtil = new CreateDirUtility(CAS.getAppName(),
   CAS.getEacHost(), CAS.getEacPort(), CAS.isSslEnabled());
mkDirUtil.init(Forge.getHostId(), destDir, CAS.getWorkingDir());
mkDirUtil.setMinWaitSeconds(30);
mkDirUtil.setMaxWaitSeconds(120);
mkDirUtil.setMaxMissedStatusQueriesAllowed(10);
mkDirUtil.setPollingIntervalMs(5000);
mkDirUtil.setSlowPollingIntervalMs(30000);
mkDirUtil.run();
```

Alternatively, if your utility was defined in your `AppConfig.xml` like this:

```
<copy id="MyCopy" src-host-id="ITLHost" dest-host-id="MDEXHost" recursive="true">

   <src>./path/to/files</src>
   <dest>./path/to/target</dest>
</copy>
```

You would add the same type of lines as above, before calling the `run()` method; for example:

```
MyCopy.setMaxMissedStatusQueriesAllowed(10);
 MyCopy.run();
```

For more information on the `Utility` methods, see the Javadocs for the EAC Toolkit package.

Replacing the Default Forge Pipeline

This chapter describes how to modify or create a Forge pipeline that is designed for use within the deployment template operational structure. This includes pipeline naming requirements, common errors encountered, etc.
Note: This chapter only applies to applications that use Forge to process source data. If your application uses CAS to produce MDEX-compatible output, this chapter does not apply.

About the XML configuration files

You use Developer Studio to manage the subset of application configuration that isn't exposed through tooling in Workbench. Deveoper Studio allows you to manage the XML configuration files that control aspects of the behavior of the Forge, Dgidx, and Dgraph processes.

The XML configuration files are documented in detail in the *Endeca XML Reference*.

For reference, the following aspects of an application are configured through Workbench:

- **Business Rules** — modify with the **Experience Manager** tool in Workbench.
- **Thesaurus entries** — modify with the **Thesaurus** tool in Workbench.
- **Keyword redirects** — modify with the **Keyword Redirects** tool in Workbench.
- **Phrasing** — modify with the **Automatic Phrasing** tool in Workbench.

Generally, Workbench is the system of record for configuring any aspect of an application that is exposed through Workbench tools. Use Developer Studio to configure features that are not exposed through Workbench, such as your Forge pipeline, dimension search behavior, and available search interfaces.

Note: Use Workbench or Developer Studio to modify your application configuration whenever possible. Only modify the XML files directly when required in unusual situations. If this is necessary, avoid writing Byte Order Marks (BOMs) into the files.

About the sample pipelines

For testing purposes, the Deployment Template includes a Developer Studio project with two Forge pipelines (a baseline and a partial). The sample pipelines facilitate testing the deployment template; however, the files should be replaced with project-specific files immediately after a deployed application has been properly configured.

The pipelines are located in `<app dir>/config/pipeline`. The pipeline for a baseline update processes 10 records, and the pipeline for a partial update that adds 2 more records.

Sample pipeline overview

This section describes the high-level steps that are necessary to integrate a new/existing pipeline with a deployment template.

Additional detail on each of these steps is provided in later sections.

1. Ensure that the application name and pipeline configuration prefix match the data prefix configured in the deployment template.
2. Place pipeline configuration files in the `<app dir>/config/pipeline/` directory of the primary server.
3. In order to enable partial updates, ensure that the project is configured with a record spec (i.e., a unique record identifier property).
4. Ensure that any input Record Adapters requiring filenames specify the file location relative to the `<app dir>/data/processing/` (or `<app dir>/data/partials/processing`) directory.

Specifying a pipeline

By default, the Deployment Template checks the `<app dir>/config/pipeline` for the pipeline to run. This includes baseline updates and partial updates. It is simplest to put your pipeline files in this directory. Alternatively, the `devStudioConfigDir` attribute in the `ConfigManager` custom component specifies the pipeline to run.

To specify a pipeline to run in `AppConfig.xml`:

1. Ensure that your pipeline files are located in `<app dir>/config/pipeline`.
2. Alternatively, modify the `devStudioConfigDir` property in the `ConfigManager` custom component to reference the pipeline directory.
   In this example, the pipeline is stored in the `pipeline` directory:

```
<custom-component id="ConfigManager" host-id="ITLHost"
  class="com.Endeca.soleng.eac.toolkit.component.ConfigManagerComponent">
  <properties>
  ...
  </properties>
  <directories>
    <directory
      name="devStudioConfigDir">./config/pipeline
    </directory>
    ...
  </directories>
```

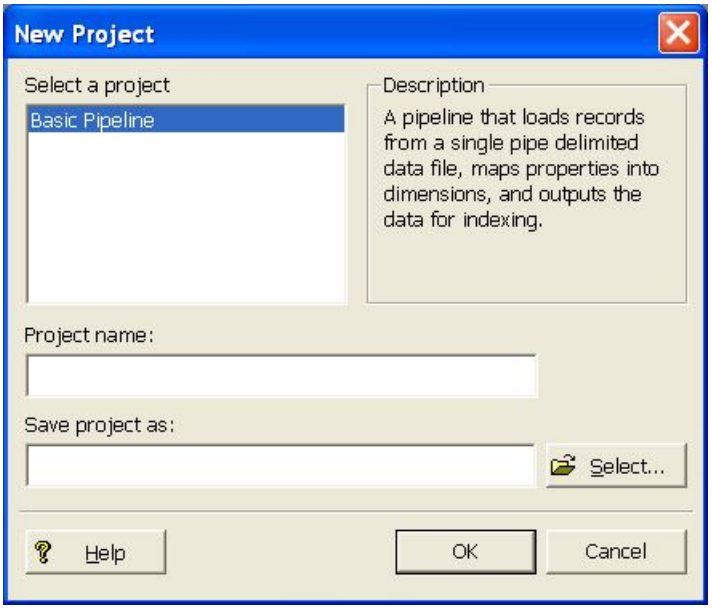
3. If you modified the value in step 2, also modify the value of the `configDir` attribute in the Partial update Forge section to reference the `config/pipeline` directory.
   For example:

```
<!—
###############################################################################
# Partial update Forge
-->
<forge id="PartialForge" host-id="ITLHost">
  <properties>
  ...
  </properties>
  <directories>
    ...
    <directory name="configDir">./config/pipeline</directory>
    ...
  </directories>
```

Creating a new project

Once the reference configuration files have been deleted, a new pipeline configuration project can be created.

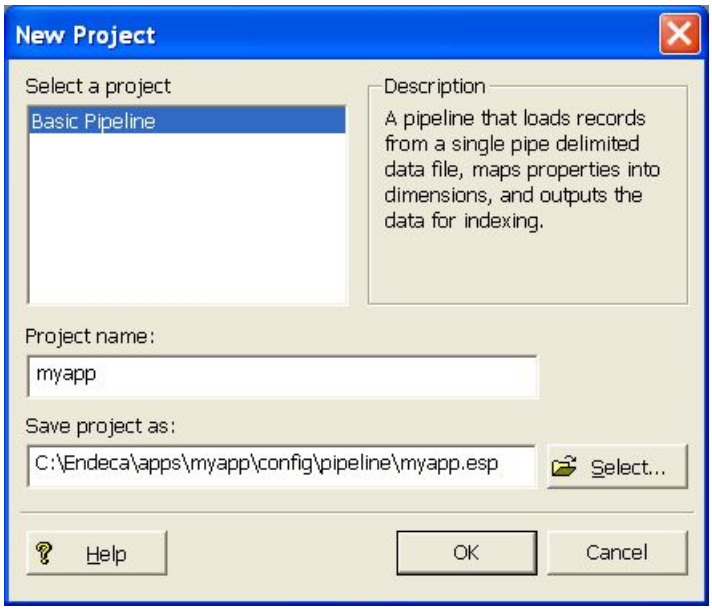
When creating a new project using the Oracle Commerce Developer Studio, you are prompted with the following dialog box:



To create a new project:

1. In order for a new pipeline to be run properly within the deployment template, the following must be properly specified:
   a) The **Project Name** field must be the same as the data prefix specified for the "app" element in <app dir>/config/script/AppConfig.xml. By default, this data prefix will have been set to the name of the application that was specified when running deploy.bat or deploy.sh.
   b) Recall that the [appname] specified was also used to create the base <app dir> directory. For example, if "myapp" was supplied as the [appname], and "c:\Endeca\apps" was supplied as the Deployment Directory, then <app dir> would be c:\Endeca\apps\myapp. In this example, the Project Name should also be specified as "myapp".
2. The **Save Project As** field should be <app dir>\config\pipeline\[appname].esp

   In the example above, the **Save Project As** field would be c:\Endeca\apps\myapp\config\pipeline\myapp.esp.



After clicking the "**OK**" button, a number of files are created in the `<app dir>/config/pipeline/` directory. The primary files to be concerned with are listed below:

File name	Description
`pipeline.epx`	This is the main pipeline file that the deployment template will reference when running forge.
`[appname].esp`	This is the Developer Studio project file that will be used whenever reopening the project. Although this file does not actually require the `[appname]` prefix, it is good practice to keep it consistent with other project files.
`[appname].*.xml`	These are the various configuration files that will be used later by the indexer and MDEX Engine processes. It is important that they have the same prefix as the deployment template `Application Name`.
`dimensions.xml`	This is the dimension file referenced by the default Dimension Adapter.

Modifying an existing project

Modifying an existing Developer Studio project to match a new deployment template application is a somewhat tedious task. In fact, it is often easier to simply create a new deployment template application instead.

The important key is that the `[appname].*.xml` files share the same `[appname]` as the deployment template project. Since there are 30+ XML files, you can either:

- Rename each of the XML files with a new prefix, and update the [appname].esp file to reference each new file.

- Update the deployed application's `AppConfig.xml` file to specify the `[appname]` of your configuration files. For example, if your configuration files are named `myapp.*.xml`, update the configuration as follows:

```
<app appName="myapp" eacHost="host1.company.com" eacPort="8888"
    dataPrefix="myapp" sslEnabled="false"
    lockManager="LockManager">
  <working-dir>C:\Endeca\apps\myapp</working-dir>
  <log-dir>./logs/baseline</log-dir>
</app>
```

In most cases, the `appName` attribute and the `dataPrefix` attribute will be identical. However, this is not required and an application can be configured to support files with a data prefix other than the application name. If the data prefix is not specified, the application defaults to using the application name.

Note that opening an existing project in the Oracle Endeca Developer Studio and using the **Save As** feature will not rename the corresponding `*.xml` files. It will only rename the `[appname].esp` file. The prefix for the XML files can only be specified when a new project is created.

Configuring a record specifier

The deployment includes support for both baseline and partial index updates. In order to support partial updates, an application must include a record specifier, which is a property marked as the unique identifier of records in the index.

For details about the record specifier property, refer to the *Platform Services Forge Guide*.

When configuring your application, identify a property for which each record will have a unique assigned value.

To enable the use of that property as a record spec:

1. Open the **Property** dialog box in Developer Studio.
2. Check the box labeled "**Use for record spec**."

Forge flags

In order to reduce the amount of configuration required to integrate a pipeline into a deployment template, a standard deployment template application runs the primary and partial update Forge processes with an abbreviated set of flags.

Since the deployment template already specifies directory structures and file prefixes, the following flags are used to override a pipeline's input and output components, specifying the appropriate directories and prefixes for either reading or writing data.

Primary Forge flags

Flag	Description
`--inputDir`	`<app dir>/data/processing`
`--stateDir`	`<app dir>/data/state`
`--tmpDir`	`<app dir>/data/forge_temp`
`--logDir`	`<app dir>/logs/baseline`
`--outputDir`	`<app dir>/data/forge_output`
`--outputPrefix`	`[dataPrefix]`

Partial update Forge flags

Flag	Description
`--inputDir`	`<app dir>/data/partials/processing`
`--stateDir`	`<app dir>/data/state`
`--tmpDir`	`<app dir>/data/forge_temp`
`--logDir`	`<app dir>/logs/partial`
`--outputDir`	`<app dir>/data/partials/forge_output`
`--outputPrefix`	`[dataPrefix]`

Input record adapters

The record adapters load the source data.

To start, here is a quick review of how sample data included with the deployment template is processed. The sample application includes a sample dataset in `<app dir>/test_data/baseline` directory. When processing the sample data, the `load_baseline_test_data` script copies the contents of this directory into the `<app dir>/data/incoming/` directory and sets a flag in the EAC.

This flag, named `baseline_data_ready`, indicates to the deployment template scripts that the data extraction process is complete and data is ready for processing. Once that has occurred, the baseline update process copies these files into the `<app dir>/data/processing` directory before running the Forge process.

When using a default deployment template application, it is therefore necessary for all input record adapters to look in the `<app dir>/data/processing` directory for incoming data extracts. The deployment template handles this automatically by specifying the `--inputDir` flag when running the primary forge process. This flag overrides any absolute path specified for specific input adapters with the proper deployment template path:

<app dir>/data/processing. However, the --inputDir flag respects relative paths, resolving them relative to the path specified as the input directory.

The URL property of any record adapter component therefore only needs to specify the relative path to a specific file or subdirectory within the <app dir>/data/incoming directory. (Remember that files and subdirectories in the incoming directory are copied to the processing directory by the deployment template before Forge is run.)

For example, if a single extract file called data.txt is copied into the <app dir>/data/incoming directory before running a baseline, the URL property of that data's input record adapter should specify a URL of data.txt.

For a more complex deployment where, for instance, multiple text extract files are copied into the <app dir>/data/incoming/extracted_data directory before running a baseline update, the URL property of a single input record adapter configured to read these files should be set to extracted_data/*.txt.

Dimension adapters

The --inputDir flag specified to forge overrides the input URL for dimension adapters.

Since the dimensions for a project are usually stored in the <app dir>/config/pipeline directory along with other configuration files, the deployment template copies these files into the <app dir>/data/processing/ directory before running the Forge process. The URLs specified in dimension adapters should follow the same rules as those described for input record adapters, specifying dimension XML file URLs relative to the --inputDir directory. In most cases, this is as simple as specifying the URL for the main dimension adapter as Dimensions.xml, which is the value used by the default "Dimensions" adapter created by Developer Studio's project template.

More complex deployments that include multiple dimension adapters or external delivery of dimension files should ensure that the dimension XML files are copied into the <app dir>/data/incoming/ directory before the forge process runs.

Indexer adapters

Because the --outputPrefix and --outputDir flags are both included, the deployment template will override any values specified for the Indexer Adapter "URL" and "Output prefix" properties.

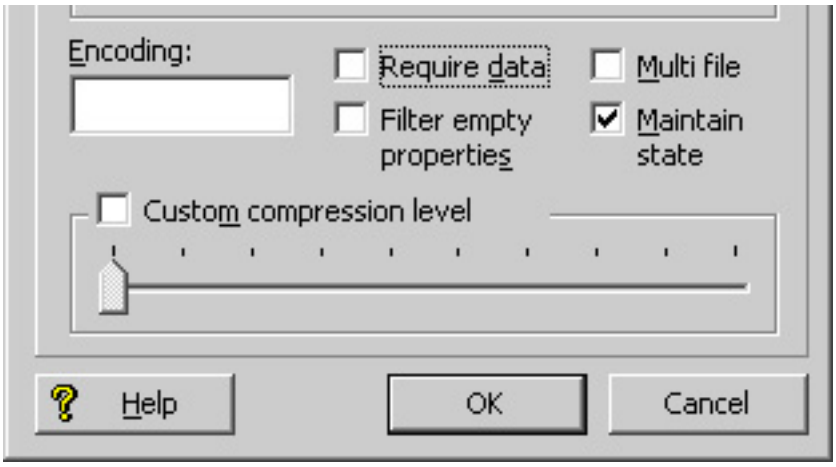
Therefore, it is unnecessary to modify these properties in most cases.

Output record adapters

Output record adapters are often used to generate debug or state information. By default, the location to which this data is written will be overridden by the --outputDir flag.

In most cases, however, it is undesirable for these files to be written to the same location as the Forge output files.

In these cases, an output record adapter can be configured to instead respect the --stateDir flag by selecting the "**Maintain State**" checkbox.



Now any files generated by this output record adapter will be written to the `<app dir>/data/state/` directory.

Note that the output file name must still be specified in the "URL" property of the record adapter. The `--outputPrefix` flag only overrides the indexer adapter output file names, not output record adapter file names.

Dimension servers

The `--stateDir` flag will override the URL value for all Dimension Server components, and place any autogen state files in the `<app dir>/data/state/` directory.

Common errors

This section provides troubleshooting information for commonly received errors.

Unable to Find Pipeline.epx

If Forge fails, check the logs (`<app dir>/logs/baseline/err.forge`) to make sure that Forge was able to find the `pipeline.epx` file in its proper location. Remember that a basic deployment template application assumes that it will find the project's `pipeline.epx` file in `<app dir>\config\pipeline\`.

On UNIX platforms, file names are case sensitive. The deployment template expects the primary pipeline file to be named `pipeline.epx` and the partial update pipeline (if one is required for the deployed application) to be named `partial_pipeline.epx`. Ensure that the files in your deployment use this capitalization.

Missing Configuration Files

This more common error is also more difficult to detect. Since all pipelines created by the Oracle Commerce Developer Studio typically contain a `Pipeline.epx` file, it is unlikely that the Forge process will be unable to find the file, unless it was placed in the wrong directory. If the XML configuration files, however, have a different prefix from the deployment template `[appname]`, these files will not be copied into the `<app dir>/data/forge_output/` , `<app dir>/data/dgidx_output/`, and `<app dir>/data/dgraphs/*/dgraph_input/` directories. All processes will likely complete successfully, but any configuration information specified by these XML files, such as search interfaces, business rules, sort keys, etc. will be missing from the resulting MDEX Engine. To correct this problem, check the XML files located in `<app dir>/config/pipeline/` and make sure they have the correct prefix. Also check the directories mentioned above to make sure that these XML files are being properly copied.

MDEX Engine Fails to Start

If an MDEX Engine fails to start, check the log for the appropriate Dgraph in <app dir>/logs/dgraphs/[dgraph]/[dgraph].log. If the log indicates that the Dgraph failed to start because no record specifier was found, follow the steps in this document to create a unique record specifier property for you project.

Record Adapter Unable to Open File

Another common error may occur if a record adapter is unable to find or open a specified file for either input or output. In this case, the Forge error log (<app dir>/logs/baseline/err.forge) should specify which file or directory could not be found. To correct this problem, make sure the files or directories specified by the record adapters correspond to the directory structure established by the deployment template application. Note that this error may be masked if the "Require Data" property is not checked for a given input adapter, since Forge will only log a warning instead of a fatal error.

Modifying Index Configuration for an Application

This section describes how to modify index configuration using the Index Configuration Command-line Utility in a CAS-based processing model. If you are using Forge to process updates, this chapter does not apply to your deployment.

Overview of the Index Configuration Command-line Utility

The Index Configuration Command-line Utility modifies index configuration stored in the Endeca Configuration Repository for an application. This utility is typically used to modify data after it has been exported from a product catalog system and modify the search configuration settings for the data. In many cases, the utility is also used to manually create index configuration that is not part of a product catalog system.

The Index Configuration Command-line Utility is a script named index-config-cmd that you run from a command prompt. After you deploy a new application, the index-config-cmd script is available in the <app name>\control directory.

Help options

The Index Configuration Command-line Utility has two help options that display the usage syntax. The --help option displays a summary of the tasks. The --help-detail option displays detailed usage information for a specified task. For example:

```
C:\Endeca\apps\Discover\control>index_config_cmd.bat --help
usage: index-config-cmd <task-name> [options]
where <task-name> is one of the following:
   get-config
   set-config
   delete-owner
   get-merged-config
For detailed usage information for individual task options, use <task-name> --
help
```

Command-line options

The command syntax for executing the tasks is:

```
index_config_cmd <task-name> [options]
```

The <*task-name*> argument is the task to be performed by the utility, such as the `get-config` task. The task options vary, depending on the task. However, the following option can be used with any task:

- `-o` (or `--owner`) specifies an import owner for a task. If you specify the `-o` option, the task applies only to the owner specified. The option can have an argument of `all`, `system`, or user-specified owner name. The `all` owner includes both the `system` owner and all user-specified owners. If you omit this option, the task applies to the `system` owner.

About index configuration ownership

The index configuration for an application is associated with one or more *import owners*. An import owner provides a way of indicating that a portion of an application's configuration came from one source rather than another source. The name of the import owner typically identifies the source. For example, the name of an import owner could reflect a product catalog system.

In addition to any number of import owners, there is also one default `system` owner. The `system` owner is typically a developer who uses the utility to augment index configuration and troubleshoot data issues as part of update processing.

Creating and deleting import owners

An import owner is typically created during the data import operation using the Endeca Configuration Import API. The owner name is specified as argument to the `ConfigRepositoryImporter` constructor. For details, see the *Endeca Configuration Import API Reference (Javadoc)*. You can also create an import owner using the `-o` option of the `set-config` task. If the owner does not already exist, the utility creates it.

The index configuration associated with an owner is removed using the `delete-owner` task.

Examining index configuration for an owner

You can retrieve and examine index configuration for an owner using the `get-config` task and the `-o` option. The task returns a JSON file with the index configuration for the specified owner. If the file contains configuration from multiple import owners, each import owner's configuration is represented as a node within the file.

Overwriting index configuration per owner

As described in *Setting the index configuration for an application* on page 156, the `set-config` task overwrites all previous index configuration for a specified owner.

Setting global configuration

Only the `system` owner can set global index configuration.

Merging index configuration from multiple owners

During baseline update processing, the Content Acquisition System merges and processes index configuration from all owners into a consolidated set of MDEX-compatible output files.

If multiple import owners modify the same attribute, the configuration from the `system` owner always overrides other import owners during the merge process.

For example, suppose an import owner named `ATG` creates an attribute that represents an Endeca property. Then the `system` owner updates the Endeca property to become an Endeca dimension with `isAutogen` set to `true`. The merged configuration processes the attribute and updates it to become an autogen dimension.

About the schema for the index configuration file

The Index Configuration Command-line Utility writes and reads index configuration as JSON. The schema for the JSON file varies depending on whether you retrieve configuration for one owner or more than one owner and whether you restrict the types of configuration that you retrieve.

Types of configuration include:

- Endeca properties, derived properties, and dimensions. These are specified under the `attributes` node.
- Precedence rules. These are specified under the `precedenceRules` node.
- Search configuration. These are specified under the `searchIndexConfig` node.

See the topics and examples below that illustrate the schema for each index configuration type.

Schema for an Endeca property, derived property, or dimension

You specify an Endeca property or dimension as an `attribute` in the index configuration file. Each attribute has a `jcr:primaryType` property with one of the following values:

- `endeca:property` - indicates that the attribute is an Endeca property.
- `endeca:derivedProperty` - indicates that the attribute is a derived property.
- `endeca:dimension` - indicates that the attribute is an Endeca dimension.

Properties, dimensions, and precedence rules must be named uniquely across the index configuration. The only exception to this is where you update a property by setting the `mergeAction` property to `UPDATE`.

Schema for an attribute node that defines an Endeca property

An `attribute` that is an `endeca:property` can have the following schema properties:

Name	Data Type	Description
`isEnabled`	BOOLEAN	Optional. Indicates whether the property is processed by CAS when CAS writes MDEX-compatible output. A value of `true` includes the property during processing; `false` excludes the property. This setting is useful when troubleshooting data issues for specific attributes. If omitted, the default value is `true`.
`isRecordFilter¬able`	BOOLEAN	Optional. Indicates whether the property can be used to filter records. Record filtering presents a subset of the data to the end-user. If omitted, the default value is `false`.
`isRecordSearchEn¬abled`	BOOLEAN	Optional. Specifies whether or not record search should be enabled for this property. Record search finds all records in a Guided Search application that are tagged with an Endeca property that matches a term the user provides. You must enable each property that you want available for record search. If omitted, the default value is `false`.
`isRollupKey`	BOOLEAN	Optional. Indicates whether the property can be used as a rollup key. This allow aggregated records to be based on this Endeca property. If omitted, the default value is `false`.
`isWildcardEn¬abledInRecord¬Search`	BOOLEAN	Optional. Indicates whether wildcard search is enabled for this Endeca property. Wildcard searching allows user queries that contain a wildcard character (*) to match against fragments of words in a property value. You must enable each

Name	Data Type	Description
		property that you want available for wildcard searching. If `isWildcardEnabledInRecordSearch` is set to `true`, then `isRecordSearchEnabled` must also be set to `true` for a property. (Enabling wildcard record search depends on first enabling record search.) If omitted, the default value is `false`.
`mergeAction`	STRING	Optional. The `mergeAction` specifies how to merge the attribute into the index configuration. Valid enumerations are `ADD` and `UPDATE`. Specify a value of `ADD` to merge new attributes that are not already in the system. Specify a value of `UPDATE` to merge changes to an existing attribute. If omitted, the default value of `mergeAction` is `ADD`.
`propertyDataType`	STRING	Optional. The `propertyDataType` enumerates the valid values for the data type of an Endeca property. The valid enumerations are `ALPHA`, `INTEGER`, `DOUBLE`, `GEOCODE`, `DATETIME`, `DURATION`, and `TIME`. Such data types have several uses. Non-alpha properties can be used for range filtering. Temporal properties can be used for record sorting and analytics. If omitted, the default value of `property¬DataType` is `ALPHA`.
`sourceProperty¬Names`	STRING (multi-valued)	Optional. Specifies an explicit mapping between one or more source properties and an Endeca property. An Endeca property is populated with data from the source property that it is mapped to. If specified, the `sourcePropertyNames` value can be empty, single-valued, or multi-valued. If empty (a zero-length list), no source property is mapped to an Endeca property. This allows you to define an Endeca property but not populate it with any data. If single-valued, then the source property has its value mapped to the Endeca property. If multi-valued, then each source property in this list has its value mapped to the Endeca property. If omitted, the source property has its value mapped to an Endeca property of the same name. In other words, a source property with a name that is identical to an Endeca property is automatically mapped to that Endeca property. This is the default behavior.

Schema for an attribute node that defines an Endeca derived property

An `attribute` that is an `endeca:derivedProperty` can have the following schema properties:

Name	Data Type	Description
`derivedProperty¬Function`	STRING	Required. The `derivedPropertyFunction` enumerates the valid functions that can be applied to the `derivedProp¬`

Name	Data Type	Description
		ertySource property. The valid enumerations are MIN, MAX, SUM, and AVG.
derivedProper¬ tySource	STRING	Required. Specifies the Endeca property from which the derived property is calculated.
isEnabled	BOOLEAN	Optional. Indicates whether the property is processed by CAS when CAS writes MDEX-compatible output. A value of true includes the property during processing; false excludes the property. This setting is useful when troubleshooting data issues for specific attributes.
mergeAction	STRING	Optional. The mergeAction specifies how to merge the attribute into the index configuration. Valid enumerations are ADD and UPDATE. Specify a value of ADD to merge new attributes that are not already in the system. Specify a value of UPDATE to merge changes to an existing attribute. If omitted, the default value of mergeAction is ADD.

Schema for an attribute node that defines an Endeca dimension

An attribute that is an endeca:dimension can have the following schema properties:

Name	Data Type	Description
displayOrder	INTEGER	Optional. Specifies the display order of a dimension relative to other dimensions in refinement results. Dimensions with lower values display before dimensions with higher values. Valid values are integers between 0 and 2147483647. If omitted, the dimension displays lower than dimensions with specified display orders. If dimensions have the same display order value (a tie), the dimensions are ordered alphabetically by dimension name.
isAutogen	BOOLEAN	Optional. Specifies whether the dimension values for a dimension are automatically generated during a CAS crawl. A value of true generates dimension values for a dimension. If omitted, the default value is false. (An error results if you set this to true and also specify dimension values for the dimension.)
isEnabled	BOOLEAN	Optional. Indicates whether the dimension is processed by CAS when CAS writes MDEX-compatible output. A value of true includes the dimension during processing; false excludes the dimension. This setting is useful when troubleshooting data issues for specific attributes.
isHierarchi¬ calDimension¬ SearchEnabled	BOOLEAN	Optional. Specifies whether a dimension search also considers ancestor dimension values in this dimension when matching a dimension search query. If omitted, the default value is false.

Name	Data Type	Description
isHierarchical¬RecordSearchEn¬abled	BOOLEAN	Optional. Specifies whether a record search also considers ancestor dimension values in this dimension when matching a record search query. If isHierarchicalRecordSearchEnabled is set to true, then isRecordSearchEnabled must also be set to true for a dimension. (Enabling hierarchical search depends on first enabling record search.) If omitted, the default value of isHierarchicalRecordSearchEnabled is set to the value of isRecordSearchEnabled.
isRecordSearchEn¬abled	BOOLEAN	Optional. Specifies whether or not record search should be enabled for this dimension. Record search finds all records in a Guided Search application that are tagged with a dimension value that matches a term the user provides. You must enable each property that you want available for record search. If omitted, the default value is false.
isWildcardEn¬abledInRecord¬Search	BOOLEAN	Optional. Indicates whether wildcard search is enabled for this dimension. Wildcard searching allows user queries that contain a wildcard character (*) to match against fragments of words in a dimension. You must enable each dimension that you want available for wildcard searching. If isWildcardEnabledInRecordSearch is set to true, then isRecordSearchEnabled must also be set to true for a dimension. (Enabling wildcard dimension search depends on first enabling dimension search.) If omitted, the default value is false.
mergeAction	STRING	Optional. The mergeAction specifies how to merge the attribute into the index configuration. Valid enumerations are ADD and UPDATE. Specify a value of ADD to merge new attributes that are not already in the system. Specify a value of UPDATE to merge changes to an existing attribute. If omitted, the default value of mergeAction is ADD.
multiSelectType	STRING	Optional. The multiSelectType enumerates the valid values for specifying multiselect dimensions. When AND is specified on a dimension, the MDEX Engine returns all records from all the select dimension values. The result set is expanded with each additional dimension value that a user selects. The OR enumeration returns the records from one selected dimension value. The result set is reduced with each dimension value that a user selects. The valid enumerations are NONE, OR, and AND.
rangeComparison¬Type	STRING	Optional. The rangeComparisonType enumerates the types that can be used to map source properties to dimension values that represent ranges. The valid enumerations are STRING, INTEGER, and FLOAT.
sourceProperty¬Names	STRING (multi-valued)	Optional. Specifies an explicit mapping between one or more source properties and an Endeca dimension. A dimension is

Name	Data Type	Description
		populated with data from the source property that it is mapped to. If specified, the `sourcePropertyNames` value can be empty, single-valued, or multi-valued. If empty (a zero-length list), no source property is mapped to a dimension. This allows you to define a dimension but not populate it with any data. This is useful when creating trigger dimension values for content spotlighting cartridges. If single-valued, then the source property has its value mapped to the Endeca dimension. If multi-valued, then each source property in this list has its value mapped to the Endeca dimension. If omitted, the source property has its value mapped to an Endeca dimension of the same name. In other words, a source property with a name that is identical to a dimension is automatically mapped to that dimension. This is the default behavior.

Example index configuration for two owners and all configuration types

In this example, the utility returns index configuration for the `system` owner. The configuration in this case is made up of attributes and global index configuration settings:

```
{
  "indexConfig" : {
    "system" : {
      "attributes" : {
        "product.price" : {
          "propertyDataType" : "DOUBLE",
          "jcr:primaryType" : "endeca:property"
        },
        "product.brand.name" : {
          "isRecordSearchEnabled" : true,
          "isAutogen" : true,
          "jcr:primaryType" : "endeca:dimension",
          "multiSelectType" : "OR"
        },
        "product.review.count" : {
          "propertyDataType" : "INTEGER",
          "jcr:primaryType" : "endeca:property"
        },
        "product.sku" : {
          "isRecordSearchEnabled" : true,
          "propertyDataType" : "ALPHA",
          "jcr:primaryType" : "endeca:property"
        },
        "product.id" : {
          "isRecordFilterable" : true,
          "isRecordSearchEnabled" : true,
          "propertyDataType" : "ALPHA",
          "jcr:primaryType" : "endeca:property"
        },
        "camera.color" : {
          "sourcePropertyNames" : [ "camera.Colour of product" ],
```

```
          "isAutogen" : true,
          "jcr:primaryType" : "endeca:dimension"
        },
        "product.category" : {
          "sourcePropertyNames" : [ "product.category_id" ],
          "isRecordSearchEnabled" : true,
          "jcr:primaryType" : "endeca:dimension"
        },
        "product.name" : {
          "isRecordSearchEnabled" : true,
          "propertyDataType" : "ALPHA",
          "jcr:primaryType" : "endeca:property"
        },
        "product.features" : {
          "isAutogen" : true,
          "jcr:primaryType" : "endeca:dimension",
          "multiSelectType" : "AND"
        },
        "product.min_price" : {
          "derivedPropertySource" : "product.price",
          "derivedPropertyFunction" : "MIN",
          "jcr:primaryType" : "endeca:derivedProperty"
        },
        "product.price_range" : {
          "sourcePropertyNames" : [ "product.price" ],
          "rangeComparisonType" : "FLOAT",
          "jcr:primaryType" : "endeca:dimension"
        },
        "common.id" : {
          "isRecordFilterable" : true,
          "propertyDataType" : "ALPHA",
          "jcr:primaryType" : "endeca:property"
        },
       ...
      },
      "precedenceRules" : {
      },
      "searchIndexConfig" : {
        "spellingDictMinNumWordOccurrences" : 4,
        "spellingDictMaxWordLength" : 16,
        "isWildcardEnabledInDimensionSearch" : true,
        "spellingDictMinWordLength" : 3
      }
    }
  }
}
```

Schema for precedence rules

You specify a precedence rule in the `precedenceRules` node of the index configuration file. The `precedenceRules` node is a sibling of the `attributes` node.

Schema for a precedenceRule node to define a precedence rule

Each node representing a precedence rule under `precedenceRules` can have the following schema properties:

Name	Data Type	Description
isEnabled	BOOLEAN	Optional. Indicates whether the precedence rule is processed by CAS when CAS writes MDEX-compatible output. A value of true includes the precedence rule during processing; false excludes the precedence rule. This setting is useful when troubleshooting data issues for specific attributes.
isLeafTrigger	BOOLEAN	Optional. Specifies a Boolean to indicate if the trigger is a leaf trigger or not. If set to true, the rule only triggers on leaf dimension values.
mergeAction	STRING	Optional. The mergeAction enumerates the valid values that describe how to merge the precedence rule into the index configuration. Valid enumerations are ADD and UPDATE. Specify a value of ADD to merge new precedence rules that are not already in the system. Specify a value of UPDATE to merge changes to an existing precedence rule. If omitted, the default value of mergeAction is ADD.
triggerDimension	STRING	Required. Specifies the trigger dimension for a precedence rule. Recall that a user's selection of the trigger dimension reveals the previously unavailable target dimension to the user.
triggerDimensionVal¬ueSpec	STRING	Optional. Specifies the dimension value specification of the triggerDimension. If omitted, the precedence rule fires for any selection from the trigger dimension.
targetDimension	STRING	Required. Specifies the target dimension for a precedence rule.

Example index configuration for precedence rules

In this example, the utility returns index configuration from one owner, named ATG, and the precedenceRules configuration type.

```
C:\Endeca\apps\Discover\control>index_config_cmd.bat get-config -o ATG -t prece¬
denceRules
[07.26.12 12:57:19] INFO: Using site Discover at URL http://JSMITH-WIN7:8006/
ifcr with username admin
```

The output contains a precedenceRules root node, because that was the specified configuration type, and then child nodes for two precedence rules:

```
{
    "precedenceRules" : {
      "aspectRatioDigitalCamerasRule" : {
        "targetDimension" : "camera.aspect_ratio",
        "triggerDimensionValueSpec" : "575",
        "triggerDimension" : "product.category",
        "isLeafTrigger" : false
      },
      "digitalZoomDigitalCamerasRule" : {
        "targetDimension" : "camera.digital_zoom",
        "triggerDimensionValueSpec" : "575",
        "triggerDimension" : "product.category",
        "isLeafTrigger" : false
      }
      ...
```

```
    }
}
```

Schema for global index configuration

You specify search configuration in the `searchIndexConfig` node of the index configuration file. In this release, the settings control spelling dictionary configuration and wildcard search. The `searchIndexConfig` node is a sibling of the `attributes` and `precedenceRules` nodes.

Schema for a searchIndexConfig node

Each property under `searchIndexConfig` represents an index configuration setting. The following properties are available:

Name	Data Type	Description
`spellingDictMin¬ NumWordOccur¬ rences`	LONG	Optional. Specifies the minimum number of times the word must appear in the source data before the word should be included in the spelling dictionary. This setting applies to record search only. If omitted, the default value is `4`. For dimension search, this setting is always set to `1`. (All dimension value names are included in the spelling dictionary by default.)
`spellingDictMax¬ WordLength`	LONG	Optional. Specifies the maximum length of a word that should be included in the spelling dictionary. Words longer than this value are excluded. This setting applies to both dimension search and record search. If omitted, the default value is `16`.
`isWildcardEn¬ abledInDimension¬ Search`	BOOLEAN	Optional. Specifies a Boolean to indicate that a query can contain a wildcard character (*) to match against fragments of words in a dimension value. If omitted, the default value is `true`.
`spellingDictMin¬ WordLength`	LONG	Optional. Specifies the minimum character length for a word to be included in the spelling dictionary. This setting applies to both dimension search and record search. If omitted, the default value is `3`.

Example index configuration for global settings

In this example, the utility returns index configuration for the `system` owner. The index configuration is restricted to only the `searchIndexConfig` type.

```
C:\Endeca\apps\Discover\control>index_config_cmd.bat get-config -o system -t
searchIndexConfig
[07.26.12 12:57:19] INFO: Using site Discover at URL http://JSMITH-WIN7:8006/
ifcr with username admin
```

The output contains a `searchIndexConfig` root node, because that was the specified configuration type, and then properties for each configuration setting:

```
{
  "searchIndexConfig" : {
    "spellingDictMinNumWordOccurrences" : 4,
    "spellingDictMaxWordLength" : 16,
    "isWildcardEnabledInDimensionSearch" : true,
```

```
    "spellingDictMinWordLength" : 3
  }
}
```

Getting the index configuration for an application

The `get-config` task retrieves the index configuration for an application.

The syntax for this task is:

```
index_config_cmd get-config [-o OwnerName] [-f FileName]
[-r] [-t precedenceRules|attributes|searchIndexConfig]
```

Where:

- `-o` (or `--owner`) specifies an import owner for a task. If you specify the `-o` option, the task applies only to the owner specified. The option can have an argument of `all`, `system`, or user-specified owner. The `all` owner includes both the `system` owner and all import owners. If you omit this option, the task applies to the `system` owner. Optional.
- `-f` (or `--file`) specifies a path to a JSON output file that contains the index configuration. Omitting the `-f` option prints the index configuration to standard out. Optional.
- `-r` (or `--repositoryMetadata`) specifies whether to return metadata about each attribute value in the index configuration. Metadata for an attribute includes properties such as `jcr:lastModifiedBy`, `jcr:createdBy`, `jcr:created`, `jcr:lastModified`, and so on. Optional.
- `-t` (or `--type`) specifies the type of index configuration you want the task to return. The arguments are `precedenceRules`, `attributes`, and `searchIndexConfig`. Specifying `precedenceRules` returns only precedence rules in the index configuration, or none. Specifying `attributes` returns the attributes in the index configuration. Specifying `searchIndexConfig` returns only the global index configuration settings. Omitting the `-t` option returns all types of index configuration. Optional.

Note: There is a size limit on the total number of attributes and precedence rules the task can retrieve. If the index configuration that you are retrieving contains more than approximately 10,000 attributes and precedence rules, the `get-config` task returns a `Multiple Choices (300)` error.

To get the index configuration for an application:

1. Start a command prompt and navigate to `<app dir>\control` (for Windows) or `<app dir>/control` (for UNIX).
2. Type `index_config_cmd` and specify the `get-config` task.

Note: This task name is case sensitive.

Example of getting the index configuration for an application

```
C:\Endeca\apps\Discover\control>index_config_cmd.bat get-config
[07.23.12 15:50:54] INFO: Using site Discover at URL http://JSMITH-WIN7:8006/
ifcr with username admin
{
  "indexConfig" : {
    "system" : {
      "attributes" : {
        "product.price" : {
          "propertyDataType" : "DOUBLE",
          "jcr:primaryType" : "endeca:property"
        },
```

```
        "product.brand.name" : {
          "isRecordSearchEnabled" : true,
          "isAutogen" : true,
          "jcr:primaryType" : "endeca:dimension",
          "multiSelectType" : "OR"
        },
        "product.review.count" : {
          "propertyDataType" : "INTEGER",
          "jcr:primaryType" : "endeca:property"
        },
        "product.sku" : {
          "isRecordSearchEnabled" : true,
          "propertyDataType" : "ALPHA",
          "jcr:primaryType" : "endeca:property"
        },
        "product.id" : {
          "isRecordFilterable" : true,
          "isRecordSearchEnabled" : true,
          "propertyDataType" : "ALPHA",
          "jcr:primaryType" : "endeca:property"
        },
        "camera.color" : {
          "sourcePropertyNames" : [ "camera.Colour of product" ],
          "isAutogen" : true,
          "jcr:primaryType" : "endeca:dimension"
        },
        "product.category" : {
          "sourcePropertyNames" : [ "product.category_id" ],
          "isRecordSearchEnabled" : true,
          "jcr:primaryType" : "endeca:dimension"
        },
        "product.name" : {
          "isRecordSearchEnabled" : true,
          "propertyDataType" : "ALPHA",
          "jcr:primaryType" : "endeca:property"
        },
        "product.features" : {
          "isAutogen" : true,
          "jcr:primaryType" : "endeca:dimension",
          "multiSelectType" : "AND"
        },
        "product.min_price" : {
          "derivedPropertySource" : "product.price",
          "derivedPropertyFunction" : "MIN",
          "jcr:primaryType" : "endeca:derivedProperty"
        },
        "product.price_range" : {
          "sourcePropertyNames" : [ "product.price" ],
          "rangeComparisonType" : "FLOAT",
          "jcr:primaryType" : "endeca:dimension"
        },
        "common.id" : {
          "isRecordFilterable" : true,
          "propertyDataType" : "ALPHA",
          "jcr:primaryType" : "endeca:property"
        },
       ...
      },
      "precedenceRules" : {
      },
```

```
      "searchIndexConfig" : {
        "spellingDictMinNumWordOccurrences" : 4,
        "spellingDictMaxWordLength" : 16,
        "isWildcardEnabledInDimensionSearch" : true,
        "spellingDictMinWordLength" : 3
      }
    }
  }
}
```

Getting the merged index configuration for an application

The `get-merged-config` task retrieves the merged index configuration for an application. In other words, the output of this task is the consolidated index configuration for all import owners.

This task is primarily used as a debugging tool to troubleshoot configuration and data issues in the MDEX-compatible output files produced by a CAS crawl.

In some ways, the `get-merged-config` task is logically similar to `get-config` task with the owner option (`-o`) set to `all`. However, there are several important differences between the two tasks:

- `get-config` outputs configuration that is grouped in nodes by the owner name. `get-merged-config` outputs the consolidated configuration with no distinction for ownership.
- `get-config` does not remove copies of attributes from other owners. For example, if an `ATG` owner adds attribute A and `system` owner updates attribute A, `get-config` returns attribute A in the node for the `ATG` owner and also the node for the `system` owner. Whereas, `get-merged-config` merges the copies of the attribute and returns only one instance of attribute A which is from the `system` owner.

The syntax for this task is:

```
index_config_cmd get-merged-config [-f FileName]
[-t precedenceRules|attributes|searchIndexConfig]
```

Where:

- `-f` (or `--file`) specifies a path to a JSON output file that contains the index configuration. Omitting the `-f` option prints the index configuration to standard out. Optional.
- `-t` (or `--type`) specifies the type of index configuration you want the task to return. The arguments are `precedenceRules`, `attributes`, and `searchIndexConfig`. Specifying `precedenceRules` returns only precedence rules in the index configuration, or none. Specifying `attributes` returns the attributes in the index configuration. Specifying `searchIndexConfig` returns only the global index configuration settings. Omitting the `-t` option returns all types of index configuration. Optional.

To get the merged index configuration for an application:

1. Start a command prompt and navigate to `<app dir>\control` (for Windows) or `<app dir>/control` (for UNIX).
2. Type `index_config_cmd` and specify the `get-merged-config` task.

Note: This task name is case sensitive.

Example of getting the merged index configuration for an application

```
C:\Endeca\apps\Discover\control>index_config_cmd.bat get-merged-config
[08.17.12 11:48:05] INFO: Using site Discover at URL http://JSMITH-WIN7:8006/
ifcr with username admin
```

```
{
  "indexConfig" : {
    "attributes" : {
      "product.price" : {
        "propertyDataType" : "DOUBLE",
        "jcr:primaryType" : "endeca:property"
      },
      "product.brand.name" : {
        "isRecordSearchEnabled" : true,
        "isAutogen" : true,
        "jcr:primaryType" : "endeca:dimension",
        "multiSelectType" : "OR"
      },
      "product.sku" : {
        "isRecordSearchEnabled" : true,
        "propertyDataType" : "ALPHA",
        "jcr:primaryType" : "endeca:property"
      },
      "product.id" : {
        "isRecordFilterable" : true,
        "isRecordSearchEnabled" : true,
        "propertyDataType" : "ALPHA",
        "jcr:primaryType" : "endeca:property"
      },
      "camera.megapixel_range" : {
        "sourcePropertyNames" : [ "camera.Megapixel" ],
        "rangeComparisonType" : "FLOAT",
        "jcr:primaryType" : "endeca:dimension"
      },
      "product.category" : {
        "sourcePropertyNames" : [ "product.category_id" ],
        "isRecordSearchEnabled" : true,
        "jcr:primaryType" : "endeca:dimension"
      },
      "product.name" : {
        "isRecordSearchEnabled" : true,
        "propertyDataType" : "ALPHA",
        "jcr:primaryType" : "endeca:property"
      },
      "product.features" : {
        "isAutogen" : true,
        "jcr:primaryType" : "endeca:dimension",
        "multiSelectType" : "AND"
      },
      "product.min_price" : {
        "derivedPropertySource" : "product.price",
        "derivedPropertyFunction" : "MIN",
        "jcr:primaryType" : "endeca:derivedProperty"
      },
      "product.code" : {
        "isRollupKey" : true,
        "isRecordSearchEnabled" : true,
        "jcr:primaryType" : "endeca:property",
        "propertyDataType" : "ALPHA"
      },
      "product.price_range" : {
        "sourcePropertyNames" : [ "product.price" ],
        "rangeComparisonType" : "FLOAT",
        "jcr:primaryType" : "endeca:dimension"
      },
```

```
      "product.max_price" : {
        "derivedPropertySource" : "product.price",
        "derivedPropertyFunction" : "MAX",
        "jcr:primaryType" : "endeca:derivedProperty"
      },
      "product.long_desc" : {
        "isRecordSearchEnabled" : true,
        "propertyDataType" : "ALPHA",
        "jcr:primaryType" : "endeca:property"
      },
      "product.short_desc" : {
        "isRecordSearchEnabled" : true,
        "propertyDataType" : "ALPHA",
        "jcr:primaryType" : "endeca:property"
      },
    ...
    },
    "precedenceRules" : {
    },
    "searchIndexConfig" : {
      "spellingDictMinNumWordOccurrences" : 4,
      "spellingDictMaxWordLength" : 16,
      "isWildcardEnabledInDimensionSearch" : true,
      "spellingDictMinWordLength" : 3
    }
  }
}
```

Setting the index configuration for an application

The `set-config` task sets the index configuration for a specified owner. You provide the index configuration in a JSON file.

Running this task overwrites any previous index configuration for the owner. Oracle recommends that developers who modify the index configuration, use the default `system` owner. This usage separates index configuration that comes from the `system` owner from configuration that comes from import operations which are owner by a user-specified owner. If the JSON configuration file contains index configuration from multiple owners, you must specify the `-o` option with a value of `all`.

If desired, you can also update an Endeca property to become a dimension by modifying the `jcr:primaryType` from `endeca:property` to `endeca:dimension`. However, you cannot modify it from dimension to property.

The syntax for this task is:

```
index_config_cmd set-config [-o OwnerName] -f FileName
```

Where:

- `-o` (or `--owner`) specifies an import owner for a task. If you specify the `-o` option, the task applies only to the owner specified. The option can have an argument of `all`, `system`, or a user-specified import owner. The `all` owner includes both the `system` owner and all import owners. If you omit this option, the task applies to the `system` owner. Optional.
- `-f` (or `--file`) specifies a path to a JSON file that contains the index configuration. Required.

To set the index configuration for an application:

1. Start a command prompt and navigate to `<app dir>\control` (for Windows) or `<app dir>/control` (for UNIX).

2. Type `index_config_cmd` and specify the `set-config` task and the `-f` option with a path to the JSON file.

Note: This task name is case sensitive.

Examples of setting the index configuration for an application

This example sets index configuration from three owners.

```
C:\Endeca\apps\Discover\control>index_config_cmd.bat set-config -f C:\temp\index¬
ConfigAllOwners.json -o all
[07.24.12 15:53:58] INFO: Using site Discover at URL http://JSMITH-WIN7:8006/
ifcr with username admin
You are attempting to write schema configuration that will be overwritten in the

 event of a fresh import.
Are you sure you want to continue? (y/n)
y
```

This example sets index configuration from an owner named `ATG`.

```
C:\Endeca\apps\Discover\control>index_config_cmd.bat set-config -f C:\temp\index¬
ConfigATGOwner.json -o ATG
[07.24.12 16:23:06] INFO: Using site Discover at URL http://JSMITH-WIN7:8006/
ifcr with username admin
You are attempting to write schema configuration that will be overwritten in the

 event of a fresh import.
Are you sure you want to continue? (y/n)
y
```

Deleting the index configuration associated with an owner

The `delete-owner` task removes index configuration from an application that is associated with an owner that you specify.

The syntax for this task is:

```
index_config_cmd delete-owner -o OwnerName
```

Where:

- `-o` (or `--owner`) specifies an import owner for a task. If you specify the `-o` option, the task applies only to a user-specified owner. You cannot delete the `system` or `all` owners. Required.

To delete the index configuration associated with an owner:

1. Start a command prompt and navigate to `<app dir>\control` (for Windows) or `<app dir>/control` (for UNIX).
2. Type `index_config_cmd` and specify the `delete-owner` task with an argument for the owner's index configuration that you want to remove.

Note: This task name is case sensitive.

Example of deleting the index configuration associated with an owner

This example deletes the index configuration for the ATG owner.

```
C:\Endeca\apps\Discover\control>index_config_cmd.bat delete-owner -o ATG
[07.24.12 17:14:50] INFO: Using site Discover at URL http://JSMITH-WIN7:8006/
ifcr with username admin
```

An example of changing multi-select on a dimension

This topic provides a simple example of using the Index Configuration Command-line Utility to update index configuration. In this example, suppose an import owner named `ATG` has added index configuration to an Endeca application. You want to update the index configuration by adding `multiSelectType` to the `product.category` dimension.

The steps to accomplish this are as follows:

1. Retrieve the index configuration by running:

```
C:\Endeca\apps\<app dir>\control>index_config_cmd.bat get-config -o ATG -f
C:\temp\indexConfig.json
```

2. Open the resulting JSON file and locate the `product.category` attribute:

```
{
  "indexConfig" : {
      "attributes" : {
        "product.category" : {
          "sourcePropertyNames" : [ "product.category_id" ],
          "isRecordSearchEnabled" : true,
          "jcr:primaryType" : "endeca:dimension"
        },
      },
....
```

3. Add `multiSelectType` and set the value to `OR`, and also add the `mergeAction` with a value of `UPDATE`. You can delete other properties of the attribute because they are not changing as part of the update:

```
{
  "indexConfig" : {
      "attributes" : {
        "product.category" : {
          "mergeAction" : "UPDATE",
          "multiSelectType" : "OR",
        },
      },
....
```

4. Set the revised index configuration by running:

```
C:\Endeca\apps\<app dir>\control>index_config_cmd.bat set-config -o ATG
-f C:\temp\indexConfig.json
```

5. If desired, examine the merged configuration by running:

```
C:\Endeca\apps\<app dir>\control>index_config_cmd.bat get-merged-config
-f C:\temp\indexConfig.json
```

You see the following:

```
{
  "indexConfig" : {
    "ATG" : {
```

```
      "attributes" : {
        "product.category" : {
          "sourcePropertyNames" : [ "product.category_id" ],
          "isRecordSearchEnabled" : true,
          "multiSelectType" : "OR",
          "jcr:primaryType" : "endeca:dimension"
        },
      },
....
```

An example of changing a product.brand.name property to a dimension

This topic provides a simple example of using the Index Configuration Command-line Utility to update index configuration. In this example, suppose an import owner named `ATG` has added index configuration to an Endeca application. You want to update the index configuration by changing the `product.brand.name` attribute from an Endeca property to an Endeca dimension.

The steps to accomplish this are as follows:

1. Retrieve the index configuration by running:

```
C:\Endeca\apps\<app dir>\control>index_config_cmd.bat get-config -o ATG -f
C:\temp\indexConfig.json
```

2. Open the resulting JSON file and locate the `product.brand.name` attribute:

```
{
  "indexConfig" : {
      "attributes" : {
        "product.brand.name" : {
          "isRecordSearchEnabled" : true,
          "jcr:primaryType" : "endeca:property"
        },
      },
....
```

3. Change `jcr:primaryType` from `endeca:property` to `endeca:dimension`, add the `mergeAction` with a value of `UPDATE`, and also add `isAutogen` with a value of `true`:

```
{
  "indexConfig" : {
      "attributes" : {
        "product.brand.name" : {
          "mergeAction" : "UPDATE",
          "isAutogen" : true,
          "jcr:primaryType" : "endeca:dimension",
          "multiSelectType" : "OR"
        },
      },
....
```

4. Set the revised index configuration by running:

```
C:\Endeca\apps\<app dir>\control>index_config_cmd.bat set-config -o ATG
-f C:\temp\indexConfig.json
```

5. If desired, examine the merged configuration by running:

```
C:\Endeca\apps\<app dir>\control>index_config_cmd.bat get-merged-config
-f C:\temp\indexConfig.json
```

You see the following:

```
{
  "indexConfig" : {
    "attributes" : {
      "product.brand.name" : {
        "isHierarchicalDimensionSearchEnabled" : true,
        "isRecordSearchEnabled" : true,
        "isAutogen" : true,
        "jcr:primaryType" : "endeca:dimension",
        "multiSelectType" : "OR"
      },
....
```

An example of setting dimension display order

This topic provides an example of how the `displayOrder` property sets the display order of dimensions in the `discover-data-cas` application.

The following JSON snippet shows the `displayOrder` property for the Category dimension, Price Range dimension, and Brand Name dimension, where `displayOrder` is set to 0, 1, and 2, respectively.

```
      ...
      "product.category" : {
        "displayOrder" : 0,
        "sourcePropertyNames" : [ "product.category_id" ],
        "isRecordSearchEnabled" : true,
        "jcr:primaryType" : "endeca:dimension"
      },
      "product.price_range" : {
        "sourcePropertyNames" : [ "product.price" ],
        "displayOrder" : 1,
        "rangeComparisonType" : "FLOAT",
        "jcr:primaryType" : "endeca:dimension"
      },
      ....
      "product.brand.name" : {
        "isHierarchicalDimensionSearchEnabled" : true,
        "displayOrder" : 2,
        "isRecordSearchEnabled" : true,
        "isAutogen" : true,
        "jcr:primaryType" : "endeca:dimension",
        "multiSelectType" : "OR"
      },
      ....
```

When the dimensions are rendered in the Discover Electronics reference application, they render in the order specified by the property value. Category displays first, Price Range second, and Brand Name third:

Configuring BeanShell scripts

The following list describes a number of customization approaches that you can implement to extend the existing functionality or add new functionality to the template.

- For example, if a deployment uses JDBC to read data into the Forge pipeline instead of using extracted data files, the following changes would be implemented in the BaselineUpdate script:

  1. Remove the line that retrieves data and configuration for Forge: `Forge.getData();`
  2. Insert a new copy command to retrieve configuration for Forge to process:

```
...
// get Workbench config, merge with Dev Studio config
ConfigManager.downloadWsConfig();
ConfigManager.fetchMergedConfig();

// fetch extracted data files, run ITL
srcDir = PathUtils.getAbsolutePath(Forge.getWorkingDir(),
  Forge.getConfigDir()) + "/\\*";
destDir = PathUtils.getAbsolutePath(Forge.getWorkingDir(),
  Forge.getInputDir());
```

```
dimensionCopy = new CopyUtility(Forge.getAppName(),
  Forge.getEacHost(), Forge.getEacPort(), Forge.isSslEnabled());
dimensionCopy.init("copy_dimensions", Forge.getHostId(),
  Forge.getHostId(), srcDir, destDir, true);
dimensionCopy.run();

Forge.getData();
Forge.run();
Dgidx.run();
...
```

Note that this amended BeanShell script imports two classes from the classpath, references variables that point to elements in the `AppConfig.xml` document (e.g. Forge, Dgidx) and defines new variables without specifying their type (e.g. srcDir, destDir). Details about BeanShell scripting can be found in Appendix A of this guide.

- Write new BeanShell scripts - Some use cases may call for greater flexibility than can easily be achieved by modifying existing BeanShell scripts. In these cases, writing new BeanShell scripts may accomplish the desired goal. For example, the following BeanShell script extends the previous example by pulling the new functionality into a separate script:

```
<script id="CopyConfig">
  <bean-shell-script>
    <![CDATA[

  // fetch extracted data files, run ITL
  srcDir = PathUtils.getAbsolutePath(Forge.getWorkingDir(),
    Forge.getConfigDir()) + "/\\*";
  destDir = PathUtils.getAbsolutePath(Forge.getWorkingDir(),
    Forge.getInputDir());

  dimensionCopy = new CopyUtility(Forge.getAppName(),
    Forge.getEacHost(), Forge.getEacPort(), Forge.isSslEnabled());
  dimensionCopy.init("copy_dimensions", Forge.getHostId(),
    Forge.getHostId(), srcDir, destDir, true);
  dimensionCopy.run();

    ]]>
  </bean-shell-script>
</script>
```

Once the new script is defined, the BaselineUpdate script simplifies to the following:

```
...
// get Workbench config, merge with Dev Studio config
ConfigManager.downloadWsConfig();
ConfigManager.fetchMergedConfig();

// fetch extracted data files, run ITL
CopyConfig.run();
Forge.getData();
Forge.run();
Dgidx.run();
...
```

- Define utilities in `AppConfig.xml` - A common use case for customization is to add or adjust the functionality of utility invocation. Our previous example demonstrates the need to invoke a new copy utility when the Forge implementation changes. Other common use cases involve invoking a data pre-processing script from the shell and archiving a directory. In order to enable this, the Deployment Template allows utilities

to be configured in the AppConfig.xml document. To configure the copy defined above in the document, use the copy element:

```
<copy id="CopyConfig" src-host-id="ITLHost" dest-host-id="ITLHost"
  recursive="true">
  <src>./data/complete_index_config/*</src>
  <dest>./data/processing</dest>
</copy>
```

Once configured, this copy utility is invoked using the same command that was previously added to the BaselineUpdate to invoke the custom BeanShell script: CopyConfig.run();

- Extend the Java EAC Development Toolkit - In rare cases, you may need to implement complex custom functionality that would be unwieldy and difficult to maintain if implemented in the AppConfig.xml document. In these cases, you can extend objects in the toolkit to create new Java objects that implement the desired custom functionality. Staying with the previous example, the developer might implement a custom Forge object to change the behavior of the getData() method to simply copy configuration without looking for extracted data files.

```
package com.Endeca.soleng.eac.toolkit.component;

import java.util.logging.Logger;
import com.Endeca.soleng.eac.toolkit.exception.*;

public class MyForgeComponent extends ForgeComponent
{
  private static Logger log =
    Logger.getLogger(MyForgeComponent.class.getName());

  protected void getData() throws AppConfigurationException,
      EacCommunicationException, EacComponentControlException,
      InterruptedException
  {

    // get dimensions for processing
    getConfig();

  }
}
```

Obviously, this trivial customization is too simple to warrant the development of a new class. However, this approach can be used to override the functionality of most methods in the toolkit or to implement new methods.

In order to use the new functionality, the developer will compile the new class and ensure that it is included on the classpath when invoking scripts. The simplest way to do this is to deploy the compiled .class file to the <app dir>/config/script directory. Once on the classpath, the new component can be loaded in place of the default Forge component by making the following change to the Forge configuration in AppConfig.xml:

```
<forge class="com.Endeca.soleng.eac.toolkit.component.MyForgeComponent"
  id="Forge" host-id="ITLHost">
...
</forge>
```

Some types of customization will require more complex configuration. Refer to Appendix A ("EAC Development Toolkit") for information about configuring custom Java classes using the Spring Framework namespace in the AppConfig.xml document.

Configuration overrides

The Deployment Template allows the use of one or more configuration override files.

These files can be used to override or substitute values into the configuration documents. For example, developers may want to separate the specification of environment-specific configuration (e.g. hostnames, ports, etc.) from the application configuration and scripts. This may be useful for making configuration documents portable across environments and for dividing ownership of configuration elements between system administrators and application developers.

Override files are specified by using the `--config-override` flag to the EAC development toolkit's controller. For example, the `runcommand` script in the template includes an environment.properties file by default, though this file only contains examples of overrides and does not specify any active overrides.

Two types of properties can be specified in an override file:

1. `[object].[field] = [value]` - This style of override specifies the name of an object and field and sets the value for that field, overriding any value specified for that field in the XML configuration document or documents. For example:

   ```
   Dgraph1.port = 16000
   Dgraph1.properties['restartGroup'] = B
   ITLHost.hostName = itl.mycompany.com
   ```

2. `[token] = [value]` - This style of override specifies the name of a token defined in the XML config file and substitutes the specified value for that token. For example, if the `AppConfig.xml` defines the following host:

   ```
   <host id="ITLHost" hostName="${itl.host}" port="${itl.port}" />
   ```

   The override can specify the values to substitute for these tokens:

   ```
   itl.host = it.mycompany.com
   itl.port = 8888
   ```

It is important to note that both styles of substitution are attempted for every value defined in the override file. When a token fails to match, a low-severity warning is logged and ignored. This is required because most tokens will only match one of the two styles of substitution. It may be important to avoid using token names that coincide with object names. For example, defining the token `${Forge.tempDir}` will cause the corresponding value to substitute for both the token as well as the `tempDir` field of the Forge component.

Modifying Deployment Template files to support custom applications

This section provides information about deploying custom applications.

Custom application descriptors

The Deployment Template deploys new applications based on application descriptor XML documents. The documents describe the directory structure associated with an application as well as the files to distribute during the deployment process.

By default, the Deployment Template ships with application descriptor files named `base_descriptor.xml` located in `<installation path>\ToolsAndFrameworks\<version>\deployment_template\app-templates`.

This document describes the directory structure of the deployment as well as the copying that is done during the deployment to distribute files into the new directories. Additionally, this document describes whether files are associated with a Windows or UNIX deployment, and whether copied files should be updated to replace tokens in the format @@TOKEN_NAME@@ with text strings specified to the installer.

The following tokens are handled by the base descriptor:

- @@WORKBENCH_PORT@@ - Oracle Commerce Workbench port.
- @@DGRAPH_1_PORT @@ - Live Dgraph port.
- @@AUTHORING_DGRAPH_PORT @@ - Authoring Dgraph port.
- @@LOGSERVER_PORT@@ - Log Server port.
- @@JPSCONFIG_LOCATION@@ - Path to Oracle Wallet configuration file.

The following tokens are handled by the Deployment Template:

- @@EAC_PORT@@ - EAC Central Server port.
- @@HOST@@ - Hostname of the server on which the deploy script is invoked.
- @@PROJECT_DIR@@ - Absolute path of the target deployment directory.
- @@PROJECT_NAME@@ - Name of the application to deploy.
- @@ENDECA_ROOT@@ - Absolute path of the ENDECA_ROOT environment variable.
- @@SCRIPT_SUFFIX@@ - ".bat" for Windows, ".sh" for Linux installs.

In addition to these tokens, you can specify custom tokens to substitute in the files. Tokens are specified in the application descriptor file, including the name of the token to substitute as well as the question with which to prompt the user or the installer configuration option to parse to retrieve the value to substitute for the token. The default application descriptors use this functionality to request the port number for Dgraphs, Log Servers and Forge servers.

If a project deviates from the Deployment Template directory structure, it may find it useful to create a custom application descriptor document, so that the default Deployment Template can continue to be used for application deployment.

Custom deployment descriptors may also be used to define add-on modules on top of a base install. For example, sample applications (such as the Sample Term Discovery and Clustering application) are shipped with a custom deployment descriptor file, which describes the additional files and directories to install on top of a base Dgraph deployment. Modules may be installed using the deploy batch or shell script, specifying the `--app` argument with the location of the application descriptor document. For example:

```
deploy.bat --app \
C:\Endeca\Solutions\sampleTermDiscovery-[VERSION]\data\deploy.xml
```

The installer prompts you to specify whether it should install the module as a standalone installation or if it should be installed on top of the base Dgraph deployment. Multiple add-on modules may be specified to the installer script, though only one of them may be a base install (that is, all but one of them should specify an attribute of update= "true").

The following excerpt from the Dgraph deployment application descriptor identifies the document's elements and attributes:

```
<!--
  Deployment Template installer configuration file. This file defines the direc¬
tory structure to create and the copies to perform to distribute files into the
new directory structure.

  The update attribute of the root install element indicates whether this is a
core installation or an add-on module. When set to false or unspecified, the in¬
stallation requires the removal of an existing target install directory (if
present). When update is set to true, the installer preserves any existing direc¬
tories, adding directories as required and distributing files based on the speci¬
fied copy pattern.
```

```
-->
<app-descriptor update="false" id="Dgraph">

  <custom-tokens>
    <!-- Template custom token:
      <token name="MYTOKEN">
        <prompt-question>What is the value to substitute for token MYTO¬
KEN?</prompt-question>
        <install-config-option>myToken</install-config-option>
        <default-value>My Value</default-value>
      </token>

      This will instruct the installer to look for the "myToken" option
      in a specified install config file (if one is specified) or to
      prompt the user with the specified question to submit a value. If a
      value is entered/retrieved, the installer will substitute instances
      of @@MYTOKEN@@ with the value.
    -->
  </custom-tokens>

  <dir-structure>
    <!-- Template directory:
      <dir platform="unix" primary="true"></dir>

      primary           builds directory only on primary server installs

      platform          builds directory only on specified platform.
                          Valid values: "win" and "unix"
    -->
  </dir-structure>

  <!—
  Copy source directory is specified relative to this file's directory
  -->
  <copy-pattern src-root="../data ">
    <!-- Template copy pattern:
      <copy clear-dest-dir="true" recursive="true"
            preserve-subdirs="true" filter-files="true"
            primary="true" platform="win" Endeca-version="480">
        <src-dir></src-dir>
        <src-file></src-file>
        <dest-dir></dest-dir>
      </copy>

      src-dir           source directory, relative to root of deployment
                          template package.

      src-file          source filename or pattern (using '*' wildcard
                          character) to copy from source dir

      dest-dir          destination directory, relative to root of target
                          deployment directory.

      clear-dest-dir    removes all files in target dir before copying

      recursive         copies files matching pattern in subdirectories
                          of the specified source dir

      preserve-subdirs  copies files, preserving dir structure. Only
                          applicable to recursive copies
```

```
      filter-files      filters file contents and file names by replacing
                          tokens (format @@TOKEN@@) with specified
                          strings.

      mode              applies the specified permissions to the files
                          after the copy. Mode string should be 3 octal
                          digits with an optional leading zero to
                          indicate octal, e.g. 755, 0644. Not relevant
                          for Windows deployments.

      platform          applies copy to specified platform. Valid
                          values: "win" "unix"

      Endeca-version    applies copy to specified Guided Search version Valid
                          values: "460" "470" "480" "500"
    -->
  </copy-pattern>
</app-descriptor>
```

Configuring an automated/file-based deployment for a custom application

The configuration file discussed in previous sections may be used to specify the location of custom application descriptor documents in place of the `--app` command line argument to the installer.

The following example shows how to install the Sample Term Discovery and Clustering application on top of the base Dgraph deployment.

```
<install app-name="MyApp" >
  <deployment-path>C:\Endeca</deployment-path>
  <base-module type="dgraph" />
  <additional-module type="custom">
    C:\Endeca\Solutions\sampleTermDiscovery-[VERSION]\data\deploy.xml
  </additional-module>
  <options>
    <option name="eac-port">8888</option>
    <option name="dgraph1Port">15000</option>
    <option name="logserverPort">15010</option>
    <option name="jps-config-location">ToolsAndFrameworks/<version>/serv¬
er/workspace/credential_store/jps-config.xml</option>
  </options>
</install>
```

Part 3

Monitoring and Administering an Application

Chapter 9

Monitoring Application Components

This section covers monitoring your application, including configuring and reviewing log files, and checking component status through the EAC Admin Console in Workbench.

Determining the state of the EAC Central Server and Agents

You can use the following service URLs to determine whether the EAC Central Server or EAC Agent is running.

To determine the state of the EAC:

- For the EAC Central Server, go to: `http://machine_name:8888/eac/ProvisioningService?wsdl`
- For the EAC Agent, go to: `http://machine_name:8888/eac-agent/IDelegateServer?wsdl`

Starting and stopping the EAC

The EAC typically starts automatically within the Endeca HTTP Service, though you can also start and stop it independently.

To start or stop the EAC:

1. To start the EAC:

Option	Description
Windows	In the **Services** menu, start or restart the Endeca HTTP Service.
UNIX	Run the `$ENDECA_ROOT/tools/server/bin/startup.sh` script.

   This also starts any other components that run on the same port.
2. To stop the EAC:

Option	Description
Windows	In the **Services** menu, stop the Endeca HTTP Service.
UNIX	Run the `$ENDECA_ROOT/tools/server/bin/shutdown.sh` script.

   This also stops any other components running on the same port.

On UNIX, you may wish to start the EAC with the `init` process. See the following topic for details.

Starting the EAC from inittab

In a UNIX production environment, the Endeca Application Controller can start with `init` by entering the corresponding shell script in `inittab`.

In a UNIX development environment, the Endeca HTTP Service can be started from the command line. In a UNIX production environment, however, Oracle recommends that it be started with the `init` process. If the service crashes or terminates, `init` automatically restarts it.

The UNIX version of Platform Services contains a variant of `startup.sh`, named `$ENDECA_ROOT/tools/server/bin/endeca_run.sh`, that calls `run` instead of `start` and redirects `stdout` and `stderr` to `$ENDECA_CONF/logs/catalina.out`.

You can write a script to call `endeca_run.sh` as a non-root user. This lets you set a `$USER` environment variable that is inherited by EAC scripts.

Example

This sample script below, `start_endeca_http_service.sh`, sets `$ENDECA_USER` to the “endeca” user, sets `$INSTALLER_SH` to the path of the environment variables script and sources it, and then changes to the “endeca” user and starts the EAC:

```
#!/bin/sh
ENDECA_USER=endeca
INSTALLER_SH=/usr/local/endeca/PlatformServices/workspace/setup/installer_sh.ini
# We want to use installer_sh.ini variables in this script,
# so we source it here.
source $INSTALLER_SH
# change to user endeca
su $ENDECA_USER -c "/bin/sh -c \"source $INSTALLER_SH; \
   cd $ENDECA_CONF/work; exec env USER=$ENDECA_USER \
   $ENDECA_ROOT/tools/server/bin/endeca_run.sh\""
```

Note: On Solaris platforms, replace "source" with "." since source is not a command in the Bourne shell.

You can add this script to `inittab` with an entry similar to the following:

```
ec:2345:respawn:/usr/local/endeca/PlatformServices/workspace/setup/start_ende¬
ca_http_service.sh
```

Logs for the EAC Central Server

Aside from log files associated with specific EAC components, utilities, and scripts, the EAC Central Server and its services generate log files in their workspace directories.

The EAC logs are located in `%ENDECA_CONF%\logs` (on Windows), or `$ENDECA_CONF/logs` (on UNIX).

Specifically, the `logs` directory contains a number of files generated by the Endeca HTTP service, and the applications running inside it, such as the EAC Central Server and EAC Agent.

The EAC logs have a default size limit of 1Gb. The log is named `main.rotation number.log` and is part of a two-log rotation that rolls automatically when the maximum size is reached. When the second log file reaches the maximum size, the first is overwritten. That is, when `main.0.log` reaches the 1 Gb size limit, the system starts to write to `main.1.log`. Once `main.1.log` reaches the 1 Gb size limit, `main.0.log` is overwritten.

The following log files are typically useful and relevant in EAC development and debugging:

Type of EAC logs	Description
Main log, such as `main.0.log`	Most EAC logging goes into this log file. For example, exceptions thrown when invoking a shell or component are logged in it. For example, if you attempt to launch a utility on a non-existent host, an exception similar to the following is logged in this file: "`The host "my_host" in application "my_app" does not exist`"
Process log, such as `process.0.log`	Logs generated by the EAC process control module go into this file. This log contains messages associated with process control and recovery. These messages include information about starting and stopping scripts, components and utilities, recovering failed processes and rebinding to active processes.
Invocation log, such as `invoke.0.log`	This file contains logs associated with the EAC Web service invocations. For example, this file records the exact XML content of Web service requests and responses.
Tomcat/Catalina logs, such as: • `catalina.out` • `catalina.`*`[date]`*`.log` • `tomcat_`*`[stdout|stderr]`*`.log`	These logs are useful when errors occur while loading the EAC. For example, if the context configuration for EAC specifies the wrong path for the EAC WAR file, an error occurs when starting the Endeca HTTP service, and is logged in these log files. Alternatively, a clean startup of the Endeca HTTP service results in no exceptions, and a successful output of a message: "`Server startup in [n] ms`"

Optimizing EAC memory usage

This topic outlines recommendations for optimizing EAC memory usage. For example, if a server is running more than one MDEX Engine and more than one EAC agent, you may encounter performance problems due to multiple EAC agent processes having a large memory footprint (amount of RAM consumed by the EAC agent processes).

To optimize EAC memory usage, use the following recommendations:

- Use third-party utilities, such as top, to measure the virtual memory usage and the Resident Set Size (RSS) of the EAC agent processes.
- Note that although the virtual memory usage may be high, as long as the working set size of the processes fits into RAM, this should not be a cause for concern.

  For detailed information on memory usage in the MDEX Engine and the information on how RSS, working set size, and virtual memory used by the process relate to each other, see the *Oracle Commerce MDEX Engine Performance Guide*.

- Free up memory for the MDEX Engine and EAC agent processes by removing non-critical non-Endeca processes running on the server. Also, consider increasing the amount of swap space.
- If your current implementation is memory constrained and processes are running out of memory, consider reconfiguring the topology of your implementation. For example, if previously you had 10 MDEX Engine

servers each with 8 cores hosting two MDEX Engines, consider reconfiguring your topology to have one MDEX Engine running with 8 threads per server instead of two. Such a configuration will be much more memory efficient.

Note: Although one 8-threaded Dgraph is not expected to yield as much throughput as two 4-threaded Dgraphs if there were enough RAM for both, in cases where there are physical memory constraints on servers running the MDEX Engine, one 8-threaded Dgraph may yield more throughput because restarting of the MDEX Engine due to it running into memory issues will be avoided.

Checking the status of application components

You can check the status of a provisioned application's components (such as Forge, the Dgidx, or Dgraphs) using either Workbench or the scripts in the Deployment Template.

To check the status of a provisioned application's components, use one of the following:

- **Oracle Endeca Workbench** — Use the **EAC Administration** console to monitor a particular application's component status. The status of each component can be set to **auto-refresh**.
- **The `eaccmd` utility**. Run the following command at a command line (UNIX and Windows):

  ```
  eaccmd status --app <application name> --comp <component id>
  ```

  Note: You can also substitute the `--comp <component id>` argument with the `--script <script_id>` argument to check the status of a specific EAC script.

- **The Deployment Template** — From the `control` directory of your deployed application, run the `runcommand` script with the `--print-status` flag:

  ```
  runcommand.bat --print-status
  ```

  This commands prints the status for each component in the current application.

Using the Assembler administrative servlet

The Assembler administrative servlet is available at `http://<application host>:<application port>/<application>/admin`. Accessing the servlet without entering an operation in the URL returns a list of available operations.

The servlet supports the following operations:

Admin Operation	Description
`/admin?op=update`	Updates the list of running Assemblers from the specified location on disk.
`/admin?op=usage`	Returns Assembler usage information as XML.
`/admin?op=stats`	Returns the performance statistics page.
`/admin?op=statsreset`	Resets Assembler statistics. It can be combined with `stats` to retrieve statistics and then reset them: `/admin?op=stats&op=statsreset`

Example

To access the performance statistics page for the Discover Electronics authoring application with default settings, enter the following in your browser's address bar:

```
http://localhost:8006/discover-authoring/admin?op=stats
```

The Assembler performance statistics page

The `/admin?op=stats` operation returns the performance statistics page, which lists the completion time for Assembler events. Most fields are documented within the page itself.

You can reset the statistics by issuing the `/admin?op=statsreset` operation or by clicking the **Reset statistics** link in the **Last Reset Time** row.

In addition to overall event statistics, the page includes two other tables that you can expand or collapse using the **Show / Hide** link.

- **Worst root events** — The 20 events that took the most time, sorted by total time in descending order. You can expand or collapse a row by using the **Show / Hide** link in the **Details** column.

  Note: Only root events created by a call to `PerfUtil.start()` show an entry in the **URL / description** column.

- **Per event statistics sorted by total time - median** — All events are sorted in descending order by median total time. For the **Median**, **90th**, and **99th** columns, all percentile values between 0.001 and 60,000 milliseconds (ms) are within 2% of their actual values.

Using the EAC Administration Console in Workbench

The EAC Administration Console page provides a way for administrators to establish and modify system provisioning, start and stop system components, and run EAC scripts.

The Administration Console of Oracle Endeca Workbench is divided into three sections:

- **Hosts** – shows a view of your application organized by the hosts you provision. This view indicates the host name, host alias, port and configuration options. You can view component status on a host, and start or stop components.
- **Components** – shows a view of your application organized by the Guided Search components provisioned for an application.

  Note: Oracle does not recommend creating new components from this tab, as any changes are not saved in the configuration files on disk. This results in clearing any created components if you later run scripts to update EAC configuration from those files.

- **Scripts** – shows the EAC scripts available to an application and allows you to add, remove, run, and monitor EAC scripts. You can stop and start system operations run by EAC scripts, such as baseline updates.

Monitoring Host and Component status

Each host and component that you provision in an Endeca application displays its system status on the EAC Admin Console page of Oracle Workbench.

Oracle Commerce Workbench displays a summary of the component's status in the collapsed view of the **Hosts** tab and **Components** tab. You can access details about each component (except the Log Server, which does not log its own activities) via the status link next to each component. Clicking the status link displays start time, duration (how long the component has been running), and the last time Oracle Commerce Workbench checked the component's status.

You can manually refresh the status display by clicking the **Refresh Status** button. You can also set the page to be refreshed automatically (at a pre-set interval) by checking the **Auto Refresh** checkbox. This option is useful when a baseline update is in progress and the system state changes frequently. By default, this option is turned off because the overall system state changes infrequently.

To view the latest log for an Endeca component (except for the Log Server which does not log its own actions), navigate to the **Components** tab and click the **Edit** link to expand the component properties. Browse to the log file directory indicated in the **Log File** field.

Starting and stopping the MDEX Engine

You can view and change the status of an MDEX Engine from the Components tab of the EAC Administration Console.

Depending on whether the MDEX Engine is Stopped or Running, either the **Start** or **Stop** link is available. When you start an MDEX Engine, it starts with any options that you specified in **Arguments** field of component configuration.

Running a baseline update

A baseline update rebuilds the back end of an Assembler application, including running Forge on the source data, running Dgidx to produce the Endeca records and indices, and starting one or more MDEX Engines with the new indices.

Additionally, running the baseline update script publishes content from the Endeca Configuration Repository to the MDEX Engine.

Starting and stopping the Log Server

You can start and stop the Log Server from the **Components** tab of the EAC Administration Console.

Depending on whether the Log Server is Stopped or Running, either the **Start** or **Stop** link is available.

Rolling Log Server logs

You cannot roll the logs created by the Log Server from the EAC Admin Console. However, you can roll the logs with a URL command.

To roll the logs use the following URL command:

```
http://logserverhost:logserverport/roll
```

For example, this command:

```
http://web002:8002/roll
```

rolls the Log Server that is running on port 8002 on the host named web002.

Archiving Dgraph log files

Dgraph log files are archived automatically upon running a baseline update script with the Deployment Template. If you prefer to archive Dgraph files on a more granular basis, you can create a custom Deployment Template script that stops the MDEX Engine process, archives the Dgraph log files, and restarts the Dgraph.

The `applyIndex()` method of the baseline update script stops the Dgraph, archives the log files, and restarts the Dgraph. This method is located in the `DistributeIndexAndApply` step of the default baseline update script in the Deployment Template. You can use this method to create a customized script.

To stop the Dgraph, archive its logs, and restart the Dgraph:

1. Copy the `applyIndex()` method into a new script within your Deployment Template project, and modify it as needed, as shown in the following example:

```
<!--########################################
 # CUSTOM: Restart all dgraphs, archiving log files
 #
 -->
<script id="DgraphRestartWithArchive">
<log-dir>./logs</log-dir>
 <bean-shell-script>
   <![CDATA[
    for ( DgraphComponent dgraph : DgraphCluster.getDgraphs() )
    {
       if ( dgraph.isActive() )
       {
         dgraph.stop();
       }
    dgraph.archiveLogDir();
    dgraph.start();
    }
  ]]>
  </bean-shell-script>
</script>
```

2. To archive Dgraph logs, go to the application `/control` directory, and run this script: `runcommand DgraphRestartWithArchive`

About Workbench logs

The Oracle Commerce Workbench logs are located in `%ENDECA_TOOLS_CONF%\logs` (on Windows) or `$ENDECA_TOOLS_CONF/logs` (on UNIX).

The following logs can be found in this directory:

File name	Description
`webstudio.log`	System log for Oracle Commerce Workbench, including activity such as user logins, updates to instance configuration, and Oracle Endeca Workbench errors.
`webstudio_audit.log`	Audit log for activity such as dynamic business rule and search configuration changes. Logging for rules

File name	Description
	includes the name of the rule being modified, when it was modified, who modified it (based on Endeca Workbench user name), and any note associated with the change.

By default, the Oracle Commerce Workbench system log and audit log have a maximum size of 1MB. Each of the logs is part of a four-log rotation.

Chapter 10

Configuring and Viewing Assembler Usage Reports

This secton describes the steps necessary to set up usage reporting in your Assembler application. Note that this is required to maintain licensing compliance.

About product licensing and usage reports

Oracle Commerce Guided Search licenses are issued for a certain amount of peak page views per day. The Tools and Frameworks package logs the necessary metrics, which you can then use to analyze overall application usage and to determine whether your current license is sufficient to cover expected daily traffic. To maintain licensing compliance, you must configure usage logging and reporting when deploying an application to a live production environment.

Tools and Frameworks logs Assembler and MDEX Engine usage by tracking `assemble()` calls and Dgraph Record Search, Dimension Search, and Navigation queries, respectively. If you are licensing the Guided Search package, only Dgraph statistics are logged.

Important: By default, usage reports are stored in the `<app name>\reports\usage` directory of a deployed EAC application on disk. You should create a script to back up this information, and retain backups for at least one calendar year.

Configuring usage logging

When you deploy an application to a live production environment, you must configure usage logging by updating the server cluster XML files in the `config` directory of the deployed EAC application on disk.

The steps below assume you have promoted your application from an authoring or staging server to your live environment.

At a high level, configuring usage logging consists of specifying the following in the logging configuration files:

- Specify production Dgraph clusters
- For Assembler applications, specify production clusters, application servers, and the applications they contain
- Specify output directories for the logs

Optionally, you can configure logging on an authoring or staging environment for testing and to ensure that everything works as expected.

To configure usage logging:

1. Navigate to the `config\script` directory of your deployed EAC application on disk.
   For example, `Endeca\apps\Discover\config\script`.
2. If you have licensed Oracle Commerce Guided Search with Experience Manager, modify the sample configuration file that defines your application server clusters:
   a) Open the `LiveAppServerCluster.xml` file.
   b) For each server cluster, create an `<app-server-cluster>` element with an `id` attribute that corresponds to the cluster name.
      For example:
      ```
      <app-server-cluster id="LiveAppServerCluster">
      </app-server-cluster>
      ```
   c) For each server within the cluster, create an `<app-server>` element with the following attributes:
      - `id` — The name of the server.
      - `hostName` — The DNS name or IP address of the server hosting the Assembler.
      - `port` — The port on which the Assembler Web application is running.

      For example:
      ```
      <app-server id="LiveDiscover" hostName="assemblerHost.example.com"
      port="8006">
      </app-server>
      ```
   d) For each application running on a given Assembler, create a `<web-app>` element with the following attributes:
      - `id` — The name of the Assembler application.
      - `contextPath` — The path to the application relative to the Assembler server.
      - `sslEnabled` — Optionally, whether the application is SSL-enabled.

      For example:
      ```
      <web-app id="DiscoverWebApp" contextPath="/discover" sslEnabled="true" />
      <web-app id="DiscoverAsService" contextPath="/discoverAsService" />
      ```
   e) Add the `<web-app>` elements to their respective `<app-server>`s as referenced elements.
      For example:
      ```
      <app-server id="LiveDiscover" hostName="assemblerHost" port="8006">
          <web-app ref="DiscoverWebApp" />
          <web-app ref="DiscoverAsService" />
      </app-server>
      ```
   f) Add the `<app-server>` elements to their respective `<app-server-cluster>`s as referenced elements.
      For example:
      ```
      <app-server-cluster id="LiveAppServerCluster">
          <app-server ref="LiveDiscover" />
      </app-server-cluster>
      ```
   g) Save and close the file.
3. Verify that your Dgraph clusters are configured correctly by reviewing the `LiveDgraphCluster.xml.` file.
   A sample file is shown below:
   ```
   <dgraph-cluster id="LiveDgraphCluster1" getDataInParallel="true" enabled="true">
   ```

```
    <dgraph ref="DgraphA1" />
</dgraph-cluster>

<dgraph-cluster id="LiveDgraphCluster2" getDataInParallel="true" enabled="true">

    <dgraph ref="DgraphA2" />
</dgraph-cluster>
```

4. Modify the sample usage collection configuration file:
   a) Open the `UsageCollectionConfig.xml` file.
   b) Specify the logging directory by modifying the `<usage-log-dir>` element:

      Enter paths relative to the EAC application directory. For example, the configuration below outputs logs to `<app dir>\logs\usage`:

      ```
      <usage-collector id="UsageCollector">
          <usage-log-dir>./logs/usage</usage-log-dir>
      </usage-collector>
      ```

   c) Modify the cluster elements to reference the Application Server clusters configured in Step 2 (if you are using Experience Manager) and the Dgraph clusters from Step 3:

      ```
      <usage-collector id="UsageCollector">
          <usage-log-dir>./logs/usage</usage-log-dir>
          <dgraph-cluster ref="LiveDgraphCluster1" />
          <dgraph-cluster ref="LiveDgraphCluster2" />
          <app-server-cluster ref="LiveAppServerCluster" />
      </usage-collector>
      ```

   d) Save and close the file.

Usage information is stored in memory in the Dgraph or Assembler, and is cleared when you restart the component. After configuring logging, you must schedule usage collection to retrieve the data and store it as log files.

Scheduling usage collection

You can save usage information to log files by running the `collect_usage` batch or shell script in the `control` directory of your deployed application on disk. To automate this process, you can set the script to run in Windows Task Scheduler on Windows, or as a `cron` job on UNIX.

Oracle recommends running usage collection at least every 10 minutes. If your environment is configured to run baseline updates at regular intervals (or otherwise restarts the Dgraph or Assembler frequently) you should configure your logging intervals such that you do not lose more than 10% of the log data stored in memory when a component restarts. For example, in an environment where you run a baseline update on the hour, you should collect usage logs at ten minute intervals starting at 9 minutes past the hour. On UNIX, this might look like the following:

```
9,19,29,39,49,59 * * * * usr/local/endeca/apps/Discover/control/collect_usage.sh
```

The log files created by the script are output to the directories configured in the `<app dir>\config\script\UsageCollectionConfig.xml` file, and are rolled on a monthly basis.

Usage log format

Log entries include a timestamp, entry type, application name, URL, start time, seconds since start, and cumulative usage.

The information is formatted as follows:

Value	Information
<entry timestamp>	Time the entry was logged, in the format [YYYY-MM-DD]T[HH:MM:SS]Z. The Z denotes that times are logged for the GMT / Zulu timezone.
<entry type>	Whether the entry logs Assembler or MDEX usage. Possible values are asm or mdex
<application>	The name of the EAC Application associated with the log entry.
<url>	The URL polled to retrieve the information.
<start time>	The start time of the associated Dgraph or Assembler server, in the format [YYYY-MM-DD]T[HH:MM:SS]Z. The Z denotes that times are logged for the GMT / Zulu timezone.
<seconds since start>	The time, in seconds, since <start time>.
<cumulative usage>	Usage since <start time>. For an Assembler, this is the number of assem¬ble() calls.

Each entry is logged on a new line, and values in each entry are comma-delineated:

```
#Logger timestamp,entry type,eac app name,polled url,asm/ dgraph start time,seconds
 since asm/ dgraph start,cumulative usage
<entry timestamp>,<entry type>,<application>,<url>,<start time>,<time offset>,<cu¬
mulative usage>
```

Example

```
#Logger timestamp,entry type,eac app name,polled url,asm/ dgraph start time,seconds
 since asm/ dgraph start,cumulative usage
2013-08-01T14:00:00Z,asm,Discover,http://10.129.150.2:8006/discover/usage.xml,2013-
08-01T12:00:00Z,7200,371
2013-08-01T15:00:00Z,asm,Discover,http://10.129.150.2:8006/discover/usage.xml,2013-
08-01T12:00:00Z,10800,1092
2013-08-01T16:00:00Z,asm,Discover,http://10.129.150.2:8006/discover/usage.xml,2013-
08-01T12:00:00Z,14400,1891
2013-08-01T17:00:00Z,asm,Discover,http://10.129.150.2:8006/discover/usage.xml,2013-
08-01T16:30:00Z,1800,950,
2013-08-01T18:00:00Z,asm,Discover,http://10.129.150.2:8006/discover/usage.xml,2013-
08-01T16:30:00Z,5400,1900
2013-08-01T19:00:00Z,asm,Discover,http://10.129.150.2:8006/discover/usage.xml,2013-
08-01T16:30:00Z,9000,2100
2013-08-01T20:00:00Z,asm,Discover,http://10.129.150.2:8006/discover/usage.xml,2013-
08-01T16:30:00Z,12600,2500
```

A log for Dgraph usage would look exactly the same as above, except with an <entry type> of dgraph and <cumulative usage> totaling the number of Dgraph queries.

Generating usage reports from log files

The Deployment Template creates a `generate_usage_report` batch or shell script that you can run to generate reports from usage logs disk. The reports let you quickly determine peak usage times and volume.

You must configure usage logging and have a set of logs against which to run the script.

The script generates the following reports:

- Daily usage — This report, `dgraph-daily-usage.txt`, lists the number of Dgraph queries, the time (in seconds) for which usage logs do not exist, and the number of Dgraphs queried:

```
#report generated for Discover on 2013-09-09T12:00:00Z
#date,usage,unreported time in seconds,# dgraph
2013-08-01,30085,600,2
2013-08-02,29001,1200,2
...
```

- Weekly usage — This report, `dgraph-weekly-peak-usage.txt`, lists the peak Dgraph usage for each week for which usage logs are available, the time (in seconds) for which logs do not exist on the peak date, and the number of Dgraphs queried:

```
#report generated for Discover on 2013-09-09T12:00:00Z
#from date,to date,peak usage date,peak usage,unreported time in seconds on
peak day,# dgraph
2013-07-28,2013-08-03,2013-08-01,30085,600,2
2013-08-04,2013-08-10,2013-08-09,28801,1200,2
...
```

- Yearly and monthly usage — This report, `dgraph-yearly-peak-usage.txt`, lists the peak Dgraph usage for each calendar year for which usage logs are available, the time (in seconds) for which logs do not exist on the peak date, and the number of Dgraphs queried. It also lists the same metrics for each month:

```
#report generated for Discover on 2013-09-09T12:00:00Z
#from date,to date,peak usage date,peak usage,unreported time in seconds on
peak day,# dgraph
2013-01-01,2013-12-31,2013-08-01,30085,600,2

#peak usage values for each month
#from date,to date,peak usage date,peak usage,unreported time in seconds on
peak day,# dgraph
2013-08-01,2013-08-31,2013-08-01,30085,600,2
2013-09-01,2013-09-30,2013-09-03,34905,800,2
...
```

Additionally, the script generates dailly, weekly, and yearly usage reports for the Assembler if you are running Oracle Commerce Guided Search with Experience Manager. These files are named `asm-daily-usage.txt`, `asm-weekly-peak-usage.txt`, and `asm-yearly-peak-usage.txt`, respectively.

To generate usage reports from log files:

1. Navigate to the `control` directory of your deployed EAC application on disk.
   For example, `Endeca\apps\Discover\control`.
2. Run the `generate_usage_report` batch or shell script.
   Optionally, pass in the following command line arguments:
   - `--usage-log-dir` — The directory where usage logs are stored. This defaults to `logs/usage`.
   - `--usage-report-dir` — The directory to store usage reports. This defaults to `reports/usage`.

- `--overwrite` — Whether to override existing reports (defaults to `yes`).When set to `no`, the script throws an error if reports already exist in the `--usage-report-dir` directory.

Important: Oracle recommends automatically generating usage reports and backing up existing reports on a daily basis. You should retain backups for at least one calendar year.

Chapter 11

Administering the MDEX Engine

This section covers administrative operations for the MDEX Engine, including the Dgidx and Dgraph components, as well as the Relevance Ranking Evaluator.

Administering the Dgidx

This section describes the Dgidx process and outlines administrative tasks that help ensure its proper operation. It also contains tips for improving Dgidx performance and troubleshooting problems.

About Dgidx processing

The Dgidx component organizes acquired records into a structure, partially precomputes results that are used by the Dgraph, and creates the Endeca index.

The Dgidx is part of the MDEX Engine installation package, and as such it is installed on both the ITL server and the MDEX Engine server. Since Dgidx is part of the offline processing that runs during baseline updates, the best practice is to run it on the ITL server (the Deployment Template implements this practice with its scripts).

At a very high level, the Dgidx process consists of the following steps:

1. It reads dimensions and records into its process memory. Records are loaded gradually, processed, and released as needed.
2. It indexes records one at a time, adding the relevant information to all indexes. These indexes are stored on disk.
3. It merges the index generations.

As part of its processing, Dgidx sorts the acquired records and produces navigational, text, and wildcard indexes.

Dgidx memory usage

Dgidx relies on the operating system caching of indexes on disk and uses memory-mapped I/O to retrieve its indexes. This affects the size of the virtual memory allocated to the Dgidx working process, which can increase periodically.

For more information about memory considerations and the way the MDEX Engine uses memory, see the *Oracle Commerce MDEX Engine Performance Tuning Guide*.

Running the Dgidx process with the Deployment Template

You typically start the Dgidx process by using the `runcommand` utility of the Deployment Template.

The `runcommand` utility lets you start the Dgidx indexing process on a remote Endeca data processing server.

To run the Dgidx process:

Run the command from the Deployment Template:

Option	Description
Windows	`runcommand.bat MyDgidx run`
UNIX	`./runcommand.sh MyDgidx run`

Where `MyDgidx` is the `id` value of a `dgidx` element specified in the `AppConfig.xml` file, such as `<dgidx id="MyDgidx" host-id="ITLHost">`.

Note: In addition to running Dgidx with the Deployment Template `runcommand`, you can also run the Dgidx executable binary from the command line. This can be useful for troubleshooting purposes.

Running the Dgidx binary at the command prompt

In rare instances, you may need to run the Dgidx binary from the command prompt outside of your EAC and Deployment Template configuration. This is helpful if the Dgidx process fails, and you need to identify whether a possible cause of the problem is in the Deployment Template scripts, the EAC, or Dgidx itself.

Note: You should only run the Dgidx binary directly at the command prompt in rare instances when you need to replicate a Deployment Template job in a separate testing environment. If you need to re-run a particular process with its normal settings, Oracle recommends using the `runcommand` of the Deployment Template.

Before running the Dgidx binary at the command prompt, do the following prerequisite tasks:

- Copy the necessary files into the locations where the Dgidx process can find them.
- Create the `dgidx_output` directory. It must exist prior to running Dgidx, but is not created automatically as part of running it from the command prompt.

To run the Dgidx binary at the command prompt:

1. Go to the `%ENDECA_MDEX_ROOT%\bin` directory on Windows, or to `$ENDECA_MDEX_ROOT/bin` on UNIX.
2. Enter the Dgidx command, such as `dgidx` (on Windows) or `dgidx` on UNIX.

   The usage information for the Dgidx binary is displayed.

   The parameters for Dgidx depend on your specific implementation. Examine the Dgidx command usage to construct the command you will run in the next step.
3. Run the command, which will be similar to the following example:

Option	Description

Windows	`%ENDECA_MDEX_ROOT%\bin\dgidx --out \localdisk2\endeca\`*`ver`*`¬`
	`sion``\dgidx.log --dtddir %ENDECA_ROOT%\conf\dtd \localdisk2\ende¬`
	`ca\`*`version`*`\endeca \localdisk2\endeca\`*`version`*`\dgidx_output\endeca`

Option	Description
UNIX	`$ENDECA_MDEX_ROOT/bin/dgidx --out /localdisk2/endeca/version/dgidx.log --dtddir $ENDECA_ROOT/conf/dtd /localdisk2/endeca/version/endeca /localdisk2/endeca/version/dgidx_output/endeca`

This command points to the location of the Dgidx log, its DTD directory, and the Dgidx output file.

Tips for speeding up indexing time

While Dgidx is optimized for best performance, you may adjust your configuration and front-end application to speed up indexing time.

To speed up indexing, consider using fewer of the following:

- Records or fields.
- Text-searchable fields.
- Wildcard-searchable fields.

 **Note:** For details on performance of specific features, see the *Performance Tuning Guide*.

Troubleshooting Dgidx failures

This topic lists major causes of possible Dgidx crashes, to help you identify and fix them, or prevent them from occurring.

Troubleshooting Deployment Template failures with Dgidx

Dgidx is often the first component to fail, because it is one of the first components that needs to run within the Deployment Template scripts.

For example, you may see the following Dgidx failure:

```
SEVERE: Batch component 'Dgidx' failed. Refer to component
 logs in /usr/local/endeca/[version]/endeca/project/sample
/control/.././logs/dgidxs/Dgidx on host ITLHost.
Occurred while executing line 32 of valid BeanShell script:
[[

29| Forge.archiveLogDir();
30| Forge.run();
31| Dgidx.archiveLogDir();
32| Dgidx.run();
33|
34| // distributed index, update Dgraphs
35| DistributeIndexAndApply.run();

]]
```

Use the following steps to investigate Dgidx failures in the Deployment Template:

- Locate and examine the Dgidx error log.
- Verify that the MDEX Engine is also installed on the data processing (ITL) server, because Dgidx is part of the MDEX Engine installation.

- Check the `eac.properties` file, which is located under your Platform Services `workspace/conf` folder (for example, `/endeca/PlatformServices/workspace/conf/eac.properties`). Verify that the `com.endeca.mdexRoot` property is set to the correct location and version of your `MDEX_ROOT` (for example, `/usr/local/endeca/MDEX/[version]`), and restart the HTTP service.
- Run Dgidx from the command line. This way, you are accessing the Dgidx directly, without the layer of the Deployment Template and EAC configuration.

  It is useful to run Dgidx directly for debugging purposes. For example, if you notice that Dgidx fails when running it with the Deployment Template `runcommand`, but runs successfully from the command line, this means that the issue is either with the EAC or the Deployment Template, as opposed to the problems in the data.

Troubleshooting memory allocation failures with Dgidx

In rare cases the Dgidx process may fail due to running out of virtual memory or swap space that it requires to run successfully.

For example, you may see a memory allocation error similar to the following:

```
FATAL DATE 13:50:40.753 UTC DGIDX {dgidx,baseline}:
memory allocation failure
-------------------------------------------------------------
Endeca fatal error detected.
-------------------------------------------------------------
```

Use the following tips to troubleshoot memory allocation crashes:

- Locate and examine the Dgidx error log.
- Run another baseline update and use `vmstat` (on UNIX) to closely monitor the Dgidx memory usage and the amount of memory and swap space available on the ITL server. You can save the output of `vmstat` and explore it to identify whether the amount of free and swap memory drops or remains sufficient.
- On UNIX, use the `top` and `prtconf` commands and explore their output.
- Temporarily shut down some processes running on this server and examine whether the Dgidx process continues to fail consistently.
- If the process does not fail, note its peak virtual memory usage while it is running.

Dgidx logs

To locate your application's Dgidx logs, consult the Dgidx definition in the `AppConfig.xml` file of the Deployment Template.

By default, if your application name is `MyApp` and your Dgidx process name is `Dgidx1`, the Dgidx logs are located in `MyApp/logs/dgidxs/Dgidx1.`

For example, the following Dgidx definition from the `AppConfig.xml` lists the location of the Dgidx logs:

```
# Dgidx
   #
 -->
 <dgidx id="Dgidx1" host-id="ITLHost">
   <properties>
    ...
   </properties>
   <directories>
     <directory name="incomingDataDir">./data/forge_output</directory>
     <directory name="configDir">./data/forge_output</directory>
   </directories>
   <args>
```

```
      <arg>-v</arg>
    </args>
    <log-dir>./logs/dgidxs/Dgidx1</log-dir>
    <input-dir>./data/dgidxs/Dgidx1/dgidx_input</input-dir>
    <output-dir>./data/dgidxs/Dgidx1/dgidx_output</output-dir>
    <data-prefix>Test-part0</data-prefix>
    <temp-dir>./data/dgidxs/Dgidx1/temp</temp-dir>
    <run-aspell>true</run-aspell>
  </dgidx>
```

The following examples list some of the typical items in a Dgidx log file and explain them:

Note: You may notice that Dgidx also creates three properties of type `admin` on each record, named `Endeca.DataSize`, `Endeca.NumAssigns`, and `Endeca.NumWords`. These properties are visible in the Dgidx log and in the key properties in the Dgraph. Because these properties may not be supported in future releases, Oracle recommends that you ignore these properties in the log and avoid building front-end application logic around them.

Example 1

```
=== DGIDX: Finished phase
"Read raw dimensions,
 properties, and records"
=== Phase Time: 19 minutes, 44.11 seconds
```

This log entry indicates that the Dgidx is reading in all data and creating all indexes.

Example 2

```
$->tail Dgidx.log
Sorting... 22.16 seconds
Writing cycle 255 to temporary file
Parsing text fields...
...
179,600,000 text fields,
5,726,985,428 elements
179,700,000 text fields,
5,730,167,391 elements
...
```

This log entry indicates the following:

- `text fields`. Text fields are individual entries in a record that Dgidx adds to its index for dimension search, record search, or both. The Dgidx output log lists the total number of text fields in each dimension or property of the record, then periodically outputs how many text fields have been processed during text search indexing.

  Because text search indexing operates on text fields from all dimensions or properties, the totals printed periodically can be greater than the totals from each.

  Text fields contain one or more terms. The large difference between the number of text fields and the number of elements is due to records containing large numbers of terms per property and/or dimension.

- `elements`. Elements represent the number of individual terms or term-related objects sent to the index.

  Elements are sorted and stored for text search (including dimension search, if applicable).

For example, consider an employee record: `Name: John Lee Age: 24 Hired: 2008-08-14 Descrip¬tion: Permanent`. If `Name` is a dimension enabled for dimension search, and `Description` is a property enabled for text search, then Dgidx would represent this record in the log as having 2 text fields and 3 elements.

Note: These numbers are approximate and reflect on the magnitude of items in the index. Do not interpret these numbers as the exact number of unique terms in the data corpus. Among other considerations, a single input word generates multiple index elements, and for different types of its indexes Dgidx uses different types of unique elements.

Dgidx log details for text search indexing

You can examine the text search indexing portion of your Dgidx logs and use the information in this topic to identify which items in the log contribute to indexing time.

Stemming and spelling do not affect the log numbers in the text search indexing portion of Dgidx logs. However, wildcard search increases the number of entries made to the index.

The following items related to text search indexing appear in the Dgidx log:

Dgidx log item	Description
`Records`	Corresponds to the actual number of records and dimensions listed in the `rec¬search_indexes.xml` file. This represents the number of records and dimensions that need to be indexed by Dgidx for text search.
`Text fields`	Corresponds to the total number of pairs that are available for text search. (Pairs are associations between a dimension or property and their corresponding values.)
`Entries`	Corresponds to the total number of entries that were made to the index.  **Note:** If wildcard search is enabled, this increases the number for `entries`.
`Rec`	Reflects the standard index.
`RecWC`	Reflects the wildcard index that is created in addition to the standard index.

Dgidx handling of records with missing or duplicate record spec values

When Dgidx processes records with missing or duplicate record specifier (or spec) values, it completes successfully, but produces a very large log file.

The log contains WARN-level messages that print entire records. These warning messages appear in the Dgidx log because records are improperly assigned property values from the project's configured record spec property.

- If the application has a record spec property defined, each record must contain a single unique value from that property. If a record contains no record spec property value, Dgidx prints the "`record... has no value assigned to it from any record specifier property`" warning, as in the following example:

```
WARN 08/16/09 15:49:23.897 UTC DGIDX {dgidx,baseline}: The record
with the following properties has no value assigned to it from any
record specifier property. This record cannot be modified with
rapid updates:
[Record Id=4]
Dimension[6200,"Wine Type"]: Value[8013] "White"
Dimension[8,"Region"]: Value[4294967254] "Mendocino Lake"
[...]
Property["P_Body"]: Value[0xce1bd0] "Ripe"
```

```
Property["P_DateReviewed"]: Value[0xce1470] "02/28/95"
[...]
```

- If a record contains a record spec property value that is already in use by another record (a non-unique value), Dgidx prints the "`Two records cannot share the value... for specifier property`" warning, as in the following example:

```
WARN 08/16/09 15:49:23.897 UTC DGIDX {dgidx,baseline}: Two records
cannot share the value "34699" for specifier property "P_WineID";
removing this record:
[Record Id=2]
Dimension[6200,"Wine Type"]: Value[8013] "White"
Dimension[8,"Region"]: Value[4294967282] "Sonoma"
[...]
Property["P_Body"]: Value[0xce1870] "Crisp"
Property["P_WineID"]: Value[0xcd6e40] "34699"
[...]
```

In either case, Dgidx prints the record's property and dimension values into the log so that it can be identified and corrected in a future update.

There is no way to suppress this display of the full record in the cases mentioned above. Instead, you should correct the record spec problems noted in the log by modifying the project's Forge pipeline or its record spec property selection. Assign record spec property values to records that lack them, and ensure that each record is assigned a unique record spec value so that duplicates do not occur. You can use the record details printed into the Dgidx log to identify the affected records even if they do not have unique record spec property values.

Variations in Dgidx indexing time

When you analyze Dgidx logs, you may notice that periodically indexing times are longer than you might expect.

The indexing operation may appear to you like your normal addition of records, and not a major or minor shift in the character of the existing records that would explain the change in indexing time.

Dgidx periodically goes through a merging process of many indexing generations. In particular, when the number of generation files becomes large, Dgidx merges them together to reduce the number of open files. Dgidx does this extra merge step when the number of generation files exceeds 200.

Administering a Dgraph

This section describes basic administrative tasks for the Dgraph.

Checking Dgraph with the ping command

A quick way of checking the availability of a Dgraph is by accessing the URL as described in this topic.

To check the whether a Dgraph is running:

For a Dgraph, access:

```
http://DgraphServerNameOrIP:DgraphPort/admin?op=ping
```

The Dgraph quickly returns a lightweight HTML response page with the following content:

```
dgraph host:port responding at date/time
```

Note: You can also view the MDEX Engine Statistics page to check as to whether the MDEX Engine is running and accepting queries.

Specifying arguments to the Dgraph in the Deployment Template

If you are using the Deployment Template, you specify Dgraph arguments in the `AppConfig.xml` file under the `<dgraph-defaults>` element.

To specify arguments to the Dgraph:

Add arguments in the `<args>` element of `<dgraph-defaults>` in the `AppConfig.xml` file, similar to the following example:

```
<dgraph-defaults>
<properties>
  ...
</properties>
<directories>
  ...
</directories>
<args>
  <arg>--threads</arg>
  <arg>2</arg>
  <arg>--spl</arg>
  <arg>--dym</arg>
</args>
<startup-timeout>120</startup-timeout>
</dgraph-defaults>
```

Note: Arguments that take a value, such as `--threads`, should be separated into two consecutive `<arg>` elements.

Collecting debugging information

Before attempting to debug an issue with the MDEX Engine, collect the following information.

- Hardware specifications and configuration.
- Description of the Endeca topology (servers, number of Dgraphs).
- The data from the MDEX Engine Statistics page.
- Your `AppConfig.xml` file.
- The contents of the pipeline directory.
- Dgraph input.
- Partial update files.
- Description of typical partial updates.
- Description of which Dgraphs are affected.

The logs created by the Dgraph

The Dgraph creates up to five logs, although some of these logs depend on your implementation and the Guided Search components that you may be using. This topic provides a summary of these logs.

You can use these Dgraph logs to troubleshoot MDEX Engine queries, or to track performance of particular queries or updates.

Dgraph request log

The Dgraph request log is always created. You can use it to debug both queries and update processing. It contains one entry for each query processed.

You can set the path to this log by using one of these methods:

- In the `AppConfig.xml` file of the Deployment Template, specify the path set by the Dgraph component's `<log-dir>` element. Based on it, the Deployment Template creates the file in the following format: `$component.reqlog`. For example, `Dgraph1.reqlog` is the path to the Dgraph log for a Dgraph component with the name `Dgraph1`.
- If you are using the Dgraph from the command line, create the path to the request log in the Dgraph working directory with the filename `dgraph.reqlog`.

Details about the Dgraph request log can be found in the *Performance Tuning Guide*.

Dgraph error log

The Dgraph error log is created only if you redirect `stderr` to a file, using a command line or a `dgraph --out` flag. Otherwise, error messages appear in `stderr`.

The Dgraph error log includes startup messages as well as warning and error messages. It can be configured via Dgraph flags (such as `-v`). In addition, the `config?op=log-enable` operation, described in an appendix to this book, makes it possible to record more details about specific features.

In the `AppConfig.xml` file, you can specify the path set by a Dgraph component's `<log-dir>` element, using the format `$component.log`. For example, `Dgraph1.log` is the path to a Dgraph component with the name `Dgraph1`.

Update log

The Dgraph update log is created only if you run the Dgraph with the `--updatelog` flag, or through the Deployment Template.

You can set the path to this log in the Deployment Template in the Dgraph component `<log-dir>` element, in the following format: `$component.updatelog`. For example, `Dgraph1.updatelog` is the path to the Dgraph update log for a Dgraph component with the name `Dgraph1`.

Process start log

The Dgraph process start log is created in the Deployment Template and EAC environments for messages which occur during the Dgraph process startup. This log is typically empty. It may sometimes be useful for debugging Deployment Template or EAC issues.

You can set the path to this log by the Dgraph component `<log-dir>` element, with the format `$compo¬nent.start.log`. For example, `Dgraph1.start.log` identifies the process start log for a Dgraph component with the name `Dgraph1`.

EQL per-query statistics log

The EQL per-query statistics log is useful if you use Endeca Query Language (EQL). It is created only if you run the Dgraph with the `--log_stats` *`path`* flag. This log is not created in the default configuration driven by the Deployment Template.

For more information on the Endeca Query Language, see the *Advanced Development Guide*.

Troubleshooting baseline update failures

To debug baseline update failures, examine the Dgraph request log and the baseline update log first, followed by the EAC process logs.

Use the following recommendations:

- **Review Dgraph request logs**. Review the logs around the time of the baseline update failure, to rule out issues in the Dgraph.

  Notice the times when health checks were sent to the Dgraph, the Dgraph was restarted, the partial updates were issued, and the last query was issued.

  For example, this modified abstract from the Dgraph request log shows activity for a period of time:

  ```
  12096521815/1/09 14:29 last search query
  12096522265/1/09 14:30 health check
  12096526095/1/09 14:36 last health check for x time
  12096571605/1/09 15:52 health checks resume
  12096574435/1/09 15:57 last empty health check
  12096601195/1/09 16:41 Dgraph startup
  12096601435/1/09 16:42 first query
  ```

  Notice that the Dgraph did not receive any requests besides health checks for a period of time from 14:29 to 15:57. The log does not include error messages. The Dgraph was not restarted during this time. These observations indicate that the problem that led to the baseline update failure in this example possibly occurred outside of the Dgraph.

- **Review baseline update.out logs**. For example, in the case below, observe that an error occurred while stopping the Dgraph component:

  ```
  [05.01.09 10:07:54] INFO: Stopping component 'Dgraph1'.
  [05.01.09 10:17:54] SEVERE: Error communicating with EAC agent while
  stopping component.
  Occurred while executing line 5 of valid BeanShell script:
  ```

  To investigate further the reason for why the EAC was not able to stop the Dgraph component, examine the logs for EAC processes and increase their verbosity.

- **Increase the verbosity of the EAC process logs**.

  Specify the EAC logging configuration in *[ENDECA_CONF]*`/conf/logging.properties` file. Set the log level for `com.endeca.eac.invoke`, `com.endeca.eac.process` and `com.endeca.eac.main` to `FINE`. This provides additional debug information, if the baseline update process fails again.

  Note: Monitor the size of the *ENDECA_CONF*`/logs` directory, to ensure it does not fill up the disk.

  To continue with the example, the EAC process logs may, for instance, indicate an outage of a hardware component between the Web server and the Dgraph server. These logs may further assist you if the baseline update fails again.

Troubleshooting partial updates

This topic contains several pointers to help you troubleshoot partial updates.

Accessing failed update files

The default directory that the MDEX Engine uses for storing the failed update files is `<updatedir>/failed_updates/`.

You can use the `--failedupdatedir <dir>` command on the Dgraph to specify another directory for these files.

Permission to access index directories

If you are encountering Dgraph failures associated with partial updates, ensure that the Dgraph has permission to access the index directories.

The Dgraph checks permissions on the index directories before applying partial updates. If the required read/write permissions are missing, the Dgraph issues an error in the standard error log, and fails to apply the update.

If the Dgraph is running in verbose mode, it also logs the path to the index directories to which the Dgraph does not have read/write permissions.

The Dgraph checks permissions on these directories in the `<app dir>/dgidx_output/myApp_indexes`:

- `/committed`
- `/generations`

(These filepaths assume that the Deployment Template scripts were used to set up the application.)

Both of these directories should have read and write permissions to allow the Dgraph to access them. However, these permissions may be reset, due to file system issues or hardware maintenance issues combined with the Endeca implementation's topology. This may make these directories unaccessible by the Dgraph.

Identifying connection errors

If the Dgraph standard out log contains `connection broken` messages, although it may look like the problem occurred with the Dgraph, the actual cause of the problem is usually a broken connection between the server that hosts the front-end application and the server that hosts the Dgraph.

In the case of connection errors, various parts of the Guided Search implementation issue the following error and warning messages:

- The .NET API throws the following exception:

```
Endeca.Navigation.ENEConnectionException:
Error reading from the connection. The operation has timed out
   at Endeca.Navigation.OptiBackendRequest.GetContent()
   at Endeca.Navigation.OptiBackend.GetNavigation(OptiBackendRequest req)
   at Endeca.Navigation.HttpENEConnection.Query(ENEQuery neq)
```

  The Java API throws a similar exception.

- The Dgraph standard out log contains warnings similar to the following:

```
WARN [DATE TIME] UTC (1239830549803)
DGRAPH {dgraph}: Aborting request: connection broken: client 10.10.21.21
```

- And finally, the Dgraph request log contains an abnormal `status 0` message similar to the following:

```
1239830549803 10.6.35.35 - 349 0 19.35 0.00 0 - 0 0 - -
```

Typically, the `connection broken` message means that the Dgraph encountered an unexpected failure in the connection between the client and the Dgraph. This type of error may occur outside the Dgraph, such as in the network, or be caused by the timeout of the client application session.

Investigate the connection between the client and the Dgraph. For example, to prevent timeouts of the client application sessions, you may decide to implement front-end application retries.

Troubleshooting socket and port errors with Dgraph

The Dgraph cannot start if its process cannot bind to a socket and its port cannot initialize. This error tends to occur when you upgrade the MDEX Engine and attempt to use a port that is already occupied by another process on your server.

The baseline update script from the Deployment Template completes, but the MDEX Engine does not start. The following errors appear in the Dgraph log:

```
ERROR (date and time)
DGRAPH {dgraph,baseline}: Unable to bind
to socket [err=`Result too large',errno=34]
FATAL (date and time)
DGRAPH {dgraph,baseline}: Unable to initialize the
main server port: 8000
```

The `"Unable to bind to socket"` errors usually indicate that the port in question is already in use by another process.

The Windows command-line utility `netstat -ano` lists all ports in use along with the process ID of the process using them. Use this utility to identify the process ID occupying port 8000, and locate that process in the Windows Task Manager to confirm that it is used by another process. This prevents the Dgraph from starting.

To identify ports in use on your Windows system:

1. Run `netstat -ano`

   This command lists ports and process IDs of all processes that are running.
2. Examine which process occupies the port that the Dgraph is trying to use. In this example, it is port 8000.
3. Run the Dgraph on another port, or ensure that the previously occupied port can be freed to be used by the MDEX Engine.

Managing the Dgraph core dump files

In the rare case of a Dgraph crash, the Dgraph writes its core dump files on disk.

When the Dgraph runs on a very large data set, its in-memory representation of the index size may exceed the size of the physical RAM. If such a Dgraph process fails, it may need to write out potentially very large core dump files on disk.

To troubleshoot the Dgraph, it is often useful to preserve the entire set of core files written out as a result of such failures. When there is not enough disk space, only a portion of the files is written to disk until this process stops. Since the most valuable troubleshooting information is contained in the last portion of the core files, to make these files meaningful for troubleshooting purposes, it is important to provision enough disk space to capture the files in their entirety.

Two situations are possible, depending on your goal:

- To troubleshoot the Dgraph crash, provision enough disk space to capture the entire set of core files. In this case, the files will be saved at the expense of potentially filling up the disk.

- To prevent filling up the disk, you can limit the size of these files on the operating system level. In this case, with large Dgraph applications, only a portion of core files is saved on disk. This may limit their usefulness for debugging purposes.

Managing Dgraph crash dump files on Windows

On Windows, all Dgraph crash dump files are saved on disk by default. (The MDEX Engine uses the `MiniDump` function from the Microsoft `DbgHelp` library.)

Provision enough disk space to accommodate core files based on this estimate:

- The projected upper limit for the size of these files is equal, at a maximum, to the size of the physical memory used by the MDEX Engine plus index size. Often the files take up less space than that.

Managing Dgraph core dump files on Linux and Solaris

Oracle recommends using the `ulimit -c unlimited` setting for Dgraph core dump files. Non-limited core files contain all Dgraph data that is resident in memory (RSS of the Dgraph).

Since large MDEX applications may take up all the available RAM, the core dump files can also grow large and take up the space equal to the size of the physical RAM on disk plus index size.

Note: For RSS discussion, see the *MDEX Engine Performance Tuning Guide*.

Provision enough disk space to accommodate core files based on this estimate:

- The projected upper limit for the size of these files should be equal, at a maximum, to the size of the physical RAM. Often the files take up less space than that.

Note: If you are not setting `ulimit -c unlimited`, you could be seeing the MDEX Engine crashes that do not write any core files to disk, since on some Linux installations the default for `ulimit -c` is set to 0.

Alternatively, to limit the size of core files, you can use the `ulimit -c <size>` command (although this is not recommended). If you set the limit size in this way, the core files cannot be used for debugging, although their presence will confirm that the Dgraph had crashed. To be able to troubleshoot the crash, change this setting to `ulimit -c unlimited`, and reproduce the crash while capturing the entire core file. Similarly, to enable troubleshooting of the crash, you will need to reproduce the crash while capturing the full core file.

Dgraph administrative and configuration operations

Administrative and configuration operations make it possible to check Dgraph statistics, and enable or disable diagnostic flags without having to stop a running Dgraph. They also let you stop and restart the Dgraphs. This section lists URLs exposed by the Dgraph, describes the functions of each URL, and defines the syntax of those URLs.

Dgraph URL operation syntax

The syntax for Dgraph URL operations is documented below. Queries to these URLs are handled in the MDEX Engine's request queue like any other request—that is, they are handled on a first-come, first-served basis. They are also reported in the MDEX Engine request log like any other request.

In the following blocks, `<mdex_host>` refers to the hostname or IP address of the MDEX Engine and `<mdex_port>` refers to the port on which the MDEX Engine is listening.

For administrative operations, the syntax is:

```
http[s]://<mdex_host>:<mdex_port>/admin?op=<supported-operation>
```

For configuration operations, the syntax is:

```
http[s]://<mdex_host>:<mdex_port>/config?op=<supported-operation>
```

List of administrative operations
Administrative (or admin) operations listed in this topic allow you to control the behavior of the MDEX Engine from within the system.

The MDEX Engine recognizes the following admin operations:

Admin operation	Description
`/admin?op=help`	Returns the usage page for all of the admin operations.
`/admin?op=ping`	Checks the aliveness of an MDEX Engine and returns a lightweight message.
`/admin?op=flush`	Specifies when the MDEX Engine should flush its dynamic cache.
`/admin?op=exit`	Stops a running MDEX Engine.
`/admin?op=restart`	Restarts the MDEX Engine.
`/admin?op=audit`	Returns the MDEX Engine Auditing page.
`/admin?op=auditreset`	Resets the MDEX Engine Auditing page.
`/admin?op=stats`	Returns the MDEX Engine Statistics page.
`/admin?op=statsreset`	Resets the MDEX Engine Statistics page.
`/admin?op=logroll`	Forces a query log roll, with the side effect of remapping `stdout`.
`/admin?op=update`	Applies any partial update files to the MDEX Engine.
`/admin?op=updateaspell`	Rebuilds the aspell dictionary for spelling correction from the data corpus without stopping and restarting the MDEX Engine.
`/admin?op=updatehistory`	Shows a list of the update files that the MDEX Engine has processed recently.
`/admin?op=reload-services`	A Web services operation that reloads the application's main and library modules.

help
`/admin?op=help` returns the usage page for all of the administrative operations.

ping
`/admin?op=ping` checks the aliveness of an MDEX Engine and returns a lightweight message.

You can view the MDEX Engine Statistics page to check whether the MDEX Engine is running and accepting queries, but that comes with some overhead. A quicker way to check the availability of a Dgraph is by running the `ping` command.

The `ping` command returns a lightweight page that lists the MDEX Engine, the current date and time, such as the following:

```
dgraph example.endeca.com:8000 responding at Wed Oct 25 15:35:27 2009
```

You can use this operation to monitor the availability of the MDEX Engine, and as an availability check for load balancers.

flush
`/admin?op=flush` flushes the Dgraph cache.

The `flush` operation clears all entries from the Dgraph cache. It returns the following message:

```
flushing cache...
```

exit
`/admin?op=exit` stops a running MDEX Engine.

The `exit` operation puts the Dgraph on hold while all outstanding transactions (queries) are completed. Once all transactions have been completed, the Dgraph shuts down. This is the recommended way to shut down a Dgraph, as opposed to manually killing the process, since it gracefully completes all transactions and exits cleanly.

The output looks similar to the following:

```
Dgraph admin, OK
Dgraph shutting down at Wed May 20 11:23:08 2009
```

Both the Oracle Endeca Application Controller (EAC) and the deprecated Control Interpreter use the `exit` operation behind the scenes to stop an MDEX Engine. However, if the MDEX Engine in question was originally started by the EAC or the Control Interpreter, manually submitting this operation fails to permanently shut down the MDEX Engine, because the EAC or the Control Interpreter interprets the resulting MDEX Engine shutdown as a failure and restarts it immediately. MDEX Engines started by the EAC or the Control Interpreter should be shut down through the control framework that originally started them.

restart
`/admin?op=restart` restarts the Dgraph.

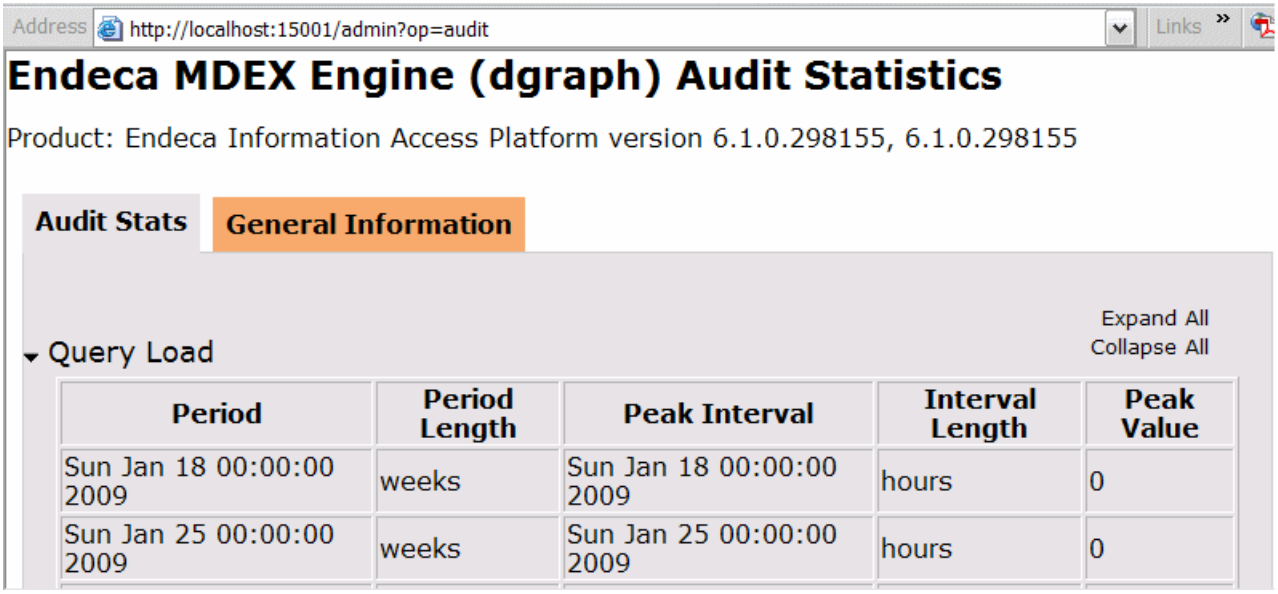
The `restart` operation acts similarly to the `exit` operation, except that after shutting down, the Dgraph restarts. This is the recommended way to restart a Dgraph, as opposed to manually stopping and starting the process, since it gracefully completes all transactions and exits cleanly before starting up again.

The `restart` operation returns output similar to the following:

```
Dgraph admin, OK
Dgraph restarting at Wed May 20 11:25:19 2009
```

audit
`/admin?op=audit` returns the MDEX Engine Auditing page.

The MDEX Engine Auditing page lets you view the aggregate Dgraph metrics over time. It provides the output of XML reports that track ongoing usage statistics. These statistics persist through process restarts. This data can be used to verify compliance with licensing terms, and is also useful for tracking product usage.



Note: For details about the MDEX Engine Auditing page, see the *Performance Tuning Guide*.

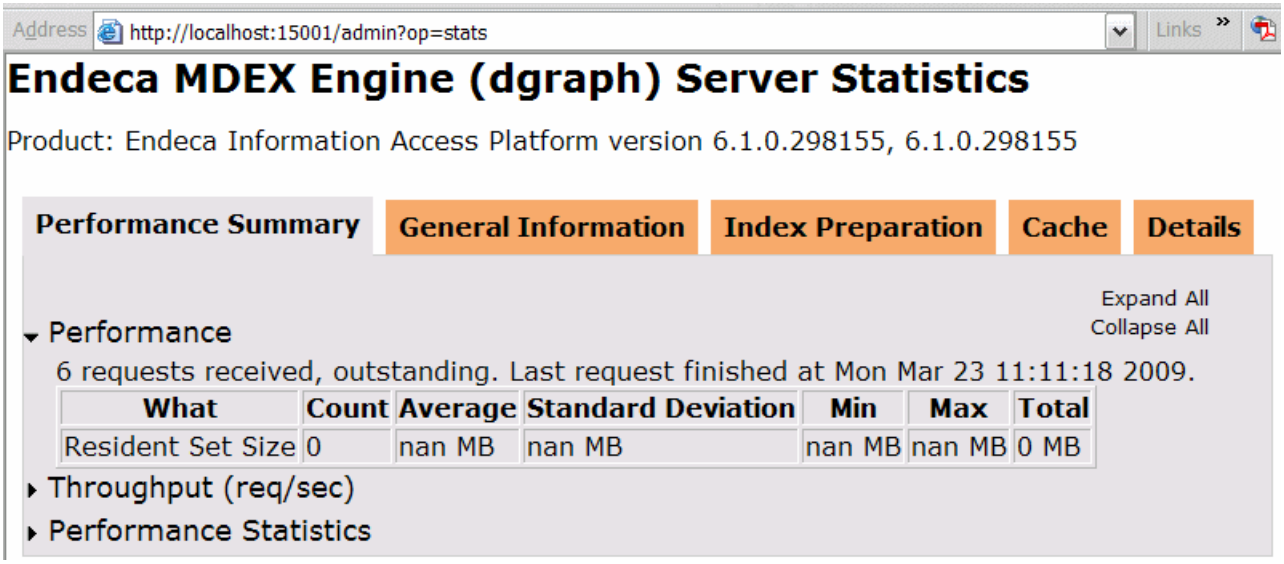
auditreset
`/admin?op=auditreset` resets the MDEX Engine Auditing page.

It returns the following message:

```
resetting auditing stats...
```

stats
`/admin?op=stats` returns the MDEX Engine Statistics page.

The MDEX Engine Statistics page provides a detailed breakdown of what the Dgraph is doing, and is a useful source of information about your Endeca implementation's configuration and performance. It provides information such as startup time, last data indexing time, and indexing data path. This lets you focus your tuning and load-balancing efforts. By examining this page, you can see where the Dgraph is spending its time. Begin your tuning efforts by identifying the features on the Details tab Hotspots section with the highest totals.



 **Note:** For details about the MDEX Engine Statistics page, see the *Performance Tuning Guide*.

statsreset
`/admin?op=statsreset` resets the MDEX Engine Statistics page.

The `statsreset` operation returns the following message:

```
resetting server stats...
```

logroll
`/admin?op=logroll` forces a query log roll, with the side effect of remapping `stdout`.

The `logroll` command returns a message similar to the following:

```
rolling log... Successfully remapped stdout/stderr to specified
path "C:\Endeca\apps\JanWine\logs\dgraphs\Dgraph2\Dgraph2.log".
Successfully rolled log file.
```

update
`/admin?op=update` applies any partial update files to the Dgraph.

This operation instructs the Dgraph to process the partial update files in its update directory. For more information on partial updates, see the *Partial Updates Guide*.

On receiving the URL update command, the Dgraph by default performs the following sequence of operations:

1. Continues processing queries concurrently with processing the update.
2. Checks the updates directory and uploads all partial updates that have not yet been uploaded.
3. Processes the update files and deletes them.

Processing updates from a single file

In some cases, you may need to run a partial update by pointing the Dgraph to a single file. In this case, run the `admin?op=update&updatefile=`*`filename`* option where `filename` is the name of an update file residing in the update directory.

If you have more than one file, rerun this command. The update file is deleted after the MDEX Engine successfully applies the results of the partial update.

updateaspell
The `admin?op=updateaspell` administrative operation lets you rebuild the aspell dictionary for spelling correction from the data corpus without stopping and restarting the MDEX Engine.

The `admin?op=updateaspell` operation performs the following actions:

- Crawls the text search index for all terms
- Compiles a text version of the `aspell` word list
- Converts this word list to the binary format required by `aspell`
- Causes the Dgraph to finish processing all existing preceding queries and temporarily stop processing incoming queries
- Replaces the previous binary format word list with the updated binary format word list
- Reloads the `aspell` spelling dictionary
- Causes the Dgraph to resume processing queries waiting in the queue

The Dgraph applies the updated settings without needing to restart.

Only one `admin?op=updateaspell` operation can be processed at a time.

The admin?op=updateaspell operation returns output similar to the following in the Dgraph error log:

```
...
aspell update ran successfully.
...
```

Note: If you start the Dgraph with the -v flag, the output also contains a line similar to the following:

```
Time taken for updateaspell, including wait time on any
previous updateaspell, was 290.378174 ms.
```

updatehistory

/admin?op=updatehistory shows a list of the update files that the Dgraph has processed since it was started.

The updatehistory operation returns output similar to the following:

```
Endeca Dgraph Server update directory history contents

Checking for update directory for directory "..\data\partition0\dgraph_input\up¬
dates\"

Files in update directory history

"..\data\partition0\dgraph_input\updates\\wine-
sgmt0.records.xml_2009.05.19.10.31.25"
"..\data\partition0\dgraph_input\updates\\wine-
sgmt0.records.xml_2009.05.19.10.30.08"
"..\data\partition0\dgraph_input\updates\\wine-
sgmt0.records.xml_2009.05.19.10.32.13"
```

reload-services

/admin?op=reload-services is a Web services operation that reloads the application's main and library modules.

The admin?op=reload-services operation causes the Dgraph to process all existing preceding queries, temporarily stop processing other queries and begin to process admin?op=reload-services. After it finishes processing this operation, the Dgraph resumes processing queries that queued up temporarily behind this request.

Note: admin?op=reload-services can be a time-consuming operation, depending on the number of XQuery modules that you have created and that have to be compiled.

List of configuration operations

Configuration (or config) operations listed in this topic allow you to modify configuration and logging information for the MDEX Engine from within the system.

The Dgraph recognizes the following config operations:

Config operation	Description
/config?op=help	Returns the usage page for all of the config operations.
/config?op=update	Loads and applies MDEX Engine configuration files from disk. **Note:** Oracle does not recommend using this operation, as it overwrites any configuration published from Workbench.

Config operation	Description
`/config?op=log-enable`	Enables verbose logging for one or more specified variables.
`/config?op=log-disable`	Disables verbose logging for one or more specified variables.
`/config?op=log-status`	Returns verbose logging status.

help
`/config?op=help` returns the usage page for all of the config operations.

update
The `config?op=update` operation loads any updated Dgraph configuration files (containing information about items such as thesaurus entries or dynamic business rules). The Dgraph applies the updated settings without needing to restart.

The `config?op=update` operation causes the Dgraph to drain all existing preceding queries, temporarily stop processing other queries and begin to process `config?op=update`. The operation loads and applies any updated configuration files. After it finishes processing these operations, the Dgraph resumes processing queries that queued up temporarily behind these requests.

Only one `config?op=update` operation can be processed at a time. This command is useful during development and debugging, when it is desirable to quickly load configuration changes without the interruption of restarting an MDEX Engine. It is not recommended when configuration exists in Workbench, as it uploads files from disk and not from the Endeca Configuration Repository.

Note: The `config?op=update` operation can be time consuming, depending on the number of configuration files the Dgraph has to process for an update.

The update operation returns output similar to the following:

```
Processing configuration file
"<full path to XML configuration file>"
Successfully processed configuration file
"<full path to XML configuration file>"
Finished processing config updates.
```

About MDEX Engine logging variables

You can use logging variables with config operations. This lets you obtain detailed information about Dgraph processing, to help diagnose unexpected application behavior or performance problems, without stopping and restarting the Dgraph or requiring a configuration update.

Although you can also specify general verbose logging at the Dgraph command line with the `--v` flag, it requires a Dgraph restart to take effect.

Logging variable operation syntax
MDEX Engine logging variables are toggled using the `/config?op=log-enable&name=<`*`variable-name`*`>` and `/config?op=log-disable&name=<`*`variable-name`*`>` operations.

You can include multiple logging variables in a single request. Unrecognized logging variables generate warnings.

For example, this operation:

```
/config?op=log-enable&name=merchverbose
```

turns on verbose logging for the dynamic business rule feature, while this operation:

```
config?op=log-enable&name=textsearchrelrankverbose&name=textsearchspellverbose
```

turns on verbose logging for both the text search relevance ranking and spelling features.

However, this operation:

```
config?op=log-enable&name=allmylogs
```

returns an `unsupported logging setting` message.

In addition, the following operations are supported:

- `/config?op=log-status` returns a list of all logging variables with their values (true or false).
- The special name `all` can be used with `/config?op=log-enable` or `/config?op=log-disable` to set all logging variables.

List of supported logging variables

The following table describes the supported logging variables that you can use with related config operations to toggle logging verbosity for specified features.

Logging variable names are not case sensitive.

Variable	Description
`verbose`	Enables verbose mode.
`requestverbose`	Prints information about each request to stdout.
`updateverbose`	Show verbose messages while processing updates.
`recordfilterperfverbose`	Enables verbose information about record filter performance.
`merchverbose`	Enables verbose debugging messages during merchandising rule processing.
`textsearchrelrankverbose`	Enables verbose information about relevance ranking during search query processing.
`textsearchspellverbose`	Enables verbose output for spelling correction features.
`dgraphperfverbose`	Enables verbose performance debugging messages during core Dgraph navigation computations.
`dgraphrefinementgroupverbose`	Enables refinement verbose/debugging messages.

log-enable

The `log-enable` operation lets you turn on verbose logging.

You can include multiple logging variables in a single request. Unrecognized logging variables generate warnings.

For example, this operation:

```
/config?op=log-enable&name=merchverbose
```

turns on verbose logging for the dynamic business rule feature, while this operation:

```
config?op=log-enable&name=textsearchrelrankverbose&name=textsearchspellverbose
```

turns on verbose logging for both the text search relevance ranking and spelling features.

However, this operation:

```
config?op=log-enable&name=allmylogs
```

returns an “unsupported logging setting” message.

log-disable
The `log-disable` operation lets you turn off verbose logging.

`/config?op=log-disable` with no arguments returns the same output as `log-status`.

log-status
The `log-status` operation returns a list of all logging variables with their values (true or false).

For example, if you have not enabled verbose logging on any feature, you would see a message similar to the following:

```
Logging settings:

verbose - FALSE
requestverbose - FALSE
updateverbose - FALSE
recordfilterperfverbose - FALSE
merchverbose - FALSE
textsearchrelrankverbose - FALSE
textsearchspellverbose - FALSE
dgraphperfverbose - FALSE
dgraphrefinementgroupverbose - FALSE
```

Configuring the Relevance Ranking Evaluator

This section discusses how to configure the Relevance Ranking Evaluator.

About the Relevance Ranking Evaluator

Relevance ranking arranges search results so that the retrieved records that are most likely to be relevant to search requests display first.

Business users can use the Relevance Ranking Evaluator to test and experiment with relevance ranking strategies. It uses their own data and MDEX Engine to model how different strategies produce results. It can be used as a testing tool to aid application development, as well as a search tuning tool. The Relevance Ranking Evaluator also provides performance metrics for each query. These metrics help business users evaluate the tradeoffs associated with run-time performance and some of the more complex relevance ranking modules.

Configuring an MDEX Engine

The Relevance Ranking Evaluator requires an MDEX Engine to produce its results.

It is not necessary to dedicate an MDEX Engine solely for use by the evaluator. The MDEX Engine that you use can be shared by another Web application or instance of your Website. Oracle recommends that the MDEX Engine be in a staging or QA environment, where the configuration is stable and does not compete for resources with production.

While you can use Relevance Ranking Evaluator with any MDEX Engine, there are certain configuration options that enhance the information and user experience. These configuration options are optional, and are not required to use the evaluator. For more information on these options, please refer to the *MDEX Engine Development Guide*.

- **whymatch**: Relevance Ranking Evaluator can take advantage of the information returned by either the `--whymatch` (recommended) or `--whymatchConcise` command-line MDEX Engine options. If one of

these options is enabled, Relevance Ranking Evaluator displays specific information about how each record was matched by the search terms that a business user entered. Use these options with care in a production environment, since they might degrade run-time performance.

- **wordinterp**: Relevance Ranking Evaluator can take advantage of the information returned by the `--wordinterp` command-line MDEX Engine option. If this option is enabled, Relevance Ranking Evaluator displays word interpretations considered for the search query, including thesaurus and stemming expansions. Again, use this option with care in a production environment, since it might degrade run-time performance.
- **DGraph.BinRelevanceRank**: When the MDEX Engine performs a relevance ranked search, it determines a relevance ranking *score* for each record, and uses that score internally to sort the results. This score, along with an indication of records that have the same the score, can be displayed by the Relevance Ranking Evaluator if the `--stat-brel` command-line MDEX Engine option is used. Like the other options, use this option with care in a production environment, since it might degrade run-time performance.

Configuring Relevance Ranking Evaluator properties

You can configure optional Relevance Ranking Evaluator properties in the `evaluator.properties` file.

The Relevance Ranking Evaluator offers additional configuration options that are geared towards overall application behavior changes. These options are for specific use cases, such as deployments with large data sets, large number of properties, or if you want to display record images. Some of the configuration options might be necessary if you observe performance issues when you use Relevance Ranking Evaluator.

To configure the Relevance Ranking Evaluator properties:

1. Navigate to `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF/conf` (on UNIX).
2. Open `evaluator.properties` in a text editor.
3. Add or edit any of the following properties:

Property	Description
`MDEX_VERSION`	To hide capabilities that are invalid for your version of the MDEX Engine, enter your MDEX Engine version as a three digit number. For example, `MDEX_VERSION=642`
`MDEX_PORT`	Set this property if you use an MDEX port other than the default of *15002*.
`MDEX_HOST`	Set this property so that it points to the correct host name for the MDEX Engine in your environment. The default if you do not set this is *localhost*.
`DISPLAYED_ATTRIBUTES`	Set this property for data with large number of properties and dimensions on each record, or if you want to limit the properties that display within the evaluator **Results** section. This option must be listed on a single line and must consist of a comma-delimited list of property and dimension names. For example, `DISPLAYED_ATTRIBUTES=P_DateReviewed,P_Description` The dimension and property names display within the **Search Results Comparison** section when you click **Show All Dimensions and Properties**. If you receive an error or observe performance issues when you use Relevance Ranking Evaluator, it might be necessary to set this property.
`MAX_DISPLAYED_ATTRIBUTES`	If you have a large number of dimensions and properties, you can set `MAX_DISPLAYED_ATTRIBUTES` to restrict the number returned for display.

Property	Description
	If this is not set, the default is 100 -- only the first 100 dimensions and properties appear. For example, to show the first 50 properties and dimensions: `MAX_DIS¬ PLAYED_ATTRIBUTES=50`
`STATIC_MODULE_PROPER¬ TIES`	If your data has a large number of properties, or if you want to limit the properties that are available for selection when choosing a property for the **Static** module, you can set `STATIC_MODULE_PROPERTIES`. Enter this property as a comma-delimited list of property names on a single line. The property names that you enter are the properties that display for selection when a user configures the **Static** module. If you receive an error or observe performance issues when attempting to configure the **Static** module, consider setting this option. For example: `STATIC_MODULE_PROPERTIES=P_Name,P_Description`
`RECORDS_TO_DISPLAY_OP¬ TIONS`	If you want to display a different number of records than the default set of options provides (10, 25 and 50), set this property. Enter it on a single pipe-delimited line within the `evaluator.properties` file. For example: `RECORDS_TO_DISPLAY_OPTIONS=10|25|50|100`
`DEFAULT_ROLLUP_KEY`	Though the application allows users to specify the aggregated record rollup key, you can set a default one as well. For example, to have **p_id** as the default rollup key for your records, you can add this line to the `evaluator.properties` file: `DEFAULT_ROLLUP_KEY=p_id`

`STORED_STRATEGY_<N>`	You can include an arbitrary number of pre-defined search strategies in the evaluator. By default, the application ships with sample Retail and Document strategies, which can be modified, deleted, or extended. Each stored strategy must be distinct in both module order and module options enabled. Each strategy must be identified by `STORED_STRATEGY_<N>`, ending with its numeric identifier, which must start at 1 and increment by 1 for each additional strategy.
	
	For example, to include two additional strategies, you can add the following pipe-delimited lines to the `evaluator.properties` file similar to this:
	
	`STORED_STRATEGY_1=stem|thesaurus|stratify(collec¬ tion()/record[P_Score>90],*,collec¬ tion()/record[P_Score<50])|nterms(considerField¬ Ranks)|maxfield|glom|phrase(subphrase,query_expan¬ sion,considerFieldRanks)|static(P_Price,descending)`
	
	`STORED_STRATEGY_2=nterms(considerFieldRanks)|max¬ field|glom|exact(considerFieldRanks)`

`STORED_STRATE¬ GY_<N>_NAME`	If you use stored strategies as described in the previous section, include these properties to label them. Each strategy defined as a

Property	Description
	STORED_STRATEGY must have a corresponding name defined wtih this property. To specify names for the STORED_STRATEGY_1 and STORED_STRATEGY_2 examples, include the following properties in the evaluator.properties file: STORED_STRATEGY_1_NAME=Mike's Stratify Strategy STORED_STRATEGY_2_NAME=Document Strategy
ORDER_ATTRIBUTES	Sets the display order for the properties to be the value set in DISPLAYED_ATTRIBUTES. By default, the application shows properties in alphabetical order. For example: ORDER_ATTRIBUTES=true
IMAGE_DISPLAY_TEMPLATE	Use with IMAGE_PROPERTY_NAME. This option lets you specify a URL format by which to retrieve images to display for the records. For example, if the default image path is http://www.acme.com/image/products/123456$thumbnail$, where 123456 is the image ID as in the value of IMAGE_PROPERTY_NAME, then add the following line to the evaluator.properties file: IMAGE_DISPLAY_TEMPLATE = http://www.acme.com/image/products/###IMAGE###?$thumbnail$
IMAGE_PROPERTY_NAME	Use with IMAGE_DISPLAY_TEMPLATE. This option lets you specify a URL format by which to retrieve images to display for the records. For example, if the default image path is http://www.acme.com/image/products/123456$thumbnail$, where http://www.acme.com/image/products/###IMAGE###?$thumbnail$ is the value of IMAGE_DISPLAY_TEMPLATE, then add the following line to the evaluator.properties file: IMAGE_PROPERTY_NAME = P_Product_Image

4. Save and close the file.
5. Restart the Endeca Tools Service.

After you have finished configuring properties for the Relevance Ranking Evaluator, business users can test and experiment with relevance ranking strategies. See the *Workbench User's Guide* for more information.

Chapter 12

Troubleshooting Application Components

This section covers steps for troubleshooting possible errors or performance issues in an application.

Troubleshooting publishing content from the Endeca Configuration Repository to the MDEX Engine

Information in the Endeca Configuration Repository is published to the MDEX Engine after making any changes in the Experience Manager or Thesaurus. In the event of errors during the publishing process, you should consult the available log files.

The log files are located at `%ENDECA_TOOLS_CONF%\logs\sling.<yyyy-mm-dd>.log` on Windows, or `$ENDECA_TOOLS_CONF/logs/sling.<yyyy-mm-dd>.log` on UNIX.

Once you have located and fixed the cause of the initial publishing failure, you can re-publish by navigating to the `control` directory of your application and running the following command:

```
runcommand.bat IFCR push<Authoring|Live>ContentToDgraphById <Dgraph ID>
```

For example, to push content to the authoring Dgraph in the Discover Electronics reference application, you would use the following command:

```
runcommand.bat IFCR pushAuthoringContentToDgraphById AuthoringDgraph
```

The selected Dgraph must belong to the Authoring or Live content group. In the Discover Electronics reference application, these settings are configured in the `<app dir>\config\script\AuthoringDgraphCluster.xml` and `LiveDgraphCluster.xml` files.

Note: This command applies to individual Dgraphs, not to Dgraph clusters or groups.

Analyzing Deployment Template script errors

When errors occur during the execution of a Deployment Template script, consult the log files of the Endeca Application Controller (EAC) or Workbench for information.

These messages can help you analyze the cause of the errors by revealing the server state, operations performed, and exceptions encountered by Workbench or EAC Note that Deployment Template scripts rely primarily on EAC and Workbench Web services, invoking their operations in sequence to accomplish the overall task.

Releasing locks set by the Deployment Template in the EAC

In some instances, you may find it necessary to manually release locks that the Deployment Template scripts have put in the EAC on a particular component.

Different types of locks can appear in your Guided Search implementation. The first type of lock is a Windows file system lock. For example, if you have Windows Explorer open at the `data\forge_output` location and the baseline update tries to clean up that folder, it will fail due to `explore.exe` holding on to the directory lock. To release file system locks, make sure that no processes and users have related folders (or files within those folders) open. (UNIX does not put exclusive system locks on files.)

Other types of locks that you may encounter are locks (or EAC flags) that are put into the EAC by the Deployment Template `baseline_update` script, `partial_update` scripts, or other scripts. The default lock is called `update_lock`. It is created by the following line in the Deployment Template script:

```
LockManager.acquireLock("update_lock")
```

If the running Deployment Template script breaks halfway through its execution due to an unhandled exception, or is manually interrupted by a user pressing `Ctrl-C` while it is running, the lock remains set within the EAC.

For example, you may see the following exception error in the Deployment Template logs:

```
[10.17.09 06:52:09] SEVERE: Caught an exception while
 invoking method 'run' on object 'BaselineUpdate'.
Releasing locks.
Caused by java.lang.reflect.InvocationTargetException
...
[10.17.09 06:52:09] INFO: Released lock 'update_lock'.
```

While there can be other causes for this exception, it typically results from a failure to release locks on the Dgraph.

To release the lock on a component within the EAC, run the following commands:

1. Using the `eaccmd` tool, run the following command to obtain a list of all outstanding flags in the application. You may want to review these flags before running the `remove-all-flags` command.
   `list-flags --app application_name`
2. Run the following commands from the command line, or using the `eaccmd` tool:

Option	Description
From the command line:	Go to the `.<AppDir>/control/` directory. Run the following Deployment Template command: On Windows: `.\runcommand.bat LockManager releaseLock update_lock` On UNIX: `./runcommand.sh LockManager releaseLock update_lock`
Using the `eaccmd` tool:	Windows: `eaccmd.bat remove-all-flags -app <your application>` UNIX: `eaccmd.sh remove-all-flags -app <your application>` The LockManager is internally using EAC flagging functionality, so the `remove-all-flags` function of `eaccmd` effectively cleans the Deployment Template locks.

This releases the locks in the EAC.

Avoiding defunct EAC processes on UNIX

On UNIX systems, the `ps` command may report a number of defunct EAC-originated processes. This is known and expected EAC behavior and it does not necessarily indicate a problem.

For example, you might see the following output from the `ps` command:

```
> ps -ef | grep endeca
endeca 1924 1875 0 - ? 2:00 <defunct>
[...]
```

Additionally, warning messages of this form appear in the `$ENDECA_CONF/logs/process.0.log` file on the affected server:

```
Apr 17, 2009 11:24:17 AM
com.endeca.esf.delegate.procctrl.ExecutableProcessHandle
tryCleanShutdown
WARNING: Process 1924 did not shutdown cleanly after 30 seconds.
Terminating forcefully.
```

The cause of these warning messages is as follows. When the EAC shuts down a child process like a Dgraph, it initially sends the correct exit command for the process (`admin?op=exit` in the case of the Dgraph) and waits 30 seconds for the process to exit. However, if the Dgraph is processing a long-running query, or if its request queue is long, it may not be able to shut down within 30 seconds.

If the process does not exit after 30 seconds, the EAC logs the warning message shown above and then kills the process with the operating system's `kill` command. When this occurs, the affected process is reported by `ps` as being in a `<defunct>` state. In this state, it does not use memory, disk space, or ports and should not be a problem for the system.

Alternatively, this can happen if you kill the EAC process directly rather than by using the `shutdown.sh` script. In this case the EAC process terminates immediately, leaving any chlid processes in a `<defunct>` state.

To avoid defunct EAC processing, consider the following recommendations:

- The request log shows whether queuing or long processing times are preventing the Dgraph from responding in time to the `admin?op=exit` command. If this is the case, spreading traffic over a larger number of MDEX Engine mirrors (for queuing) or reducing query complexity (for long processing times) should allow the Dgraph to respond more quickly to the exit command.
- Another option may be to override the default 30-second timeout period for EAC shutdowns by modifying the value of the `com.endeca.eac.process.shutdownTimeoutSecs` setting in your server's `$ENDECA_CONF/conf/eac.properties` file.

  This value shows the length of time in seconds that the EAC Agent on that server waits for a process to exit. Specifying a higher value for this setting may help prevent creation of `<defunct>` EAC child processes, but may also make EAC updates slower, because the EAC Agent will wait longer for all processes to exit.

  Note: Modifications to this setting will not take effect until the Endeca HTTP service (which contains the EAC) is restarted on the server.

Part 4

Backing Up, Restoring, and Exporting Applications

Chapter 13

Backing Up an Assembler Application

This section provides a list of the directories and files that you should back up if planning to migrate or significantly modify an application. The directory structure described assumes an environment deployed with the Deployment Template.

About backing up and restoring a Guided Search application

The backup process allows you to take a snapshot of your application, including Experience Manager content and search configuration, user information, and permission data.

This process does not include the provisioning information for an application.

For backup purposes, a Guided Search application is composed of three pieces:

- Application configuration — the configuration stored in the Endeca Configuration Repository for a particular application, including Experience Manager content.
- Application state — the `%ENDECA_TOOLS_CONF%\state\` directory contains information about your application state, including user and permission settings, preview application settings, content XML, and resource metadata.
- Workbench configuration files — XML and properties files that customize the behavior of an Oracle Workbench installation.

The application state and configuration constitute an application backup. Backing up the Workbench configuration ensures consistent handling of Workbench users when migrating or restoring an application.

Backing up an application from the Workbench

You back up application configuration in the Workbench using the `export_site` script provided with the Deployment Template.

The script exports application configuration in a format that can be re-imported to the Workbench. The script connects to the Workbench instance for the current application based on the configuration in `AppConfig.xml`.

Important: To guarantee consistent data, ensure that no baseline or partial updates are running during the backup process.

To back up application configuration in the Workbench:

1. Navigate to the `control` directory of your deployed application, for example, `C:\Endeca\apps\discover\control`.
2. Run the `export_site` script, passing in an optional name for the export file, as in the following examples:

   On Windows:

   ```
   export_site.bat ..\ECR-backups\20121221.xml
   ```

   On UNIX:

   ```
   ./export_site.sh ../ECR-backups/20111221.xml
   ```

   If no file name is provided, it defaults to a file named according to the pattern `<site-name>-DD-MM-YYYY.xml` in the working directory.

Backing up the application state

The `webstudiostore` and `emanager` directories contain information about your application state, including user and permission settings, preview application settings, content XML, and resource metadata.

To back up the application state directories:

1. Stop the Endeca Tools Service.
2. Copy the `webstudiostore` directory and its subdirectories from `%ENDECA_TOOLS_CONF%\state\` (on Windows) or `$ENDECA_TOOLS_CONF/state/` (on UNIX) to another location.

   This directory contains information such as users and permissions, as well as preview application settings.
3. Copy the `emanager` directory and its subdirectories from `%ENDECA_TOOLS_CONF%\state\` (on Windows) or `$ENDECA_TOOLS_CONF/state/` (on UNIX) to another location.

   This directory contains resource metadata, state information, and content XML.
4. Start the Endeca Tools Service.

Backing up the Workbench configuration files

Workbench uses several configuration files located in `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$EN¬DECA_TOOLS_CONF/conf` (on UNIX) to customize the behavior of various aspects of Workbench.

These files store Workbench configuration, user authentication configuration, and definitions of the menus and extensions in Workbench. If you have manually modified any of the following files from their default state, you should copy them to a backup location:

File name	Description
`Login.conf`	Configuration for user authentication using LDAP
`webstudio.properties`	Miscellaneous configuration parameters for Workbench
`webstudio.log4j.properties`	Configuration for the Workbench system log and audit log

File name	Description
ws-extensions.xml	Definitions of Workbench extensions
ws-mainMenu.xml	Definitions of the Workbench navigation menu and launch page

Backing up CAS configurations

Backing up CAS configurations requires a different approach than you would perform for copying and archiving files.

The components you should back up for your CAS environment include the following:

- Crawl configurations
- Crawl data
- Record store configurations
- Record store data
- Custom web crawlers

For information about how to extract your CAS configurations for backup, refer to the *CAS Developer's Guide*.

Assembler application file reference

The relevant Assembler application files to back up are described below.

You should back up the following files. The directory structure is based on an environment managed by the Deployment Template.

Note: *app_dir* indicates the path to the deployed application on disk, for example, `C:\Endeca\apps\MyApp`.

Component	Directory	Contents	Comments
Endeca application	*app_dir*\config	Configuration and pipeline files	
	app_dir\control	Deployment Template scripts	
	app_dir\logs	Logging and usage reports	
	app_dir\data\dgidx_output	Initial indexed data	
	app_dir\data\partials\cumulative_partials	Partial update files	You must apply these files to the initial index.
	app_dir\data\state	State files	These files contain the dimension value IDs for the application. In some

			applications, it is important to retain IDs from one update to the next.
Platform Services	%ENDECA_CONF%\conf	Server context files and configuration files	
	%ENDECA_CONF%\etc		
	%ENDECA_CONF%\reports		You should back up this directory if a custom report directory is specified in the webstudio-report-dir parameter of the AppConfig.xml file.
Workbench	%ENDECA_TOOLS_CONF%\conf	Workbench extensions, menu configuration, and user definitions	
	%ENDECA_TOOLS_CONF%\state	Application state information	

Note: For more information about backing up indexes, see the *Oracle Commerce MDEX Engine Partial Updates Guide* and the *Oracle Commerce MDEX Engine Migration Guide*.

Non-essential files

The following files are not required for restoring an application:

File Type	Files	Comments
Archives	Time-stamped archive files such as forge_output, dgidx_output, and logs are optional.	
Data files	<app dir>\test_data <app Dir>\data\complete_data_config <app dir>\data\forge_output <app dir>\data\incoming <app dir>\data\partials <app dir>\data\processing <app dir>\data\temp <app dir>\data\workbench	You should back up data\partials\cumulative_partials, as noted above.
Dgraph instances	<app dir>\data\dgraphs	

Chapter 14

Restoring an Assembler Application

This section describes how to restore application data and configuration from backup files.

Restoring a backup of a site to the Endeca Configuration Repository

You can restore a backup of your application configuration to an instance of the Endeca Configuration Repository using the `import_site` script provided with the Deployment Template.

The script takes a file created by a previous export and imports its content to the Endeca Configuration Repository. The script connects to the Endeca Configuration Repository instance for the current application based on the configuration in `AppConfig.xml`.

Important: To guarantee consistent data, ensure that no baseline or partial updates are running during the backup process.

Restoring backups of the ECR is supported only between instances of the same release version.

To restore a backup of your application configuration in the Endeca Configuration Repository:

1. Navigate to the `control` directory of your deployed application, for example, `C:\Endeca\apps\discover\control`.
2. Run the `import_site` script, passing in the file name of the backup, as in the following examples:

   On Windows:

   ```
   import_site.bat discover-21-12-2012.xml
   ```

   On UNIX:

   ```
   ./import_site.sh discover-21-12-2012.xml
   ```

3. Run the `load_baseline_test_data` script.
4. Run the `baseline_update` script to publish updated information to the MDEX Engine.
5. Run the `set_templates` script to push updated templates to the repository.
6. Run the `promote_content` script.

Restoring a backup of the application state

You can restore backups of the application state directories, `webstudiostore` and `emanager`, to an installation of the same version or later.

To restore backups of the application state directories:

1. Stop the Endeca Tools Service.
2. Delete the `webstudiostore` directory from `%ENDECA_TOOLS_CONF%\state\` (on Windows) or `$ENDECA_TOOLS_CONF/state/` (on UNIX).
3. Copy the backup of the `webstudiostore` directory, including all its subdirectories, to `%ENDECA_TOOLS_CONF%\state` (on Windows) or `$ENDECA_TOOLS_CONF/state/` (on UNIX) to another location.
4. Delete the `emanager` directory from `%ENDECA_TOOLS_CONF%\state\` (on Windows) or `$ENDECA_TOOLS_CONF/state/` (on UNIX).
5. Copy the backup of the `emanager` directory and its subdirectories to `%ENDECA_TOOLS_CONF%\state\` (on Windows) or `$ENDECA_TOOLS_CONF/state/` (on UNIX).
6. Start the Endeca Tools Service.

Restoring a backup of Workbench configuration files

You should restore only backups of configuration files to the same exact version of Workbench -- for example, from version 3.1.0 to version 3.1.0, but not from 3.1.0 to any other 3.1.x version.

Upgrading your installation may introduce configuration changes that require you manually to merge your configuration files. For more details about changes to the Oracle Commerce Workbench configuration files, see the *MDEX Engine Migration Guide*.

To restore a backup of the Workbench configuration files:

1. Stop the Endeca Tools Service.
2. Copy the backup files to `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF/conf` (on UNIX).
3. Start the Endeca Tools Service.

Chapter 15

Exporting and Importing Workbench Content

This section summarizes the steps that you must follow to export and import Workbench content.

Steps for Exporting and Importing Workbench Content

This section summarizes the steps that you take to export and import Workbench content.

You can export and import Workbench content for either of two purposes:

- To examine, and optionally modify, your exported application content before you re-import it. For this purpose, you export your Workbench content in the public format described in *Public JSON Formats for Exported Configuration* on page 229.
- Backup and restore format. Use this format if you need only to back up your application without examining or modifying it before you restore it. For complete backup and restore, use the `export_site` and `im¬port_site` scripts. Oracle recommends that you not edit data that you export using `export_site`. For information about these scripts, refer to the *Oracle Commerce Deployment Template Usage Guide*. For more information about these scripts, see *Backing up an application from the Workbench* on page 215.

Types of Workbench Content That Can be Exported and Imported In Public Format

The following types (ecr:type) of Workbench content can be exported and imported in public format:

- content-root
- content-item
- content-collection-folder
- page-root, site-home, and page
- packaged services for Guided Search
- phrase-root and phrases
- redirects and redirect-group
- template-root and template
- thesaurus
- user-segments

For information about how to export these content types, see *Exporting Workbench Content* on page 225. For information about how to import these content types, see *Importing Workbench Content* on page 226.

Types of Workbench Content that Are Exported and Imported in Other (non-Public) Formats

The following types of Workbench content cannot be exported, imported, or stored in public format:

- preview settings. To export default preview settings to the exportedPreviewSettings directory, use the following command: `runcommand.bat IFCR legacyExportContent "configuration/tools/preview" exportedPreviewSettings`. To import default preview settings from the `exportedPreviewSettings` directory, use the following command: `runcommand.bat IFCR legacyUpdateContent "configuration/tools/preview" exportedPreviewSettings`
- CAS configuration such as attributes and precedence rules. For more information about how to export and import CAS configuration, refer to the *Oracle Commerce Content Acquisition System Developer's Guide*.
- media. For importing and exporting only media, use the `set_media` and `get_media` commands, respectively. For information about these scripts, see the *Oracle Commerce Deployment Template Usage Guide*.
- editor configuration. For importing and exporting only editor configuration, use the `set_editors_config` and `get_editors_config` scripts, respectively. For information about these scripts, see the *Oracle Commerce Tools and Frameworks Installation Guide*.

Steps for Exporting and Importing a Complete Set of Workbench Content

To export a complete set of Workbench content, follow these steps:

1. To export all the content that can be exported in public format, run the following command:

```
runcommand[.bat/.sh] IFCR exportApplication export_folder1
```

2. To export media, editor configuration, and preview settings, run the following command:

```
runcommand.[.bat/.sh] IFCR legacyExportContent "/" export_folder2
```

3. To export CAS configuration, run the following command:

```
index_config_cmd[.bat/.sh] get-config [-o OwnerName] -f FileName
```

Note: For detailed information about the `get-config` task, refer to the *Content Acquisition System Developer's Guide*.

4. To export all users and their tool permissions, run the following command:

```
export_users.[.bat/.sh]  --config config-prop-file --output output-file
  --admin-user admin-user --admin-password admin-password
```

where:

config-file is the location of the configuration property file

output-file is the name of the output file. If file name is not specified, the output will be *users-timestamp*.json. (Optional)

admin-user is the admin username in the destination Workbench (deprecated)

admin-password is the admin password in the destination Workbench (deprecated)

To import a complete set of Workbench content, run the following commands in the order shown:

1. To import users and their tool permissions, run the following command:

```
import_users.[.bat/.sh] --input input-file --config config-file --default_user_pswd  user-password
```

```
  -- single-app permissions --admin-user admin-user --admin-password admin-
password
```

where:

input-file is the location of data file

config-file is the location of the configuration property file

user-password is the default password for imported users (optional)

permissions is the set of import permissions for a single application (optional)

admin-user is the admin username in the destination Workbench (deprecated)

admin-password is the admin password in the destination Workbench (deprecated)

2. To import all content that has been exported in public format, run the following command:

```
runcommand.[.bat/.sh] IFCR importApplication export_folder1
```

3. To import CAS configuration, run this command:

```
index_config_cmd[.bat/.sh] set-config [-o OwnerName] -f FileName
```

Note: For detailed information about the `set-config` task, refer to the *Content Acquisition System Developer's Guide*.

4. To import the rest of the Workbench content, run this command:

```
runcommand.[.bat/.sh] IFCR legacyUpdateContent "/" export_folder2
```

5. The following example imports thesaurus content from a zip file named `thesaurus.zip`:

```
runcommand.sh IFCR importContent thesaurus /localdisk/foo/content/thesaurus/the¬
saurus.zip
```

Chapter 16

Exporting Workbench Content in Public Formats

You can export Workbench content from the configuration repository (ECR) to files, and import it from the files into your ECR. The Workbench content is exported and imported in common public formats.

Overview of Exporting and Importing Workbench Content

You can import and export Workbench content for any of the following purposes:

- Copying Workbench content from one environment to another environment
- Versioning of your application's Workbench content
- Promoting content from authoring to live environments. For detailed information about how to do this, see *Configuring the promotion method for your application* on page 40.

Workbench content is exported in JSON files. The JSON file is exported to a folder whose name is identical to that of the Workbench content; for example, the JSON file for pages is exported to a folder named pages.

Some types of Workbench content have specific data relating to individual components associated with them; this additional data is exported as xml files to the same folder as the JSON files. Workbench content for thesaurus entries, keyword redirects, and user segments is exported only as JSON, because it has no additional content associated with it.

You can use the `runcommand` script in the `\control` directory of your application to run the commands that import and export Workbench content.

Exporting Workbench Content

This section describes commands that you can use to export most types of Workbench content.

Exporting Workbench content

The exportApplication command exports most Workbench content associated with your application to a target directory either in a .zip file or as unzipped files. The exportApplication command overwrites existing content; it does not update it.

The syntax of the exportApplication command is as follows:

```
runcommand.sh IFCR exportApplication target isDirectoryFormat
```

where:

- *target* is the directory into which the Workbench content is exported.

- *isDirectoryFormat* is either false (export as a .zip file) or true (export as unzipped files). The default is false.

Exporting Specified Portions of Workbench Content

The exportContent command can export specified portions of Workbench content to a specified folder either in a .zip file or as unzipped files:

```
runcommand.sh IFCR exportContent relative_path target isDirectoryFormat
```

where:

- *relative_path* is the path of the Workbench content that is to be exported. The path is relative to the location of your application in the Workbench.
- *target* is the directory to which the Workbench content is exported.
- *isDirectoryFormat* is either false (export as a .zip file) or true (export as unzipped files). The default is false. A .zip file will have the same name as the Workbench content located at the *relative_path* that you specify. In the exported data, the root document will be the .zip file name.

Note: If the .zip file or unzipped files already exist, the export operation overwrites them.

For example, the following command exports thesaurus content to the `/import` folder in unzipped files:

```
runcommand.sh IFCR exportContent thesaurus /localdisk/apps/Discover/config/im¬
port/thesaurus true
```

The following command, however, exports the thesaurus content to the `/import` folder in a .zip file named "thesaurus.zip":

```
runcommand.sh IFCR exportContent thesaurus /localdisk/apps/Discover/config/import
```

Importing Workbench Content

You can import Workbench content from specified folders. The imported Workbench content overwrites any existing Workbench content.

Note: The credentials key configured for the IFCR component in the WorkbenchConfig.xml is used to determine which user is running the import process. If this user does not have, or loses, administrator rights on Workbench, any import that this user attempts to run will fail with the error AccessDeniedException. For example, if you create a new user, "smith", with admininstrator rights on Workbench, and also change the user in the credentials store to "smith", the last modified by fields of any imported types of content are subsequently set to "smith". If "smith" attempts to run an import after losing administrator rights on Workbench, the import fails with the error AccessDeniedException. You can override this behavior by specifying the `ecr:lastModifiedBy` property in the JSON file of the content-item.

Importing Workbench Content from a Specified Directory

The following command imports Workbench content from a specified folder:

```
runcommand.sh IFCR importContent  relative_path  source
```

where:

- *relative_path* specifies the location within Workbench where the content is to be imported. The path is relative to the location of your application in the Workbench.
- *source* is the pathname of the folder from which the Workbench content is imported. You can specify either an unzipped folder or a specific zip file.

For example, the following command imports thesaurus content from a folder named thesaurus:

```
 runcommand.sh IFCR importContent thesaurus /localdisk/apps/Discover/config/im¬
port/thesaurus
```

The following example imports configuration for folder2 from a directory called folder2:

```
 runcommand.sh IFCR importContent content/folder1/folder2 /localdisk/foo/con¬
tent/folder1/folder2
```

Note: The importContent command deletes the Workbench content at the `relative_path` before importing the contents of source directory.

Recommended relative paths for importing public format content using importContent command

The following table lists the relative paths where public format content imported by the importContent command is best stored:

Type of content / ecr:type	Recommended relative path
site	/
user-segments	userSegments
thesaurus	thesaurus
templates-root	templates
template	templates/SampleTemplate where SampleTemplate is the template id
redirects	redirects
redirect-group	redirects/Default
phrases-root	phrases
phrase	phrases/UniqueId where UniqueId is the unique identifier of the phrase
page-root	pages
site-home	pages/SampleSiteId where SampleSiteId is the identifier of the site-home
page	pages/SampleSiteId/SamplePage1 where SamplePage1 is the name or URL of the page. Note that a page can be nested inside another page. For example, the following relative path is also valid: pages/SampleSiteId/SamplePage1/SamplePage2
content-root	content
content-collection-folder	content/SampleFolder1 where SampleFolder1 is the name of the content-collection-folder. Note that content-collection-folder can be nested. For example, the following relative path is also valid: content/SampleFolder1/SampleFolder2
content-item	content/SampleFolder1/SampleContentItem where SampleContentItem is the name of the content-item.

Note: Use "/" (the forward leaning slash) to construct a relative path. For example, the following is a valid relative path: `content/SampleFolder1`. But the following is not a valid relative path: `content\Sam¬pleFolder1`.

Importing an Application Configuration

The following command overwrites most Workbench content with the content stored in the folder specified as *source*:

```
runcommand.sh IFCR importApplication source
```

importApplication imports the following types of Workbench content:

- content-root
- content-item
- content-collection-folder
- page-root, site-home, and page
- packaged services for Guided Search
- phrase-root and phrases
- redirects and redirect-group
- template-root and template
- thesaurus
- user-segments

If a folder for a particular type of content is not present in the specified source folder or zip, that type of content is deleted from the Workbench upon import. Thus, if the phrases folder is not present in the specified source folder or zip, phrases will be deleted from the Workbench and the phrasing feature will not work.

Exporting and Importing Permissions

Permissions can be associated explicitly with any type of Experience Manager content. Thus, permission assignments can be associated with any of the following types of content:

- content-root
- content-collection-folder
- content-item
- page-root
- site-home
- page

The exported JSON files for any of these types of Workbench content will include any permissions that have been explicitly assigned for these types.

For example, the following line from a JSON file assigns all permissions to "userone":

```
"ecr:permissions" : {"userone" : "all"}
```

Folders for Exported Content

The following table lists and describes the types of exportable Workbench content and the names of the folders into which each type is exported.

Exported .zip file or unzipped files	Folder	Corresponding ecr:type	Subfolders

application_name.zip	The zip file is exported to the folder specified in the export command.	site	
content	content	content-root	All subfolders of type content-collection-folder or content-items, such as Mobile, Shared, and Web.
pages	pages	page-root	All site-home content, such as DiscoverElectronics.
phrases	phrases	phrases-root	
redirects	redirects	redirects	Default
services	services	page-root	dimensionsearch, Discoverelectronics, guidedsearch, recorddetails
thesaurus	thesaurus	thesaurus	Contains a JSON file of thesaurus entries.
templates	templates	templates-root	All application templates, including AutoSuggestPanel, breadcrumbs, and so on.
userSegments	userSegments	user-segments	Contains a JSON file of user segments.

Note: You can export and import the contents of site-home, page, content-item, service and template only in their entirety. You cannot export or import individual files for a content type. For example, to overwrite the `filterState.xml` file of a site-home, you must import the folder containing both the JSON file and the `filterState.xml` file of that site-home.

Public JSON Formats for Exported Configuration

This section describes the public JSON formats of exportable application content.

Public format for template-root:

Properties in the JSON file:

Public format for template-root:

"ecr:type": Must be set to "templates-root"

Example:

```
{"ecr:type": "templates-root"}
```

Public format for template:

Properties in the JSON file:

"ecr:type": Must be set to "template"

A template must have a template.xml file associated with it. The template.xml describes structure of a content-item that can be created in Experience Manager using this template.

Example:

```
{"ecr:type": "template"}
```

Public format for phrases-root:

Properties in the JSON file:

"ecr:type": Must be set to "phrases-root"

Example:

```
{"ecr:type": "phrases-root"}
```

Public format for phrase:

Properties in the JSON file:

"ecr:type": Must be set to "phrase"

"phrase": The search term for which phrase search should be performed.

Example:

```
{
   "phrase": "hd camera",
   "ecr:type": "phrase"
}
```

Public format for page-root :

Properties in the JSON file:

"ecr:type": Must be set to "page-root".

Example:

```
{ "ecr:type": "page-root" }
```

Public Format for site-home

The JSON file for a site home can include the following properties:

Attribute	Type	Required	Description
"ecr:type"	String	Yes	Must be "site-home".
"displayName"	String	Yes	The site "displayName" is used as the display name in the Workbench wherever sites are referenced.
"description"	String	No	Information about the nature and content of the site home.
"urlPattern"	String	No	The patterns used to match the request URL to this site definition, based on the scheme used by the application to encode site into the URL. When attempting to identify a site from a request, this patterns will be matched against the URL. For example, "/discoverSony" can be specified for the DiscoverSony site definition. Multiple patterns can be specified, delimited by commas.
"previewUrl"	String	No	The URL used to preview the specific site definition.
"linkServiceUrl"	String	No	A host specific for this site or a domain or URL specific for this site.

The following example illustrates the exported JSON format for defining a site home whose display name is "Discover Sony" and whose URL pattern is "/sony":

```
{
  "ecr:type":"site-home",
  "displayName":"Discover Sony",
```

```
  "urlPattern":"/sony"
}
```

Public format for page:

Properties in the JSON file:

"ecr:type": Must be set to "page".

A page usually has a content.xml associated with it. This content.xml configures the ContentItem that will be returned by Assembler for this page.

Example:

```
{
   "contentType": "Page",
   "ecr:type": "page"
}
```

Public format for content-root

Properties in the JSON file:

"ecr:type": Must be set to "content-root"

Example:

```
{ "ecr:type": "content-root" }
```

Public format for content-collection-folder

Properties in the JSON file:

"ecr:type": Must be set to "content-collection-folder"

"contentType": Optional. Restricts the type of content-item which can be added to this folder by an Experience Manager user.

Example:

```
{
  "contentType": "MainContent",
  "ecr:type": "content-collection-folder"
}
```

Public format for content-item

Properties in the JSON file:

Property	Value	Description
ecr:type	content-item	Required.
workflowState	ACTIVE or INACTIVE	Required. Describes whether this content-item can be triggered for any request.
priority	Positive integer.	Required. Denotes relative priority in which this content-item can be triggered.
previewable	true (default) or false	Optional. Denotes whether this content-item can be triggered in preview mode.
startTime	yyyy-MM-dd'T'HH:mm	Optional. Specifies the time from when this content-item can be triggered.

endTime	yyyy-MM-dd'T'HH:mm	Optional. Restricts the time until when this content-item can be triggered. If "endTime" is specified, "startTime" must be specified as well.
workflowNote	string	Optional. Contains comments about this content-item. Empty by default.
userSegments	array of strings	Optional. Restricts this content-item to trigger only when a specified user-segment is present in the request.
ecr:lastModifiedBy	string	Optional. User id of the user who last modified this content-item. Defaults to the user who is importing this content-item.
ecr:lastModified	yyyy-MM-dd'T'HH:mm:ss.SSSX	Optional. The last time this content-item was modified. Defaults to the time at which this content-item was imported into Workbench.

triggers	See description.	Required. Specifies the search term/ dimension value under which this content-item is triggered. Specify an array of one or more of the following types of trigger:
		• search term based
		• dimension value based
		• search term and dimension value based
		Specify an empty array if you are not using triggers.
		Search term triggers
		A search term triggers a content-item when a user's query includes the search term.
		Properties in the JSON file:
		• "searchTerms": Required. The search term for which the content-item should be triggered. Must be a string.
		• "matchmode": Required. Specifies how a user's search term is matched to the configured search-term. Valid values are "MATCHPHRASE", "MATCHEXACT", and "MATCHALL". For information about these values, see the *Oracle Commerce Workbench User's Guide*.
		• "exactLocation": Required. Must be a boolean (true or false). When "exactLocation" is set to true, the trigger is fired only when the search term matches the "searchTerms" value in the way specified by the "matchmode" value, and no dimension values are included in user's navigation state. When "exactLocation" is set to false, the trigger is fired when the search term is matched in the way specified by the "matchmode" value, regardless of the dimension values included in user's navigation state.
		Example:
		`{`
		`"exactLocation": false,`
		`"searchTerms": "canon",`

```
   "matchmode": "MATCHPHRASE"
}
```

Dimension value trigger

A collection of one or more dimension values can trigger a content-item, if user's navigation state contains those dimension values.

Properties in the JSON file:

- "dvalIDs": Required. An array of dimension value id strings.
- "exactLocation": Required. Must be a boolean. When set to true, this trigger is fired when the user's navigation state contains all the specified dimension values and no other dimension value. When set to false, this trigger is fired when all of the specified dimension values are included in the user's navigation state, but can also be fired when one or more other dimension values are in the user's navigation state.

Example:

```
{
    "exactLocation": true,
    "dvalIDs": ["4294967266"]
}
```

Search term and dimension value based trigger

A search term can trigger a content-item if user's query includes those terms and user's navigation state contains specified dimension values.

Properties in the JSON file:

- "searchTerms": Required. The search term for which the content-item should be triggered. Must be a string.
- "matchmode": Required. Configures how a user's search term is matched to the configured search-term. Valid values are "MATCHPHRASE", "MATCHEXACT", "MATCHALL"
- "dvalIDs": Required. An array of dimension value id strings.
- "exactLocation": Required. Must be a boolean (true or false). When "exactLocation" is set to true, the trigger is fired only when the user's navigation state contains all the specified dimension values and no other dimension value. When "exactLocation" is set to false, the trigger is fired when the search term is matched in the way specified by the "matchmode" value and all the specified dimension values are included in user's navigation state, regardless of the other dimension values included in user's navigation state.

		Example: `{ "exactLocation": false, "searchTerms": "canon", "matchmode": "MATCHPHRASE", "dvalIDs": ["18827"] }`

Example of a content-item

Example of a content-item JSON:

```
{
    "ecr:type": "content-item",
    "workflowState": "ACTIVE",
    "previewable": true,
    "priority": 20,
    "startTime": "2014-04-23T17:38",
    "endTime": "2014-04-24T17:38",
    "workflowNote": "Reviewed by John B",
    "userSegments": ["retailCustomer"],
    "ecr:lastModifiedBy": "curtis",
    "ecr:lastModified": "2014-04-23T17:39:32.956-04:00",
    "triggers": [
      {
        "exactLocation": false,
        "searchTerms": "canon",
        "matchmode": "MATCHPHRASE"
      },
      {
        "exactLocation": true,
        "dvalIDs": ["4294967266"]
      }
    ]
}
```

Note: A content-item must have a content.xml associated with it. This content.xml configures the content-item that will be returned by Assembler for this content-item.

Public Format for Keyword Redirects

The JSON file for keyword redirects includes the following properties:

Attribute	Value			
ecr:type	redirect-group			
redirects	One or more redirect entries. Each redirects entry has the following attributes:			
	Attribute	Type	Required	Value
	matchmode	string	yes	The type of match between the specified searchTerms value and the search term that the user enters. Must be one of: MATCHEXACT, MATCHPHRASE, or MATCHALL. For information about these values, see the *Oracle Commerce Workbench User's Guide*.

	url	string	yes	The URL to which users are redirected when they enter the search terms specified by searchTerms.
	searchTerms	string	yes	The search term entered by the user that triggers the redirect to the specified url.

The following example illustrates the exported JSON format for two keyword redirects:

```
{ "ecr:type": "redirect-group",
  "redirects": [
    { "matchmode":  "MATCHEXACT",
      "url": "/browse/Canon/_/N-1z141ya",
      "searchTerms": "canon"
    },
    { "matchmode": "MATCHPHRASE",
      "url": "/browse/bags-cases/_/N-25xw",
      "searchTerms": "camera bag"
    },
  ]
}
```

Public Format for Thesaurus Entries

The JSON file for thesaurus entries includes the following properties:

Attribute	**Value**			
ecr:type	thesaurus			
entries	One or more entries. Each entry has the following attributes:			
	Attribute	Type	Required	Value
	type	string	yes	• one-way: Specifies a single synonym for the searchTerms value. The word configured as the searchTerm matches the synonym wherever it occurs in your records. However, a word configured as a synonym, if entered as a search term, will not match the word configured as a search term. • multi-way: Specifies a list of synonyms, any one of which, if entered as the search term, will match any other synonym wherever it occurs in your records.
	searchTerms	string		Required if the type value is one-way. Not used if the entry type is multi-way.
	synonyms	string or string[]	yes	• If type is one-way, a single string that is considered a match for the searchTerm value. • If type is multi-way, a set of two or more strings, any one of which is considered a match for any of the others when entered by the user as a search term.

The following example illustrates the exported JSON format for a one-way thesaurus entry (the search term "digital" matches "digi" in your records), and a multi-way thesaurus entry ("converter", "adapter", and "adapter-converter" all match each other):

```
{
 "ecr:type" : "thesaurus",
 "entries" : [
  {
     "type": "one-way",
     "searchTerms": "digital",
     "synonyms" : [
       "digi"
     ]
   },
   {
     "type": "multi-way",
     "synonyms": [
       "converter","adapter","adapter-converter"
     ]
   }
  ]
}
```

Public Format for User Segments

User segments can be used to limit what particular customers can see. User-segments can be associated with particular users by any logic that is available to your application, and then passed to the Assembler as query parameters. In response to the query, the Assembler returns content that is appropriate to that user-segment.

The JSON file for user segments includes the following properties:

Attribute	**Value**			
ecr:type	user-segments			
segments	One or more segment definitions. Each segment has the following attributes and values:			
	Attribute	Type	Required	Value
	name	string	yes	The name by which the user segment can be referenced.
	description	string	no	Text providing information about the user segment.

The following example illustrates the exported JSON format for defining user-segments; here, two user-segments are defined, named "members" and "customers":

```
{
  "ecr:type": "user-segments",
  "segments": [
    {
      "name": "members",
         "description":"Members only"
    }
    {
      "name": "customers",
         "description":"Returning shoppers"
    }
  ]
}
```

Appendix A

Guided Search Environment Variables and Port Usage

This section lists the environment variables and ports used by the Guided Search software. Depending on which components you have installed, not all of them may apply to your implementation.

Guided Search environment variables

The Guided Search installation programs create several environment variables.

For each variable, the first value listed is the path if you accept the default installation path on Windows (under `C:\Endeca\<product name>`) and use a per-machine installation. The default paths for a per-user installation will be rooted in the `%USERPROFILE%` directory.

The second value is the path within your installation directory on UNIX. For example, if you install Endeca to `/usr/local/`, the full path of `ENDECA_ROOT` would be `/usr/local/endeca/PlatformServices/version` in your environment.

In addition to creating the variables below, the installation may add Endeca directories to the `PATH` variable.

Note: For the MDEX Engine installation, environment and `PATH` variables are set by running the `mdex_setup` scripts provided by the installation. See the *Oracle Commerce MDEX Engine Installation Guide* for more information.

MDEX Engine variables

The following variable is used by the MDEX Engine:

Variable	Description	Default value
`ENDECA_MDEX_ROOT`	Specifies the path of the MDEX Engine root directory.	• `C:\Endeca\MDEX\version` • `endeca/MDEX/version`

Endeca Tools Service

The Endeca Tools Service uses the following environment variables:

Environment Variable	Value	Setting
JAVA_HOME	<ENDECA_TOOLS_ROOT>\server\j2sdk	Sets JAVA_HOME to the included Java 2 SDK.
ENDE¬ CA_TOOLS_ROOT	<Tools and Frameworks directory>	Sets the value to the Tools and Frameworks installation directory.
ENDE¬ CA_TOOLS_CONF	<ENDECA_TOOLS_ROOT>\server\workspace	Sets the value to the Tools and Frameworks server\workspace directory.
CATALINA_BASE	ENDECA_TOOLS_CONF	Sets the location of the Tomcat instance to the Endeca workspace directory.

Note: If you are not running the Tools and Frameworks from the Endeca Tools Service on Windows, you should set the above environment variables manually.

Platform Services variables

The following variables are used by the Platform Services:

Variable	Description	Default value
ENDECA_ROOT	Specifies the path of the Platform Services root directory.	• C:\Endeca\PlatformServices*ver¬ sion* • endeca/PlatformServices/*version*
ENDECA_REFERENCE_DIR	Specifies the path of the directory that contains the Endeca reference implementations, such as the sample wine project and the JSP and .NET UI references.	• C:\Endeca\PlatformServices\ref¬ erence • endeca/PlatformServices/refer¬ ence
ENDECA_CONF	Specifies the path of the workspace directory for the Endeca HTTP service, which contains configuration files, logs, and temporary storage directories.	• C:\Endeca\PlatformSer¬ vices\workspace • endeca/PlatformSer¬ vices/workspace
PERLLIB	Specifies the path of the perl root directory and its directory of libraries.	• %ENDECA_ROOT%\perl and %ENDE¬ CA_ROOT%\perl\5.8.3\lib • $ENDECA_ROOT/lib/perl:$ENDE¬ CA_ROOT/lib/perl/Control:$ENDE¬

Variable	Description	Default value
		CA_ROOT/perl/lib:$ENDE¬ CA_ROOT/perl/lib/site_perl
PERL5LIB	Same as the PERLLIB variable.	Same as the PERLLIB variable.
UnixUtils	Specifies the path of the utili¬ ties directory, which contains Windows versions of some UNIX common utilities.	• %ENDECA_ROOT%\utilities • not available on UNIX

Endeca Tools and Frameworks variables

The following variables are used by the Workbench:

Variable	Description	Default value
ENDECA_TOOLS_ROOT	Specifies the path of the Endeca Tools and Frameworks root directory.	• C:\Endeca\ToolsAndFrameworks*version* • endeca/ToolsAndFrameworks/*version*
ENDECA_TOOLS_CONF	Specifies the path of the workspace directory for the Endeca Tools Service, which contains configuration files, logs, and temporary storage directories.	• C:\Endeca\ToolsAndFrameworks\serv¬ er\workspace • endeca/ToolsAndFrameworks/serv¬ er/workspace

Other variables

Other variables used by Endeca include the following:

Variable	Description	Default value
ENDECA_PROJECT_DIR	Specifies the path of the deployed application. This variable is set and used by the Guided Search Deployment Template.	Value is taken from user input at installation time.
ENDECA_PROJECT_NAME	Specifies the project name that is used, for example, as the JCD job prefix for jobs defined in the project's Job Control Daemon. This variable is set and used by the Guided Search Deployment Template.	Value is taken from user input at installation time.

Guided Search ports

This topic describes the ports used by the Endeca packages and their default port numbers.

You can replace any of the default port numbers with numbers of your own, as long as they do not conflict with an existing port on your machine. Port numbers can be no larger than 32767.

Service ports

Port	Default
Endeca Tools Service port	8006
Endeca Tools Service Promotion port	8007
Endeca Tools Service SSL port	8446
Endeca Tools Service shutdown port	8084
CAS Service port	8500
CAS Service shutdown port	8506
Endeca HTTP Service port	8888
Endeca HTTP Service SSL port	8443
Endeca HTTP Service shutdown port	8090

Deployment Template ports

These are the port numbers suggested by the Deployment Template installation, but you can specify any other port when you deploy your application.

Port	Default
Dgraph1 user query port	15000
Dgraph2 user query port	15001
Endeca Logging and Reporting Server port  **Note:** The Logging Server port number can be no larger than 32767.	15010

Assembler ports

Port	Default
Client port If the client port is set to -1, the system uses an ephemeral port. An ephemeral port is allocated automatically for a short time and is only used for the duration of a communication session. When the session ends, it is available for another request.	-1

Reference implementation ports

These port numbers are used in the configuration files that ship with the reference implementation (`sample_wine_data`).

Port	Default
MDEX Engine user query port	`8000`
Endeca Logging and Reporting Server port **Note:** In the JSP reference implementation, the default Logging server port number is larger by 2 than the corresponding Dgraph port number. For example, for the Dgraph port 15000, the default port for the Logging Server in the reference implementation is 15002. For the Dgraph port 15001, the default port for the Logging Server in the reference implementation is 15003. (This assumes that the Logging Server is running on the same host as the MDEX Engine.)	`8002`

Appendix B

Guided Search Flag Reference

This appendix provides a description of the flags (options) used by the Dgidx and Dgraph programs. For information on Forge flags, see the *Platform Services Forge Guide*.

Dgidx flags

The Dgidx program indexes the tagged Endeca records that were prepared by Forge, and creates the proprietary indices for the Endeca MDEX Engine.

The usage of Dgidx is as follows:

```
dgidx [-qv] [--flags] <data export file> <output db_prefix>
```

where <db_prefix> specifies the path to the directory, and the prefix used for the files in your Guided Search application.

Dgidx supports the following flags:

Flag	Description
-q	Quiet mode.
-v	Verbose mode.
--compoundDimSearch	Enable compound dimension search for the application. Use of this option increases indexing time. However, if this option is not enabled at index time, compound dimension results (multiple-dimension-value results) are not returned by the MDEX Engine.
--cov	Compute and report coverage statistics for dimensions and properties.
--diacritic-folding	Ignore character accents when indexing text. For details about how characters with diacritical marks are mapped to their ASCII equivalents, see the *MDEX Engine Advanced Development Guide*.
--help	Print the help message and exit.
--lang <lang-id>-u-<collation>	Indexes documents as being in the language specified by <lang-id>, and the flag also specifies an optional collation order in the <collation> portion of the argument. If unspecified, the default for <lang-id> is en (US English). For details about using international languages, see the *MDEX Engine Advanced Development Guide*.

Flag	Description
`--nostrictattrs`	Disable strict attribute checking. Allows records to retain property values for properties with no property (or PROP_REF element) defined in the XML configuration file, and in the Properties view of Developer Studio.
`--numbins <num>`	Limit the number of records that Dgidx reads.
`--out <stdout/stderr file>`	Specify file path to which stdout/stderr should be remapped (the default is to use default stdout/stderr for the process).

`--sort <spec>`	Specify a default sort specification for the data set. The format of <*spec*> is (including the quotation marks):
	`"key|dir"`
	where *key* is the name of a property or dimension on which to sort and *dir* is either `asc` for ascending or `desc` for descending (if not specified, the order will be ascending).
	key can also be a geocode property, as in this example:
	`"Location(43,73)|desc"`
	You can specify multiple sort keys in the format:
	`"key_1[|dir_1]||key_2[|dir_2]||...||key_n[|dir_n]"`
	If you specify multiple sort keys, the records are sorted by the first sort key, with ties being resolved by the second sort key, whose ties are resolved by the third sort key, and so on.
	Note that if you are using the Oracle Endeca Application Controller (EAC) to control your environment, you must omit the quotation marks from the `--sort` flag. Instead, use the following syntax:
	`--sort key_1|dir_1||key_2|dir_2||...||key_n|dir_n`

`--spellmode <mode>`	Specify the spelling correction mode for the application. Supported modes are: • default • aspell • espell • aspell_OR_espell • aspell_AND_espell
`--spellnum`	In spelling modes that enable the `espell` module, include non-word terms (numbers, symbols, and so on) in the espell dictionary. By default, such terms are not included.
`--stemming-updates <file>`	Specify an optional XML file of stemming updates to apply to a default stemming dictionary. See the *MDEX Engine Developer's Guide* for XML examples and file name requirements.
`--threads <num>`	Specify the number of sorting threads to use for the multi-threaded portion of the indexing process. The default is `1`. If this flag is not specified, or if 1 is specified for it, Dgidx uses one sorting thread. If the specified value is greater than 1, Dgidx uses the specified number of threads to sort data.

Flag	Description
	Note that Dgidx runs in multithreaded mode by default. In addition to the number of sorting threads that you can control with the `--threads` flag, Dgidx may use additional maintenance threads that run in the background by default, and are not used for sorting data. To improve indexing performance, Oracle recommends increasing the number of sorting threads. In deployments where a dedicated server is used for indexing the Endeca application, allocate as many threads as your server allows to the Dgidx sorting process. For best performance, the number of sorting threads specified should correlate with the number of cores on the server. Since sorting is only part of the indexing process, using *N* sorting threads does not speed up Dgidx by *N* times.
`--version`	Print version information and exit.

Dgraph flags

The Dgraph program starts the MDEX Engine.

You start the MDEX Engine by running a program called Dgraph, which you point at a set of indices prepared by the Dgidx. The Dgraph has a number of options that allow you to adjust the MDEX Engine. For example, you can tweak spelling, caching, and so forth.

The usage of Dgraph is as follows:

```
dgraph [-?] [--flags] <db_prefix>
```

where `<db_prefix>` specifies the path to the directory, and the prefix used for the files in your Guided Search application.

Flag	Description
?	Print the help message and exit.
`-d`	*Deprecated in MDEX Engine 6.4.0.* Start in debug mode.
`-v`	Verbose mode. Print information about each request to `stdout`.
`--ancestor_counts`	Compute counts for root dimension values. When the flag is omitted, the Dgraph only computes refinement counts for selected and available refinements but not root dimension values.
`--back_compat <api-version>`	Enable backwards compatibility, so that the Dgraph can communicate with previous versions of the Presentation API. In addition to the currently supported version of the Presentation API, the following previous versions are supported: 6.3.x, 6.2.x, 6.1.x, and 6.0.x. Therefore, the value for *<api-version>* must be one of the following: • `630` for the 6.3.0 version of the Presentation API. • `620` for the 6.2.2 and 6.2.1 versions of the Presentation API. • `614` for the 6.1.5 and 6.1.4 versions of the Presentation API. • `601` for the 6.1.3 through 6.0.1 versions of the Presentation API.

Flag	Description
`--backlog-timeout <seconds>`	Specify the wait limit (in seconds) for a query that has been read and queued for processing. This is the maximum number of seconds that a query is allowed to spend waiting in the processing queue before the Dgraph responds with a timeout message. The default value is 60 seconds.
`--cmem <MB>`	Specify the maximum memory usage in MB for the MDEX Engine main cache. When `--cmem` is not specified, the default value is `1024` MB (1GB), for Dgraph installations on 64-bit platforms.
`--config <path>`	Specify a configuration file to read on startup. The configuration file should contain arguments of the same format used on the command line (that is, it ignores whitespace, including newlines).
`--disable_fast_aspell`	Disable fast mode for the aspell spelling module. If you disable fast mode, it decreases the performance of the spelling correction, but may allow additional queries to be corrected. When the fast mode is enabled, it can significantly speed up applications that use spelling correction features with the aspell module. The fast mode is used by default.
`--disable_web_services`	*Deprecated in MDEX Engine 6.3.0.* Suppress the automatic loading of XQuery modules at startup. Note that when web services are disables, all features that write to the dgraph will fails. This includes most Workbench features, such as thesaurus entries, business rules, stop words, and automatic processes.
`--dym`	Enable DYM (Did You Mean?) explicit query spelling suggestions for full-text search queries.
`--dym_hthresh <thresh>`	Specify the threshold number of hits at or above which DYM (Did You Mean?) suggestions will not be generated. The default is `20`.
`--dym_nsug <count>`	Specify the maximum number of DYM (Did You Mean?) query suggestions to return for any query. The default is `1`.
`--dym_sthresh <thresh>`	Specify the threshold spelling correction score for words used by the DYM (Did You Mean?) engine. The default is `175`.
`--dynrank_consider_collapsed`	Use this flag to force the MDEX Engine to consider intermediate collapsible dimension values as candidates for dynamic ranking. This flag alters the default behavior of the MDEX Engine when dynamically ranking dimensions with collapsible dimension values. By default (without this flag specified), the MDEX Engine considers only leaf dimension values for dynamic ranking, removing all intermediate dimension hierarchy from consideration. With the default behavior, when a hierarchical dimension's mid-level values (all except the root and leaf values) are configured as collapsible in Developer Studio, and when the dimension is also set to use dynamic refinement ranking, the dimension collapses and displays only leaf values for all navigation queries. The mid-level dimension values are never displayed regardless of the number of leaf values present in the navigation state.

Flag	Description
	If you would like the MDEX Engine to consider intermediate dimension values (that are configured as collapsible) for dynamic ranking, use this flag.
`--esampmin <num>`	Specify the minimum number of records to sample during refinement computation. The default is `0`. Tuning recommendations: • For most applications, larger values reduce performance without improving dynamic refinement ranking quality. • For some applications with extremely large, non-hierarchical dimensions (if they cannot be avoided), larger values can meaningfully improve dynamic refinement ranking quality with minor performance cost.
`--failedupdatedir <dir>`	Specify the directory into which the MDEX Engine should save the failed update files. The default directory that the MDEX Engine uses for storing the failed update files is `<updatedir>/failed_updates/`.
`--help`	Print the help message and exit.
`--lang <lang-id>-u-<collation>`	Indexes partial updates and processes queries as being in the language specified by `<lang-id>`, and the flag also specifies an optional collation order for search results in the `<collation>` portion of the argument. If unspecified, the default value is inherited from the Dgidx `--lang` flag, and if the flag is not specified for either Dgidx or the Dgraph, then the default for `<lang-id>` is `en` (US English). For details about using international languages, see the *MDEX Engine Developer's Guide*.
`--log <path>`	Specify the path for the Dgraph request log file. The default log file is named `dgraph.reqlog`.
`--log_stats <path>`	Specify the path and filename for the EQL (Endeca Query Language) statistics log. By default, this log is turned off; specifying this flag activates logging of statistics for EQL requests.
`--log_stats_thresh <value>`	Set the threshold above which statistics information for an Endeca Query Language request will be logged. The value is specified in milliseconds (1000 milliseconds = 1 second). The value can also be specified in seconds by adding a trailing `s` to the number, such as `1s` for 1 second. The default is `60000` milliseconds (1 minute). Note that this flag is dependent on the `--log_stats` flag being used.
`--mergepolicy <policy>`	Set the default merge policy of the MDEX Engine for partial updates. The value for *<policy>* must be either `balanced` or `aggressive`. If this flag is not used, `balanced` will be the default merge policy. For details on the merge policy, see the *Partial Updates Guide*.
`--net-timeout`	Specify the maximum number of seconds the Dgraph waits for the client to download data from queries across the network. The default network timeout value is `30` seconds.
`--nomrf`	Disable filtering for dynamic business rules.

Flag	Description
`--out <stdout/stderr file>`	Specify file path to which stdout/stderr should be remapped (the default is to use default stdout/stderr for the process). Running the Dgraph in an Oracle Endeca Application Controller environment creates a default file named `dgraph-S0-R0.out`.
`--persistdir`	Direct the Dgraph audit persistence file to a directory of your choice. By default, the file is written to a directory called `persist` that is located in the application's working directory. For details about the audit persistence file, see the *Endeca Performance Tuning Guide*. **Important:** Use the `--persistdir` flag only when you first start the Dgraph. Do not move or rename this directory after it has been created.
`--phrase_max <num>`	Specify the maximum number of words in each phrase for text search. The default number is 10. If the maximum number of words in a phrase is exceeded, the phrase is truncated to the maximum word count and a warning is logged.
`--pidfile <pidfile-path>`	Specify the file to which to write the process ID (pid). If unspecified, the default name of the pid file depends on how the Dgraph starts. Running the Dgraph in an EAC environment or from the command line creates a file named `dgraph.pid`.
`--port <num>`	Specify the port to use in server (non-interactive) mode. The default is `5555`.
`--search_max <num>`	Specify the maximum number of terms for text search. Default is `10`.
`--snip_cutoff <num>`	Limit the number of words in a property that the MDEX Engine evaluates to identify the snippet. If a match is not found within *<num>* words, the MDEX Engine does not return a snippet, even if a match occurs later in the property value. If the flag is not specified, or *<num>* is not specified, the default is 500.
`--snip_disable`	Globally disable snippeting.
`--spellpath <path>`	Specify location of spelling data files. Parameter should be a full path to a directory containing the needed aspell support files for spelling correction features (see the `--dym` and `--spl` options). Note that this path must be an absolute path (relative paths are not supported). In addition, this is a path to a directory containing at least the generic pspell/aspell support files. This does not need to be the same as the location of the .spelldat file for the indexed data set. The Dgraph typically requires write permissions in this directory, unless a correct or writable .pwli file is already available in this directory.
`--spell_bdgt <num>`	Set maximum number of variants considered for spelling and DYM (Did You Mean?) correction (the default is 32).
`--spell_nobrk`	Disable word-break analysis in the suggestion engine. Normally, in addition to considering spelling corrections, the suggestion engine considers alternate word separation points for the query to generate suggestions for DYM (Did You Mean?) and auto-correct.

Flag	Description
`--spl`	Enable auto-suggest spelling corrections for record (full text) and dimension search.
`--spl_hthresh <thresh>`	Specify the minimum number of hits at or above which auto-correct suggestions will not be generated. The default is `1`, meaning that if there are one or more hits for a user's search, then auto-correct does not provide spelling suggestions. Stated differently, if you use the default of 1 and there are zero (0) hits for a user's search, then spelling auto-correct does engage and provides suggestions for alternate keyword spellings.
`--spl_nsug <count>`	Specify the maximum number of auto-correct suggestions to return. The default is `1`.
`--spl_sthresh <thresh>`	Specify the threshold spelling correction score for words used as auto-correct suggestions. The default is `125`.
`--sslcertfile <certfile-path>`	Specify the path of the `eneCert.pem` certificate file that will be used by the Dgraph to present to any client for SSL communications. Using this flag provides the certificate which the MDEX Engine presents to the client for SSL; this option also forces HTTPS connections rather than HTTP. If not given, SSL is not enabled for Dgraph communications.
`--sslcafile <CA-certfile-path>`	Specify the path of the `eneCA.pem` Certificate Authority file that the Dgraph will use to authenticate SSL communications with other Endeca components. This flag defines the Certificate Authority file the MDEX Engine uses to validate client connections for mutual authentication purposes. If not given, SSL mutual authentication is not performed.  **Note:** If you need to establish a secure but not authenticated connection, use the `--sslcertfile` flag without the `--sslcafile` flag.
`--sslcipher <cipher-list>`	Specify one or more cryptographic algorithms, one of which Dgraph will use during the SSL negotiation. If you omit this setting, the Dgraph chooses a cryptographic algorithm from its internal list of algorithms. See the *Endeca Commerce Security Guide* for more information
`--stat-all`	Enable all available dynamic dimension value attributes. Note that this option has performance implications and is not intended for production use.
`--stat-abins`	Enable refinement counts for aggregated records. A refinement count is the number of records that would be in the result set if you were to refine on a dimension value. An aggregated record is a record that represents several records that are rolled up into a single record for display purposes. If you use this flag, the refinement counts reflect how many aggregated records the MDEX Engine would return in a result set if you were to refine on a dimension value. In general, the MDEX Engine calculates refinement counts as follows: • When returning regular (non-aggregated) record results, the MDEX Engine calculates refinement counts per refinement. (You enable refinement counts in Developer Studio.) The refinement counts for

Flag	Description
	regular records are returned by the MDEX Engine as the `Dgraph.Bins` property. • When returning aggregated record results, the `--stat-abins` flag lets the MDEX Engine return the refinement counts for aggregated records. These counts accurately reflect the number of aggregated records per refinement. (You enable refinement counts for aggregated records by using this flag.) The refinement counts for aggregated records are returned by the MDEX Engine as the `Dgraph.AggrBins` property. Note that dynamic statistics on aggregated records is an expensive computation for the MDEX Engine. Use this flag only if you intend to display the refinement counts for aggregated records in your front-end application.
`--syslog`	Direct all output to syslog.
`--thesaurus_cutoff <limit>`	Set a limit on the number of words in a user's search query that are subject to thesaurus replacement. The default value of <*limit*> is 3. This means that up to 3 words in a user's search query can be replaced with thesaurus entries. If there are more terms in the query that match thesaurus entries, none of the words are thesaurus expanded. This option is intended as a performance guard against very expensive thesaurus queries. Lower values improve thesaurus engine performance.
`--thesaurus_multiword_nostem`	Specify that words in a multiple-word thesaurus form should be treated like phrases and should not be stemmed, which increases performance for some query loads. Single-word terms will be subject to stemming regardless of whether this flag is specified. This flag prevents the Dgraph from expanding multi-word thesaurus forms by stemming. Thesaurus entries continue to match any stemmed form in the query, but multi-word expansions only include explicitly listed forms. To get the multi-word stemmed thesaurus expansions, the various forms must be listed explicitly in the thesaurus.
`--threads <num>`	Specify the number of threads in the MDEX Engine threading pool. The value of <num> must be a positive integer (that is, 1 or greater). The default for <num> is 2. The recommended number of threads for the MDEX Engine is typically equal to the number of cores on the MDEX Engine server. Additional threads are also started to perform internal maintenance tasks that are less CPU-intensive and do not affect query processing or updates (their number cannot be controlled).
`--tmpdir <dir>`	Specify the path to a temporary directory to be used to hold temporary files (the default is the base directory of *db_prefix*).

Flag	Description
`--unctrct`	Specify to the Dgraph not to compute implicit dimensions, and to only compute and present explicitly specified dimensions, when displaying refinements in navigation results. Specifying this flag does not reduce the size of the resulting record set that is being displayed; however, it improves run-time performance of the MDEX Engine. Be aware that if you use this flag, in order to receive meaningful navigation refinements, you need to make top-level precedence rules work for ALL outbound queries.
`--updatedir <dir>`	Specify the directory into which completed partial update files will be placed. Partial update files are also read from this directory.
`--updatelog`	Specify the file for update-related log messages. If unspecified, the default name of the update file depends on how the Dgraph starts. Running the Dgraph in a Control System environment (deprecated) or from the command line creates a default named `dgraph.updatelog`. Running the Dgraph in an Endeca Application Manager environment creates a default named `dgraph-S0-R0-update.log`.
`--updateverbose`	Show verbose messages while processing updates.
`--version`	Print version information and exit.
`--warmupseconds <seconds>`	Specify a duration for post-update cache warming (in seconds).
`--wb_maxbrks`	In word-break analysis, specify the maximum number of breaks to insert or remove per query. The default is `1`.
`--wb_minbrklen`	In word-break analysis, specify the minimum length of a new word-break term. The default is `2`.
`--wb_noibrk`	In word-break analysis, disable word-break insertion analysis.
`--wb_norbrk`	In word-break analysis, disable word-break removal analysis.
`--wildcard_max <count>`	Specify the maximum number of terms that can match a wildcard term in a wildcard query that contains punctuation, such as `ab*c.def*`. The default is `100`.
`--whymatch`	*Deprecated in MDEX Engine 6.2.0.* Enable computation of "Why Did It Match" dynamic record attributes returned as results of full-text search queries. These dynamic attributes contain a copy of the property/dimension key and value that caused the match, along with query interpretation notes (spelling, thesaurus, and so on).
`--whymatchConcise`	*Deprecated in MDEX Engine 6.2.0.* Similar to `--whymatch`, but produces more concise dynamic attribute values containing only the property/dimension key and query interpretation notes. This is useful when the property value might include large amounts of text, such as document contents.
`--wordinterp`	Enable computation of word interpretation dynamic supplement (or see-also) objects, which report on alternate forms of user query terms considered by the text search engine while processing full-text (record) search requests.

Flag	Description
`--xquery_fndoc <mode>`	*Deprecated in MDEX Engine 6.3.0.* Specifies the handling of the `fn:doc()` function within XQuery. The following three values are supported: • `none` causes all calls to `fn:doc()` to fail. • `sandbox` allows `fn:doc()`, but interprets its argument as a relative path within the XML subdirectory of the XQuery service directory. • `open` allows `fn:doc()` and interprets its argument as a URL. If not specified, defaults to `none`. Note that `open` is not supported for use in deployed applications.
`--xquery_path <path>`	*Deprecated in MDEX Engine 6.3.0.*Specify the directory in which XQuery Web service resources are located. XQuery main modules and WSDL files are loaded from this directory. Library modules are loaded from the `lib` subdirectory. If not specified, a user XQuery path is not used.

Appendix C

XML Configuration Files

This section describes the XML configuration files in an application. As a system administrator, you do not typically create these files. However, you need to understand and be able to locate the files, in case you need to modify or move them.

About the XML configuration files

You use Developer Studio to manage the subset of application configuration that isn't exposed through tooling in Workbench. Deveoper Studio allows you to manage the XML configuration files that control aspects of the behavior of the Forge, Dgidx, and Dgraph processes.

The XML configuration files are documented in detail in the *Endeca XML Reference*.

For reference, the following aspects of an application are configured through Workbench:

- **Business Rules** — modify with the **Experience Manager** tool in Workbench.
- **Thesaurus entries** — modify with the **Thesaurus** tool in Workbench.
- **Keyword redirects** — modify with the **Keyword Redirects** tool in Workbench.
- **Phrasing** — modify with the **Automatic Phrasing** tool in Workbench.

Generally, Workbench is the system of record for configuring any aspect of an application that is exposed through Workbench tools. Use Developer Studio to configure features that are not exposed through Workbench, such as your Forge pipeline, dimension search behavior, and available search interfaces.

Note: Use Workbench or Developer Studio to modify your application configuration whenever possible. Only modify the XML files directly when required in unusual situations. If this is necessary, avoid writing Byte Order Marks (BOMs) into the files.

Creating XML configuration files

The XML configuration files are created as part of your application when you deploy a new application using the Deployment Template.

A set of default XML configuration files are created in the deployed application's `config/pipeline` directory. These files are initially prefixed with the application name, such as `MyApp.Dimensions.xml`.

Note: Do not change the `output-prefix` value in the Indexer Adapter pipeline component in Developer Studio. Changing this setting creates a new set of XML files is saved with that modified prefix. The

Deployment Template project, however, retains the default prefix and can copy older files into the output directory when running compoments via the EAC.

The `config/pipeline` directory also contains the Developer Studio data ingest pipeline file, `pipeline.epx`.

Appendix D

Deployment Template Script Reference

This section describes scripts that are included with the Deployment Template, provides additional sample scripts, and provides information about running and configuring them.

Deployment Template script reference

The Deployment Template includes a set of utility scripts with deployed applications.

The following scripts are available in the `control` directory of a deployed application:

Script	Purpose
`baseline_update`	Runs a baseline update.
`export_site`	Takes a path to an XML file as an argument and exports the content in the Endeca Configuration Repository to the specified XML file. If no file is specified, site data is exported to `<App_Name>-<timestamp>.xml`, where the timestamp format is `YYYY-MM-DD_HH-MM-SS`.
`get_editors_config`	Exports editor configuration to the `<app dir>\config\ifcr\configuration\tools\xmgr` directory.
`get_media`	Exports media configuration to the `<app dir>\config\ifcr\media` directory.
`get_templates`	Exports template configuration to the `<app dir>\config\import\template` s directory.
`import_site`	Takes a path to an XML file and imports the content to the Endeca Configuration Repository. Optionally, you can use the `--force` flag to override the confirmation prompt for overwriting site content that already exists.
`load_baseline_test_data`	Copies data from the `<app dir>\test_data\baseline\` directory to `<app dir>\data\incoming` for a baseline update and calls the `set_baseline_data_ready_flag` script.

Script	Purpose
load_partial_test_data	Copies data from the <app dir>\test_data\partial\ directory to <app dir>\data\partials\incoming for a partial update and calls the set_partial_data_ready_flag script.
partial_update	Runs a partial update.
promote_content	Promotes content and configuration in the authoring environment to the live environment.
runcommand	Provides a means of invoking methods in AppConfig.xml against specified instances of objects. You can run runcommand with the --help flag for a list of command line arguments and flags.
set_baseline_data_ready_flag	Sets the baseline_data_ready flag in the EAC.  **Note:** This script is not required in applications that use CAS to produce MDEX-compatible output.
set_editors_config	Imports editor configuration from <app dir>\config\ifcr\configuration\tools\xmgr.
set_media	Imports media from <app dir>\config\ifcr\media.
set_partial_data_ready_flag	Sets the partial_extract flag in the EAC.
set_templates	Imports templates from <app dir>\config\import\templates.
initialize_services	This script should be run once after deploying an application. It does the following: • Removes existing application provisioning • Sets new EAC provisioning and performs initial setup

Provisioning scripts

The EAC allows scripts to be provisioned and invoked via Web service calls. A script is provisioned by specifying a working directory, a log directory into which output from the script is recorded, and a command to execute the script.

The AppConfig.xml document allows defined scripts to be provisioned by specifying the command used to invoke the script from the command line. When the provisioning configuration information is included, the script

is provisioned and becomes available for invocation via Web service calls or from the EAC Admin console in Oracle Commerce Workbench. When excluded, the script is not provisioned.

```
  <script id="BaselineUpdate">
    <log-dir>./logs/provisioned_scripts</log-dir>
    <provisioned-script-command>
        ./control/baseline_update.bat
    </provisioned-script-command>
    <bean-shell-script>
      <![CDATA[
...
      ]]>
    </bean-shell-script>
  </script>
```

The command line used to invoke scripts can always be specified in this form, relative to the default Deployment Template working directory:

```
./control/runcommand.[sh|bat] [script id]
```

Forge-based data processing

The Deployment Template supports running baseline and partial updates using Forge. In this processing model, an update essentially runs a CAS crawl (if applicable), Forge, Dgidx, and then updates the Dgraphs in an application.

Dgraph baseline update script using Forge

The baseline update script defined in the `DataIngest.xml` document for a Dgraph deployment is included in this section, with numbered steps indicating the actions performed at each point in the script.

```
<script id="BaselineUpdate">
    <![CDATA[
  log.info("Starting baseline update script.");
```

1. Obtain lock. The baseline update attempts to set an "`update_lock`" flag in the EAC to serve as a lock or mutex. If the flag is already set, this step fails, ensuring that the update cannot be started more than once simultaneously, as this would interfere with data processing. The flag is removed in the case of an error or when the script completes successfully.

   ```
       // obtain lock
       if (LockManager.acquireLock("update_lock")) {
   ```

2. Validate data readiness. Check that a flag called "`baseline_data_ready`" has been set in the EAC. This flag is set as part of the data extraction process to indicate that files are ready to be processed (or, in the case of an application that uses direct database access, the flag indicates that a database staging table has been loaded and is ready for processing). This flag is removed as soon as the script copies the data out of the `data/incoming` directory, indicating that new data may be extracted.

   ```
       // test if data is ready for processing
       if (Forge.isDataReady()) {
   ```

3. Clean processing directories. Files from the previous update are removed from the `data/processing`, `data/forge_output`, `data/temp`, `data/dgidx_output` and `data/partials/cumulative_partials` directories.

   ```
       // clean directories
       Forge.cleanDirs();
   ```

```
    PartialForge.cleanCumulativePartials();
    Dgidx.cleanDirs();
```

4. Copy data to processing directory. Extracted data in `data/incoming` is copied to `data/processing`.

```
    // fetch extracted data files to forge input
    Forge.getIncomingData();
```

5. Release Lock. The "`baseline_data_ready`" flag is removed from the EAC, indicating that the incoming data has been retrieved for baseline processing.

```
    LockManager.releaseLock("baseline_data_ready");
```

6. Copy config to processing directory. Configuration files are copied from `data/complete_index_config` to `data/processing`.

```
    // fetch config files to forge input
    Forge.getConfig();
```

7. Archive Forge logs. The `logs/forges/Forge` directory is archived, to create a fresh logging directory for the Forge process and to save the previous Forge run's logs.

```
    // archive logs
    Forge.archiveLogDir();
```

8. Forge. The Forge process executes.

```
    Forge.run();
```

9. Archive Dgidx logs. The `logs/dgidxs/Dgidx` directory is archived, to create a fresh logging directory for the Dgidx process and to save the previous Dgidx run's logs.

```
    // archive logs
    Dgidx.archiveLogDir();
```

10. Dgidx. The Dgidx process executes.

```
    Dgidx.run();
```

11. Distribute index to each server. A single copy of the new index is distributed to each server that hosts a Dgraph. If multiple Dgraphs are located on the same server but specify different `srcIndexDir` attributes, multiple copies of the index are delivered to that server.
12. Update MDEX Engines. The Dgraphs are updated. Engines are updated according to the `restartGroup` property specified for each Dgraph. The update process for each Dgraph is as follows:
    a. Create `dgraph_input_new` directory.
    b. Create a local copy of the new index in `dgraph_input_new`.
    c. Stop the Dgraph.
    d. Archive Dgraph logs (e.g. `logs/dgraphs/Dgraph1`) directory.
    e. Rename `dgraph_input` to `dgraph_input_old`.
    f. Rename `dgraph_input_new` to `dgraph_input`.
    g. Start the Dgraph.
    h. Remove `dgraph_input_old`.

This somewhat complex update functionality is implemented to minimize the amount of time that a Dgraph is stopped. This restart approach ensures that the Dgraph is stopped just long enough to rename two directories.

```
    // distributed index, update Dgraphs
    DistributeIndexAndApply.run();
```

```
<script id="DistributeIndexAndApply">
    <bean-shell-script>
      <![CDATA[
    DgraphCluster.cleanDirs();
    DgraphCluster.copyIndexToDgraphServers();
    DgraphCluster.applyIndex();
      ]]>
    </bean-shell-script>
  </script>
```

13. If Workbench integration is enabled, upload post-Forge dimensions to Oracle Commerce Workbench. The latest dimension values generated by the Forge process are uploaded to Oracle Endeca Workbench, to ensure that any new dimension values (including values for autogen dimensions and external dimensions) are available to Oracle Endeca Workbench for use in, for example, dynamic business rule triggers.

Note: This action does not add new dimensions or remove existing dimensions. These changes can be made by invoking the `update_web_studio_config.[bat|sh]` script.

```
    // if Workbench is integrated, update Workbench with latest
    // dimension values
    if (ConfigManager.isWebStudioEnabled()) {
       ConfigManager.cleanDirs();
       Forge.getPostForgeDimensions();
       ConfigManager.updateWsDimensions();
    }
```

14. Archive index and Forge state. The newly created index and the state files in Forge's state directory are archived on the indexing server.

```
    // archive state files, index
    Forge.archiveState();
    Dgidx.archiveIndex();
```

15. Cycle LogServer. The LogServer is stopped and restarted. During the downtime, the LogServer's error and output logs are archived.

```
    // cycle LogServer
    LogServer.cycle();
```

16. Release Lock. The "`update_lock`" flag is removed from the EAC, indicating that another update may be started.

```
    // release lock
    LockManager.releaseLock("update_lock");

    log.info("Baseline update script finished.");
      } else {
    log.warning("Failed to obtain lock.");
      }
    ]]>
  </bean-shell-script>
</script>
```

Dgraph partial update script using Forge

The partial update script defined in the `DataIngest.xml` document for a Dgraph deployment is included in this section, with numbered steps indicating the actions performed at each point in the script.

```
<script id="PartialUpdate">
  <bean-shell-script>
    <![CDATA[
```

1. Obtain lock. The partial update attempts to set an "`update_lock`" flag in the EAC to serve as a lock or mutex. If the flag is already set, this step fails, ensuring that the update cannot be started more than once simultaneously, as this would interfere with data processing. The flag is removed in the case of an error or when the script completes successfully.

```
    log.info("Starting partial update script.");
      // obtain lock
      if (LockManager.acquireLock("update_lock")) {
```

2. Validate data readiness. Test that the EAC contains at least one flag with the prefix "`partial_extract::`". One of these flags should be created for each successfully and completely extracted file, with the prefix "`partial_extract::`" prepended to the extracted file name (e.g. "`partial_extract::adds.txt.gz`"). These flags are deleted during data processing and must be created as new files are extracted.

```
    // test if data is ready for processing
    if (PartialForge.isPartialDataReady()) {
```

3. Archive partial logs. The `logs/partial` directory is archived, to create a fresh logging directory for the partial update process and to save the previous run's logs.

```
    // archive logs
    PartialForge.archiveLogDir();
```

4. Clean processing directories. Files from the previous update are removed from the `data/partials/processing`, `data/partials/forge_output`, and `data/temp` directories.

```
    // clean directories
    PartialForge.cleanDirs();
```

5. Move data and config to processing directory. Extracted files in `data/partials/incoming` with matching "`partials_extract::`" flags in the EAC are moved to `data/partials/processing`. Configuration files are copied from `config/pipeline` to `data/processing`.

```
    // fetch extracted data files to forge input
    PartialForge.getPartialIncomingData();

    // fetch config files to forge input
    PartialForge.getConfig();
```

6. Forge. The partial update Forge process executes.

```
    // run ITL
    PartialForge.run();
```

7. Apply timestamp to updates. The output XML file generated by the partial update pipeline is renamed to include a timestamp, to ensure it is processed in the correct order relative to files generated by previous or following partial update processes.

```
    // timestamp partial, save to cumulative partials dir
    PartialForge.timestampPartials();
```

8. Copy updates to cumulative updates. The timestamped XML file is copied into the cumulative updates directory.

```
    PartialForge.fetchPartialsToCumulativeDir();
```

9. Distribute update to each server. A single copy of the partial update file is distributed to each server specified in the configuration.

```
    // distribute partial update, update Dgraphs
    DgraphCluster.copyPartialUpdateToDgraphServers();
```

10. Update MDEX Engines. The Dgraph processes are updated. Engines are updated according to the `updateGroup` property specified for each Dgraph. The update process for each Dgraph is as follows:

    a. Copy update files into the `dgraph_input/updates` directory.
    b. Trigger a configuration update in the Dgraph by calling the URL `admin?op=update`.

```
    DgraphCluster.applyPartialUpdates();
```

11. Archive cumulative updates. The newly generated update file (and files generated by all partial updates processed since the last baseline) are archived on the indexing server.

```
    // archive partials
    PartialForge.archiveCumulativePartials();
```

12. Release Lock. The "`update_lock`" flag is removed from the EAC, indicating that another update may be started.

```
    // release lock
    LockManager.releaseLock("update_lock");
    log.info("Partial update script finished.");
      }
      else {
        log.warning("Failed to obtain lock.");
      }
    ]]>
  </bean-shell-script>
</script>
```

Preventing non-nillable element exceptions

When running the partial updates script, you may see a Java exception similar to this example:

```
INFO: Starting copy utility 'copy_partial_update_to_host_MDEXHost1'.
Oct 20, 2008 11:46:37 AM org.apache.axis.encoding.ser.BeanSerializer serialize
SEVERE: Exception:
java.io.IOException: Non nillable element 'fromHostID' is null.
...
```

If this occurs, make sure that the following properties are defined in the `AppConfig.xml` configuration file:

```
<dgraph-defaults>
  <properties>
      ...
      <property name="srcPartialsDir" value="./data/partials/forge_output" />
      <property name="srcPartialsHostId" value="ITLHost" />
      <property name="srcCumulativePartialsDir" value="./data/partials/cumula¬
tive_partials" />
      <property name="srcCumulativePartialsHostId" value="ITLHost" />
      ...
    </properties>
  ...
</dgraph-defaults>
```

The reason is that the script is obtaining the `fromHostID` value from this section.

Dgraph baseline update script using Forge and a CAS full crawl script

After running a full CAS crawl, you can run a baseline update using Forge to incorporate the records from a Record Store instance.

This example runs a *baseline* update that includes a *full* CAS crawl. The crawl writes output to a Record Store instance and then Forge incorporates the records from the crawl. To create this sequential workflow of CAS crawl and then baseline update, you can do the following:

- Remove the default `Forge.isDataReady` check from the baseline update script. This call handles concurrency control around Forge input files. The Record Store has built-in logic to handle concurrency between read and write operations, so no external concurrency control is required. Removing this call means that the lock manager does not check the flag or wait on the flag to be cleared before running a CAS crawl.
- Add a call to `runBaselineCasCrawl()` to run the full CAS crawl.
- Remove the call to `Forge.getIncomingData()` that fetches extracted data files.

For example, this baseline update script calls `CAS.runBaselineCasCrawl("MyCrawl")` which runs a full CAS crawl that writes output to a Record Store instance. Then the script continues with baseline update processing.

```
<!--
    ###########################################################################
    # Baseline update script
    #
  -->
  <script id="BaselineUpdate">
    <log-dir>./logs/provisioned_scripts</log-dir>
   <provisioned-script-command>./control/baseline_update.bat</provisioned-script-
command>
    <bean-shell-script>
      <![CDATA[
    log.info("Starting baseline update script.");
    // obtain lock
    if (LockManager.acquireLock("update_lock")) {

        // call the baseline crawl script to run a full CAS
        // crawl.
        CAS.runBaselineCasCrawl("MyCrawl");

        // clean directories
        Forge.cleanDirs();
        PartialForge.cleanCumulativePartials();
        Dgidx.cleanDirs();

        // fetch extracted data files to forge input
        Forge.getIncomingData();
        LockManager.removeFlag("baseline_data_ready");

        // fetch config files to forge input
        Forge.getConfig();

        // archive logs and run ITL
        Forge.archiveLogDir();
        Forge.run();
        Dgidx.archiveLogDir();
        Dgidx.run();

        // distributed index, update Dgraphs
        DistributeIndexAndApply.run();
```

```
      WorkbenchManager.cleanDirs();
      Forge.getPostForgeDimensions();
      WorkbenchManager.updateWsDimensions();

      // archive state files, index
      Forge.archiveState();
      Dgidx.archiveIndex();

      // (start or) cycle the LogServer
      LogServer.cycle();

    // release lock
    LockManager.releaseLock("update_lock");
    log.info("Baseline update script finished.");
    } else {
         log.warning("Failed to obtain lock.");
    }
    ]]>
  </bean-shell-script>
</script>
```

You run the baseline update by running `baseline_update` in the `apps/<app dir>/control` directory.

For example:

```
C:\Endeca\apps\DocApp\control>baseline_update.bat
```

Dgraph partial update script using Forge and a CAS incremental crawl script

After running an *incremental* CAS crawl, you can run a *partial* update that incorporates the records from a Record Store instance.

To create this sequential workflow of incremental CAS crawl and then partial update, you can do the following:

- Remove the default `PartialForge.isPartialDataReady` check from the partial update script. This call handles concurrency control around Forge input files. The Record Store has built-in logic to handle concurrency between read and write operations, so no external concurrency control is required. Removing this call means that the lock manager does not check the flag or wait on the flag to be cleared before running a CAS crawl.
- Add a call `runIncrementalCasCrawl()` to run the incremental CAS crawl.
- If the pipeline does not read from sources in the Forge incoming directory, remove the call to `Partial¬ Forge.getPartialIncomingData()` that fetches extracted data files.

For example, this partial update script calls `CAS.runIncrementalCasCrawl("MyCrawl")` which runs an incremental CAS crawl named `MyCrawl`. Then the script continues with partial update processing.

```
  <!--
    ##########################################################################
    # Partial update script
    #
  -->
  <script id="PartialUpdate">
    <log-dir>./logs/provisioned_scripts</log-dir>
   <provisioned-script-command>./control/partial_update.bat</provisioned-script-
command>
    <bean-shell-script>
      <![CDATA[
    log.info("Starting partial update script.");
```

```
    // obtain lock
    if (LockManager.acquireLock("update_lock")) {

       // call the partial crawl script to run an incremental
       // CAS crawl.
       CAS.runIncrementalCasCrawl("MyCrawl");

       // archive logs
       PartialForge.archiveLogDir();

       // clean directories
       PartialForge.cleanDirs();

       // fetch config files to forge input
       PartialForge.getConfig();

       // run ITL
       PartialForge.run();

       // timestamp partial, save to cumulative partials dir
       PartialForge.timestampPartials();
       PartialForge.fetchPartialsToCumulativeDir();

       // distribute partial update, update Dgraphs
       DgraphCluster.cleanLocalPartialsDirs();
       DgraphCluster.copyPartialUpdateToDgraphServers();
       DgraphCluster.applyPartialUpdates();

       // archive partials
       PartialForge.archiveCumulativePartials();

      // release lock
      LockManager.releaseLock("update_lock");
      log.info("Partial update script finished.");
    } else {
      log.warning("Failed to obtain lock.");
      }
      ]]>
    </bean-shell-script>
  </script>
```

You run the partial update by running `partial_update` in the `apps/<app dir>/control` directory. For example:

```
C:\Endeca\apps\DocApp\control>partial_update.bat
```

Multiple CAS crawls and Forge updates

There are more complicated cases where multiple CAS crawls are running on their own schedules, and Forge updates are running on their own schedules. To coordinate this asynchronous workflow of CAS crawls and baseline or partial updates, you add code that calls methods in `ContentAcquisitionServerComponent`.

In your `DataIngest.xml` code, the main coordination task is one of determining how you time running CAS crawls and how you time running baseline or partial updates that consume records from those crawls. For example, suppose you have an application that runs three full CAS crawls and those records are consumed by a single baseline update. In that scenario, each of the three full crawls has its own full crawl script in `DataIngest.xml` that runs on a nightly schedule. And the `DataIngest.xml` file contains a baseline update that runs nightly to consume the latest generation of records from each of the three crawls. The `Forge.is¬DataReady` check is not required in the baseline update script because the source data is not locked.

CAS-based data processing

The Deployment Template supports running baseline and partial updates using CAS as a replacement for Forge. In this processing model, the update runs a CAS crawl to produce MDEX-compatible output. This is the step that removes the need for Forge. Then the update runs Dgidx and updates the Dgraphs in an application.

Dgraph baseline update script using CAS

You do not need to run Forge if you run a CAS crawl that is configured to produce MDEX-compatible output as part of your update process.

This example runs a *baseline* update that includes a *full* CAS crawl. The crawl writes MDEX compatible output and then the update invokes Dgidx to process the records, dimensions, and index configuration produced by the crawl. To create this sequential workflow of CAS crawl and then baseline update, you add a call to `runBaselineCasCrawl()` to run the CAS crawl.

For example, this baseline update script calls `CAS.runBaselineCasCrawl("${lastMileCrawlName}")` which runs a CAS crawl that writes MDEX-compatible output. Then the script continues with baseline update processing by running Dgidx and distributing the index files.

```
<!--
    ###########################################################################
    # Baseline update script
    #
  -->
  <script id="BaselineUpdate">
    <log-dir>./logs/provisioned_scripts</log-dir>
   <provisioned-script-command>./control/baseline_update.bat</provisioned-script-
command>
    <bean-shell-script>
      <![CDATA[
    log.info("Starting baseline update script.");
    // obtain lock
    if (LockManager.acquireLock("update_lock")) {
      // clean directories
      CAS.cleanCumulativePartials();
      Dgidx.cleanDirs();

      // run crawl and archive any changes in the dvalId mappings
      CAS.runBaselineCasCrawl("${lastMileCrawlName}");
      CAS.archiveDvalIdMappingsForCrawlIfChanged("${lastMileCrawlName}");

      // archive logs and run the Indexer
      Dgidx.archiveLogDir();
      Dgidx.run();

      // distributed index, update Dgraphs
      DistributeIndexAndApply.run();

      // Upload the generated dimension values to Workbench
      WorkbenchManager.cleanDirs();
      CAS.copyOutputDimensionsFile("${lastMileCrawlName}", WorkbenchManager.get¬
WorkbenchTempDir());
      WorkbenchManager.updateWsDimensions();

      // Upload the generated config to Workbench
      WorkbenchManager.updateWsConfig();
```

```
      // archive state files, index
      Dgidx.archiveIndex();

      // (start or) cycle the LogServer
      LogServer.cycle();
      // release lock
      LockManager.releaseLock("update_lock");
      log.info("Baseline update script finished.");
    } else {
      log.warning("Failed to obtain lock.");
    }
      ]]>
    </bean-shell-script>
  </script>
```

You run the baseline update by running `baseline_update` in the `apps/<app dir>/control` directory.

For example:

```
C:\Endeca\apps\DocApp\control>baseline_update.bat
```

Dgraph partial update script using CAS

You do not need to run Forge if you run a CAS crawl that is configured to produce MDEX-compatible output as part of your update process.

This example runs an *incremental* CAS crawl that writes MDEX compatible output and then runs a *partial* update to process data records. Remember that in an incremental CAS crawl, the index configuration and dimension value records are not processed.

To create this sequence of CAS crawl and then partial update, you add a call to `runIncrementalCasCrawl()` to run the CAS crawl. For example, this partial update script calls `CAS.runIncrementalCasCrawl("${lastMileCrawlName}")` which runs a CAS crawl that writes MDEX-compatible output. Then the script continues with update processing by running Dgidx and distributing the index files.

Note: In some applications, the `archiveDvalIdMappingsForCrawlIfChanged` call can take modest amounts of processing time (for example, typically less than 10 seconds). This method is recommended in all but the most time-sensitive partial update scenarios.

```
<!--
    ###########################################################################
    # Partial update script
    #
  -->
  <script id="PartialUpdate">
    <log-dir>./logs/provisioned_scripts</log-dir>
   <provisioned-script-command>./control/partial_update.bat</provisioned-script-
command>
    <bean-shell-script>
      <![CDATA[
    log.info("Starting partial update script.");
    // obtain lock
    if (LockManager.acquireLock("update_lock")) {

      // run crawl and archive any changes in the dvalId mappings
      CAS.runIncrementalCasCrawl("${lastMileCrawlName}");
      CAS.archiveDvalIdMappingsForCrawlIfChanged("${lastMileCrawlName}");

      // Copy the partial to the master cumulative directory
```

```
      CAS.fetchPartialsToCumulativeDir("${lastMileCrawlName}");

      // copy from srcPartials to localCumulative for authoring
      AuthoringDgraphCluster.copyPartialUpdateToDgraphServers();

     // copy from local to mdex's update-dir and trigger the update for authoring

      AuthoringDgraphCluster.applyPartialUpdates();

      // copy from srcPartials to localCumulative for live
      LiveDgraphCluster.copyPartialUpdateToDgraphServers();

      // copy from localCumulative to mdex's update-dir and trigger the update
      LiveDgraphCluster.applyPartialUpdates();

      // Archive accumulated partials
      CAS.archiveCumulativePartials();

      // release lock
      LockManager.releaseLock("update_lock");
      log.info("Partial update script finished.");
    } else {
      log.warning("Failed to obtain lock.");
    }
      ]]>
    </bean-shell-script>
  </script>
```

You run the baseline update by running `partial_update` in the `apps/<app dir>/control` directory.

For example:

```
C:\Endeca\apps\DocApp\control>partial_update.bat
```

CAS crawl scripts for Record Store output

This topic provides an example CAS crawl script with a crawl that is configured to write to Record Store output. To create a similar CAS crawl script in your application, add code to `AppConfig.xml` that specifies the CAS crawl to run locks the crawl (to wait for any running crawls to complete), runs the crawl, and releases the lock. Depending on your environment, you may need a script that runs a full CAS crawl and a script that runs an incremental CAS crawl.

This example `AppConfig.xml` code runs a *full* crawl that writes to a Record Store instance:

```
<!--
    ###############################################################################
    # full crawl script
    #
 -->

  <script id="MyCrawl_fullCrawl">
    <log-dir>./logs/provisioned_scripts</log-dir>
    <provisioned-script-command>./control/runcommand.bat MyCrawl_fullCrawl
run</provisioned-script-command>
    <bean-shell-script>
      <![CDATA[
    crawlName = "MyCrawl";

    log.info("Starting full CAS crawl '" + crawlName + "'.");
```

```
    // obtain lock
    if (LockManager.acquireLock("crawl_lock_" + crawlName)) {

      CAS.runBaselineCasCrawl(crawlName);

      LockManager.releaseLock("crawl_lock_" + crawlName);
    }
    else {
      log.warning("Failed to obtain lock.");
    }

    log.info("Finished full CAS crawl '" + crawlName + "'.");
      ]]>
    </bean-shell-script>
  </script>
```

This example runs an *incremental* crawl that writes to a Record Store instance:

```
   <!--
    ###########################################################################
    # incremental crawl script
    #
  -->
  <script id="MyCrawl_IncrementalCrawl">
    <log-dir>./logs/provisioned_scripts</log-dir>
   <provisioned-script-command>./control/runcommand.bat MyCrawl_IncrementalCrawl
 run</provisioned-script-command>
    <bean-shell-script>
      <![CDATA[
    crawlName = "MyCrawl";

    log.info("Starting incremental CAS crawl '" + crawlName + "'.");

    // obtain lock
    if (LockManager.acquireLock("crawl_lock_" + crawlName)) {

      CAS.runIncrementalCasCrawl(crawlName);

      LockManager.releaseLock("crawl_lock_" + crawlName);
    }
    else {
      log.warning("Failed to obtain lock.");
    }

    log.info("Finished incremental CAS crawl '" + crawlName + "'.");
      ]]>
    </bean-shell-script>
  </script>
```

CAS crawl scripts for record file output

This topic provides an example CAS crawl script with a crawl that is configured to write to record file output. To create a similar CAS crawl script in your application, add code to `DataIngest.xml` that specifies the CAS crawl to run locks the crawl (to wait for any running crawls to complete), runs the crawl, and releases the lock. Depending on your environment, you may need a script that runs a full CAS crawl and a script that runs an incremental CAS crawl.

This example `DataIngest.xml` code runs a *full* crawl that writes to record file output:

```
<!--
    ##########################################################################
    # full crawl script
    #
 -->

  <script id="MyCrawl_fullCrawl">
    <log-dir>./logs/provisioned_scripts</log-dir>
    <provisioned-script-command>./control/runcommand.bat MyCrawl_fullCrawl
run</provisioned-script-command>
    <bean-shell-script>
      <![CDATA[
    crawlName = "MyCrawl";

    log.info("Starting full CAS crawl '" + crawlName + "'.");

    // obtain lock
    if (LockManager.acquireLock("crawl_lock_" + crawlName)) {

        if (!CAS.isCrawlFileOutput(crawlName)) {
           throw new UnsupportedOperationException("The crawl " + crawlName +
             " does not have a File System output type. The only supported output
 type for this script is File System.");
        }

        log.info("Starting full CAS crawl '" + crawlName + "'.");
        // Remove all files from the crawl's output directory
        CAS.cleanOutputDir(crawlName);
        CAS.runBaselineCasCrawl(crawlName);
        // Rename the output to files to include the crawl name
        // so they do not collide with the output from other crawls
        CAS.renameBaselineCrawlOutput(crawlName);

        destDir = PathUtils.getAbsolutePath(CAS.getWorkingDir(),
           CAS.getCasCrawlFullOutputDestDir());

        // create the target dir, if it doesn't already exist
        mkDirUtil = new CreateDirUtility(CAS.getAppName(),
           CAS.getEacHost(), CAS.getEacPort(), CAS.isSslEnabled());
        mkDirUtil.init(CAS.getCasCrawlOutputDestHost(), destDir, CAS.getWork¬
ingDir());
        mkDirUtil.run();

        // clear the destination dir of full crawl from previous crawls
        CAS.clearFullCrawlOutputFromDestinationDir(crawlName);

        // remove previously collected incremental crawl files,
        // which are expected to be incorporated in this full crawl
        CAS.clearIncrementalCrawlOutputFromDestinationDir(crawlName);

        // copy the full crawl output to destination directory
        CAS.copyBaselineCrawlOutputToDestinationDir(crawlName);
        LockManager.releaseLock("crawl_lock_" + crawlName);
     }

    else {
      log.warning("Failed to obtain lock.");
    }
```

```
    log.info("Finished full CAS crawl '" + crawlName + "'.");
      ]]>
    </bean-shell-script>
  </script>
```

This example `DataIngest.xml` code runs an *incremental* crawl that writes to record file output:

```
   <!--
    ##########################################################################
    # incremental crawl script
    #
  -->
  <script id="MyCrawl_IncrementalCrawl">
    <log-dir>./logs/provisioned_scripts</log-dir>
   <provisioned-script-command>./control/runcommand.bat MyCrawl_IncrementalCrawl
 run</provisioned-script-command>
    <bean-shell-script>
      <![CDATA[
    crawlName = "MyCrawl";

    log.info("Starting incremental CAS crawl '" + crawlName + "'.");

    // obtain lock
    if (LockManager.acquireLock("crawl_lock_" + crawlName)) {

       if (!CAS.isCrawlFileOutput(crawlName)) {
          throw new UnsupportedOperationException("The crawl " + crawlName +
            " does not have a File System output type. The only supported output
 type for this script is File System.");
       }

       log.info("Starting incremental CAS crawl '" + crawlName + "'.");
       // Remove all files from the crawl's output directory
       CAS.cleanOutputDir(crawlName);
       CAS.runIncrementalCasCrawl(crawlName);
       // Timestamp and rename the output to files to include the
       // crawl name so they do not collide with the output from
       // previous incremental output from this crawl or incremental
       // output from other crawls
       CAS.renameIncrementalCrawlOutput(crawlName);

       destDir = PathUtils.getAbsolutePath(CAS.getWorkingDir(),
          CAS.getCasCrawlIncrementalOutputDestDir());

       // create the target dir, if it doesn't already exist
       mkDirUtil = new CreateDirUtility(CAS.getAppName(),
          CAS.getEacHost(), CAS.getEacPort(), CAS.isSslEnabled());
       mkDirUtil.init(CAS.getCasCrawlOutputDestHost(), destDir, CAS.getWork¬
ingDir());
       mkDirUtil.run();

       // copy crawl output to destination directory
       // Note: We assume a downstream process removes incremental crawl output
       // from this directory that has already been processed.
       CAS.copyIncrementalCrawlOutputToDestinationDir(crawlName);

       LockManager.releaseLock("crawl_lock_" + crawlName);
     }

     else {
        log.warning("Failed to obtain lock.");
```

```
    }

   log.info("Finished incremental CAS crawl '" + crawlName + "'.");
     ]]>
   </bean-shell-script>
 </script>
```

Report generation

Four report generation scripts are defined in the `DataIngest.xml` document.

Two of the scripts are used to generate XML reports for Oracle Endeca Workbench and two generate HTML reports that can be viewed in a browser. All scripts share similar functionality, so only one is included below, with numbered steps indicating the actions performed at each point in the script.

```
 <script id="DailyReports">
   <bean-shell-script>
     <![CDATA[
   log.info("Starting daily Workbench report generation script.");
```

1. Obtain lock. The report generation script attempts to set a "`report_generator_lock`" flag in the EAC to serve as a lock or mutex. If the flag is already set, this step fails, ensuring that the report generator cannot be started more than once simultaneously, as the default report generators share input directories and working directories. The flag is removed in the case of an error or when the script completes successfully.

   ```
       if (LockManager.acquireLock("report_generator_lock")) {
   ```

2. Clean working directories. Clear any files in the report generator's input directory.

   ```
         // clean report gen input dir
         DailyReportGenerator.cleanInputDir();
   ```

3. Distribute configuration files to each server. A single copy of the Dgraph configuration files is distributed to each server specified in the configuration.

   ```
   DgraphCluster.copyDgraphConfigToDgraphServers();
   ```

4. Roll LogServer. If the LogServer is actively writing to a file and the file is required for the specified time range, the LogServer needs to be rolled in order to free up the log file. This code handles that test and invokes the roll administrative URL command on the LogServer, if necessary.

   ```
         // roll the logserver, if the report requires the active log file
         if (LogServer.isActive() &&
             LogServer.yesterdayIncludesLatestLogFile()) {
           LogServer.callLogserverRollUrl();
         }
   ```

5. Retrieve logs for specified report. The LogServer identifies log files in its output directory that are required to generate a report for the requested date range. Those files are copied to the target directory configured for the LogServer. Note that this step could be modified to include retrieving logs from multiple LogServers, if more than one is deployed.

   ```
         // retrieve required log files for processing
         LogServer.copyYesterdayLogFilesToTargetDir();
   ```

6. Update Report Generator to the appropriate time range and output file name. Oracle Commerce Workbench requires reports to be named according to a time stamp convention. The Report Generator component's provisioning is updated to specify the appropriate time range, time series and output filename. The output file path in the existing provisioning is updated to use the same path, but to use the date stamp as the

filename. Files default to a ".xml" extension, though the component will attempt to retain a ".html" extension, if specified in the AppConfig.xml.

```
      // update report generator to the appropriate dates, time series
      // and to output a timestamped file, as required by Workbench
      DailyReportGenerator.updateProvisioningForYesterdayReport();
```

7. Archive logs. If one or more files were copied into the report generator's input directory, report generation will proceed. Start by archiving logs associated with the previous report generator execution.

```
      if (DailyReportGenerator.reportInputDirContainsFiles()) {
        // archive logs
        DailyReportGenerator.archiveLogDir();
```

8. Run report generator. Execute the report generation process.

```
        // generate report
        DailyReportGenerator.run();
```

9. Copy report to Oracle Workbench report directory. By default, Oracle Commerce Workbench reads reports from a directory in its workspace. Typically, the directory is [ENDECA_TOOLS_CONF]/reports/[appName]/daily or [Endeca_TOOLS_CONF]/reports/[appName]/weekly. Starting in Oracle Commerce Workbench 1.0.1, this location can be configured by provisioning a host named "webstudio" with a custom directory named "webstudio-report-dir." The Deployment Template provisions this directory and delivers generated reports to that location for Workbench to read. The report file (and associated charts) will be copied to this directory, as specified in the AppConfig.xml, which defaults to <app dir>/reports. Note that this step is not necessary for HTML reports, as those reports are not viewed in Oracle Endeca Workbench.

```
        // copy generated report and charts
        // defined in "webstudio" host and its "webstudio-report-dir"
        // directory
        reportHost = "webstudio";
        absDestDir = PathUtils.getAbsolutePath(webstudio.getWorkingDir(),
            webstudio.getDirectory("webstudio-report-dir"));
        isDaily = true;
        DailyReportGenerator.copyReportToWebStudio(reportHost,
            absDestDir, isDaily);
      }
      else {
        log.warning("No log files for report generator to process.");
      }

      LockManager.releaseLock("report_generator_lock");
      log.info("Finished daily Workbench report generation.");
    }
    else {
      log.warning("Failed to obtain lock.");
    }
      ]]>
    </bean-shell-script>
  </script>
```

Appendix E

The Endeca Application Controller eaccmd Utility

This section describes the eaccmd command-line tool, which can be used to access the Endeca Application Controller directly. Oracle recommends using Deployment Template scripts to provision and manage your application components. The commands listed here are intended for use in debugging an application in the EAC.

Running eaccmd

This topic describes how to run eaccmd.

The eaccmd tool is installed by default in `%ENDECA_ROOT%\bin` on Windows. On UNIX, it is `$ENDECA_ROOT/bin`. You run eaccmd within a scripting environment such as Bash or Perl. You can run eaccmd on any machine as long as it is pointing at the EAC Central Server.

The eaccmd syntax is platform-independent.

eaccmd usage

This topic describes the usage of eaccmd.

eaccmd usage is as follows:

```
eaccmd host:eac_port <cmd> [--async] [-verbose]
```

where settings in square brackets ([ ]) are optional and <cmd> is one of:

```
[Provisioning commands:]
  define-app [--app app_id] [--def def_file]
  describe-app --app app_id [--canonical]
  remove-app [--force] --app app_id
  list-apps
[Incremental Provisioning commands:]
  add-component --app app_id [--comp comp_id] --def def_file
  add-host --app app_id [--host host_id] --def def_file
  add-script --app app_id --script script_id (--def def_file |
    [--wd working_dir] [--log-file log_file] --cmd command [args...])
  remove-component [--force] --app app_id --comp comp_id
  remove-host [--force] --app app_id --host host_id
  remove-script --app app_id --script script_id
  update-component [--force] --app app_id [--comp comp_id] --def def_file
  update-host [--force] --app app_id [--host host_id] --def def_file
  update-script [--force] --app app_id --script script_id
```

```
    (--def def_file | [--wd working_dir] [--log-file log_file]
    --cmd command [args...])
[Synchronization commands:]
  set-flag --app app_id --flag flag
  remove-flag --app app_id --flag flag
  remove-all-flags --app app_id
  list-flags --app app_id
[Component and Script Control commands:]
  start --app app_id [--comp comp_id | --script script_id]
  stop --app app_id [--comp comp_id | --script script_id]
  status --app app_id [--comp comp_id | --script script_id]
[Utility commands:]
  ls --app app_id --host host_id --pattern file_pattern
  start-util --type shell --app app_id [--token token]
    --host host_id [--wd working_dir] --cmd command [args...]
  start-util --type copy --app app_id [--token token] [--recursive]
    --from host_id --to host_id --src src_path --dest dest_path
  start-util --type backup --app app_id [--token token] --host host_id
    --dir ls [--method <copy|move>] [--backups num_backups]
  start-util --type rollback --app app_id [--token token] --host host_id
    --dir ls
  stop-util --app app_id --token token
  status-util --app app_id --token token
```

eaccmd Feedback

Eaccmd gives no feedback in case of success (that is, if a component is running or completed or a service is completed). If an operation fails, a FAILED message is printed to the screen.

If instead you want eaccmd to run asynchronously, you must use the `--async` flag as follows:

```
eaccmd host:port <cmd> [--async]
```

About incremental provisioning

With incremental provisioning, it is possible to add, remove, or modify one or more hosts, components, or scripts without having to bring down an entire implementation.

Incrememental provisioning guidelines

The following guidelines apply to incremental provisioning.

- Scripts can be changed at any time, as long as they are not running.
- Properties on either hosts or components can be changed at any time.
- Anything other than a property on a component cannot be changed, nor can a component be removed, if the component is either running or unreachable.
- Anything other than a property or a directory on a host cannot be changed, nor can a host be removed, if any components or utilities on it are running, or if the host is unreachable.

You can attempt to override the constraints mentioned above by using the `--force` flag.

About the def_file setting

The def_file is the provisioning document used to add a component or host to the implementation.

You can use a file that specifies exactly one component or host, or a file that includes multiple entries. If you use a larger provisioning file, you must specify which of the listed components or hosts you wish to add.

For example, say you want to add a host called new_host to your application. You could add provisioning information for new_host to your existing provisioning file, `AppConfig.xml`. When you run the `add-host` command, you pass in the host name as well as the file name.

In the case of scripts, you have two options: you can use a def_file, as you do with components and hosts, or you can provide the necessary information individually, through the `--cmd` (command), `--wd` (working directory), and `--log-file` settings.

About the --force flag

The `--force` flag indicates whether or not the Application Controller should attempt to force any running components, utilities, or scripts to stop before attempting an update or a remove operation.

In the case of updates, the update persists in the application provisioning regardless of whether the forced stop is successful, even if this leaves a dangling process somewhere.

Examples

- In the case of a component, the command:

  ```
  update-component --force --app myApp --name forge
  ```

  stops the `forge` component (if it is running) before updating it.
- In the case of a host, the command:

  ```
  remove-host --force --app myApp --name dev777
  ```

  stops any running components or services on host `dev777` before removing that host.
- In the case of a script, the command:

  ```
  update-script --force --app myApp --script newbaseline.pl
  --cmd perl
  ```

  stops the script `newbaseline.pl` before updating it.

Adding a component in eaccmd

You can use eaccmd to add components to your application.

To add a component in eaccmd, use the following syntax:

```
add-component --app app_id [--comp comp_id] --def def_file
```

For example:

```
add-component --app myApp --comp new_forge --def AppConfig.xml
```

Removing a component in eaccmd

You can use eaccmd to remove components from applications.

To remove a component in eaccmd, use the following syntax:

```
remove-component [--force] --app app_name --comp comp_id
```

For example:

```
remove-component --force --app myApp --comp forge
```

Modifying a component in eaccmd

You can use eaccmd to modify components in an application.

To change the attributes of a previously-defined component in eaccmd, use the following syntax:

```
update-component [--force] --app app_id [--comp comp_id]
--def def_file
```

For example:

```
update-component --force --app myApp --def newDgraphProps.xml
```

Adding a host in eaccmd

You can use eaccmd to add hosts to your application.

To add a host in eaccmd, use the following syntax:

```
add-host --app app_id [--host host_id] --def def_file
```

For example:

```
add-host --app myApp --host mktg022 --def myApp.xml
```

Removing a host in eaccmd

You can use eaccmd to remove hosts from an application.

To remove a host in eaccmd, use the following syntax:

```
remove-host [--force] --app app_id --host host_id
```

For example:

```
remove-host --force --app myApp --host dev777
```

Modifying a host in eaccmd

You can use eaccmd to modify hosts in an application.

To change the attributes of a previously-defined host in eaccmd, use the following syntax:

```
update-host [--force] --app app_id [--host host_id]
--def def_file
```

For example:

```
update-host --force --app myApp --host mktg022
--def newMktgHostProps.xml
```

Adding a script in eaccmd

You can use eaccmd to add scripts to your application.

To add a script in eaccmd, use the following syntax:

```
add-script --app app_id --script script_id [--cmd command --wd working_dir --log-
file log_file] | [--def def_file]
```

For example:

```
add-script --app myApp --script newbaseline.pl --cmd perl
```

Removing a script in eaccmd

You can use eaccmd to remove scripts from applications.

To remove a script in eaccmd, use the following syntax:

```
remove-script [--force] --app app_id --script script_id
```

For example:

```
remove-script --app myApp --script testbaseline.pl
```

Modifying a script in eaccmd

You can use eaccmd to modify a script in an application.

To modify an existing script in eaccmd, use the following syntax:

```
update-script [--force]  --app app_id --script script_id [--cmd command --wd
working_dir --log-file log_file] | [--def def_file]
```

For example:

```
update-script --app myApp --script newbaseline.pl --def myApp.xml
```

Component and utility status verbosity

By default, eaccmd provides single-word component and utility status messages, such as Running. To receive more detailed feedback, you can run eaccmd with the `--verbose` flag.

This flag provides useful information beyond simply the state.

Server component status verbosity

The following is an example of a verbose status message for a server component. Server components include the Dgraph and LogServer.

```
State: NotRunning
Start time: 10/11/08 3:58 PM
Failure Message:
```

Batch component status verbosity

The following is an example of a verbose status message for a batch component. Batch components include Forge, Dgidx, and ReportGenerator.

```
State: NotRunning
Start time: 10/11/08 3:58 PM
Duration: 0 days 0 hours 0 minutes 6.96 seconds
Failure Message:
```

Specifying the EAC Central Server host and port

The `eaccmd.properties` file supplies host and port information to eaccmd.

In the `eaccmd.properties` file, which is located in the `$ENDECA_CONF/conf` directory on UNIX and `%ENDECA_CONF%\conf` on Windows, you can specify a host and port for eaccmd to use. (The default values are host=localhost and port=8888.) With this file in place, you do not have to specify the host and port on the command line.

If your EAC Central Server is not on `localhost:8888`, you must either edit the file to point to the correct host and port or continue to specify `host:port` on the command line. Any `host:port` specified on the command line overrides the settings in the `eaccmd.properties` file.

Application component reference

This section includes details and examples about the following components: Forge, Dgidx, Dgraph, LogServer, and ReportGenerator.

Forge

A Forge element launches the Forge (Data Foundry) software, which transforms source data into tagged Endeca records.

Every Application Controller component contains the following attributes:

Attribute	Description
component-id	Required. The name of this instance of the component.
host-id	Required. The alias of the host upon which the component is running.
properties	An optional list of properties, consisting of a required name and an optional value.

The Forge element contains the following sub-elements:

Sub-element	Description

args	Command-line flags to pass to Forge, expressed as a set of arg sub-elements. If an argument takes a value, the argument and value must be on separate lines in the provisioning file. For example:
	`<args>`
	`  <arg>--threads</arg>`

Sub-element	Description
	<arg>3</arg> </args>
input-dir	The path to the Forge input.
log-file	Name of the Forge log file. If the log-file is not specified, the default is component working directory plus component name plus ".log".
output-prefix-name	The implementation-specific prefix name, without any associated path information.
output-dir	Directory where the output from the Forge process will be stored.
pipeline-file	Required. Name of the Pipeline.epx file to pass to Forge.
num-partitions	The number of partitions.
working-dir	Working directory for the process that is launched. If it is specified, it must be an absolute path. If any of the other properties of this component contain relative paths, they are interpreted as relative to the working directory. If working-dir is not specified, it defaults to `$ENDECA_CONF/work/<appName>/ <componentName>` on UNIX, or `%ENDECA_CONF%\work\<appName>/ <componentName>` on Windows.
state-dir	The directory where the state file is located.
temp-dir	The temporary directory that Forge uses.
web-service-port	The port on which the Forge metrics Web service listens.
ssl-configuration	Both the parallel Forge and Forge metrics Web service can secure their communications with SSL. The `ssl-configuration` element contains three sub-elements of its own: • `cert-file`: The `cert-file` specifies the path of the eneCert.pem certificate file that is used by Forge processes to present to any client. This is also the certificate that the Application Controller Agent should present to Forge when trying to talk to it. The file name can be a path relative to the component's working directory. • `ca-file`: The `ca-file` specifies the path of the eneCA.pem Certificate Authority file that Forge processes uses to authenticate communications with other Guided Search components. The file name can be a path relative to the component's working directory. • `cipher`: Specify one or more cryptographic algorithms, one of which Dgraph will use during the SSL negotiation. If you omit this setting, the Dgraph chooses a cryptographic algorithm from its internal list of algorithms. See the *Endeca Commerce Security Guide* for more information.

Example

The following example provisions a Forge component for use with the sample wine data:

```
<forge component-id="wine_forge" host-id="wine_indexer">
 <args>
  <arg>-vw</arg>
 </args>
```

```
 <num-partitions>1</num-partitions>
 <working-dir>
   C:\Endeca\PlatformServices\reference\sample_wine_data
 </working-dir>
 <pipeline-file>.\data\forge_input\pipeline.epx</pipeline-file>
 <input-dir>.\data\forge_input</input-dir>
 <output-dir>.\data\partition0\forge_output</output-dir>
 <state-dir>.\data\partition0\state</state-dir>
 <log-file>.\logs\wine_forge.log</log-file>
 <output-prefix-name>wine</output-prefix-name>
</forge>
```

Dgidx

A Dgidx component sends the finished data prepared by Forge to the Dgidx program, which generates the proprietary indices for each Dgraph.

Every Application Controller element contains the following attributes:

Attribute	Description
component-id	Required. The name of this instance of the component.
host-id	Required. The alias of the host upon which the component is running.
properties	An optional list of properties, consisting of a required name and an optional value.

The Dgidx element contains the following sub-elements:

Sub-element	Description

args	Command-line flags to pass to Dgidx, expressed as a set of arg sub-elements. If an argument takes a value, the argument and value must be on separate lines in the provisioning file. For example:
	`<args>`
	`  <arg>--threads</arg>`
	`  <arg>3</arg>`
	`</args>`

app-config-prefix	Path and file prefix that define the input for Dgidx. For example, in `/endeca/project/files/myProject`, files beginning with myProject in the directory `/endeca/project/files` are the ones to be considered.
output-prefix	Required. Path and prefix name for the Dgidx output. For example, `output_prefix = c:\temp\wine` generates files that start with “wine” in the `c:\temp` directory.
log-file	The path to and name of the Dgidx log files. If the log-file is not specified, the default is component working directory plus component name plus “.log”. Dgidx can generate three distinct log files: the basic component log file, and two files that log the subtasks described in run-aspell, below. • The file dgwordlist logs stdout/stderr for the dgwordlist subtask described below. The name of this file is derived from the Dgidx component’s log-file location, plus the term “dgwordlist”. If an extension

Sub-element	Description
	exists, “dgwordlist” is added before the extension. For example, if the original log-file is `C:\dir\dgidx-1.log`, then the dgwordlist log would be `C:\dir\dgidx-1.dgwordlist.log`. • The file aspellcopy logs the stdout/stderr for the subtask of uploading the Aspell files to Dgidx’s output directory, where the Dgraph can access them. The name of this file is derived from the Dgidx component’s log-file location, plus the term “aspellcopy”. If an extension exists, “aspellcopy” is added before the extension. For example, if the original log-file is `C:\dir\dgidx-1.txt`, then the aspellcopy log would be `C:\dir\dgidx-1.aspellcopy.txt`.
input-prefix	Required. Path and prefix name for the Forge output that Dgidx indexes.
working-dir	Working directory for the process that is launched. If it is specified, it must be an absolute path. If any of the other properties of this component contain relative paths, they are interpreted as relative to the working directory. If working-dir is not specified, it defaults to `$ENDECA_CONF/work/<appName>/<componentName>` on UNIX, or `%ENDECA_CONF%\work\<appName>/<componentName>` on Windows.
run-aspell	Specifies Aspell as the spelling correction mode for the implementation. This causes the Dgidx component to run dgwordlist and to copy the Aspell files to its output directory, where the Dgraph component can access them. The default is true. See log-file above for details on the logging of these subtasks. For Aspell details, see the *MDEX Engine Development Guide*.
temp-dir	A temporary directory used by this component.

Example

The following example provisions a Dgidx component to work with the sample wine data:

```
<dgidx component-id="wine_dgidx" host-id="wine_indexer">
 <args>
  <arg>-v</arg>
 </args>
 <working-dir>
   C:\Endeca\PlatformServices\reference\sample_wine_data
 </working-dir>
 <input-prefix>.\data\partition0\forge_output\wine</input-prefix>
 <app-config-prefix>
   .\data\partition0\forge_output\wine
 </app-config-prefix>
 <output-prefix>.\data\partition0\dgidx_output\wine</output-prefix>
 <log-file>.\logs\wine_dgidx.log</log-file>
 <run-aspell>true</run-aspell>
</dgidx>
```

Dgraph

A Dgraph element launches the Dgraph (MDEX Engine) software, which processes queries against the indexed Endeca records.

Every Application Controller component contains the following attributes:

Attribute	Description
component-id	Required. The name of this instance of the component.
host-id	Required. The alias of the host upon which the component is running.
properties	An optional list of properties, consisting of a required name and an optional value.

The Dgraph element contains the following sub-elements:

Sub-element	Description

args	Command-line flags to pass to Dgraph, expressed as a set of arg sub-elements. If an argument takes a value, the argument and value must be on separate lines in the provisioning file. For example:
	`<args>`
	`  <arg>--threads</arg>`
	`  <arg>3</arg>`
	`</args>`

port	Required. The port at which the Dgraph should listen. The default is 8000.
log-file	The path to and name of the Dgraph log file. If the log-file is not specified, the default is component working directory plus component name plus ".log".
input-prefix	Required. Path and prefix name for the Dgidx output that the Dgraph uses as an input.
app-config-prefix	Path and file prefix that define the input for the Dgraph. For example, in `/endeca/project/files/myProject`, files beginning with myProject in the directory `/endeca/project/files` are the ones to be considered.
working-dir	Working directory for the process that is launched. If it is specified, it must be an absolute path. If any of the other properties of this component contain relative paths, they are interpreted as relative to the working directory. If working-dir is not specified, it defaults to `$ENDECA_CONF/work/<appName>/<componentName>` on UNIX, or `%ENDECA_CONF%\work\<appName>/<componentName>` on Windows.
startup-timeout	Specifies the amount of time in seconds that the Application Controller waits while starting the Dgraph. If it cannot determine that the Dgraph is running in this timeframe, it times out. The default is 60.
req-log-file	Path to and name of the request log.
spell-dir	If specified, is the directory in which the Dgraph will look for Aspell files. If it is not specified, the Dgraph will look for Aspell files in the Dgraph's input directory (that is, input-prefix without the prefix). For example, if input-prefix is /dir/prefix and all the Dgraph input files are /dir/prefix.*, the Dgraph will look for the Aspell files in /dir/.
update-dir	Specifies the directory from which the Dgraph reads partial update file. For more information, see the *Endeca Partial Updates Guide*.
update-log-file	Specifies the file for update-related log messages.
temp-dir	A temporary directory used by this component.

Sub-element	Description
ssl-configuration	Contains three sub-elements of its own: • `cert-file`: The `cert-file` specifies the path of the eneCert.pem certificate file that is used by the Dgraph to present to any client. This is also the certificate that the Application Controller Agent should present to the Dgraph when trying to talk to the Dgraph. The file name can be a path relative to the component's working directory. • `ca-file`: The `ca-file` specifies the path of the eneCA.pem Certificate Authority file that the Dgraph uses to authenticate communications with other Guided Search components. The file name can be a path relative to the component's working directory. • `cipher`: Specify one or more cryptographic algorithms, one of which Dgraph will use during the SSL negotiation. If you omit this setting, the Dgraph chooses a cryptographic algorithm from its internal list of algorithms. See the *Endeca Commerce Security Guide* for more information..

Example

The following example provisions an SSL-enabled Dgraph component for use with the sample wine data:

```
<dgraph component-id="wine_dgraph" host-id="wine_indexer">
 <args>
   <arg>--spl</arg>
   <arg>--dym</arg>
 </args>
 <port>8000</port>
 <working-dir>
   C:\Endeca\PlatformServices\reference\sample_wine_data
 </working-dir>
 <input-prefix>.\data\partition0\dgraph_input\wine</input-prefix>
 <app-config-prefix>
   .\data\partition0\dgraph_input\wine
 </app-config-prefix>
 <log-file>.\logs\wine_dgraph.log</log-file>
 <req-log-file>.\logs\wine_dgraph_req_log.out</req-log-file>
 <startup-timeout>120</startup-timeout>
 <ssl-configuration>
   <cert-file>
     C:\Endeca\PlatformServices\workspace\etc\eneCert.pem
   </cert-file>
   <ca-file>
     C:\Endeca\PlatformServices\workspace\etc\eneCA.pem
   </ca-file>
   <cipher>AES128-SHA</cipher>
 </ssl-configuration>
</dgraph>
```

LogServer

The LogServer component controls the use of the Endeca Log Server.

Every Application Controller component contains the following attributes:

Attribute	Description
component-id	Required. The name of this instance of the component.
host-id	Required. The alias of the host upon which the component is running.
properties	An optional list of properties, consisting of a required name and an optional value.

The LogServer component contains the following sub-elements:

Sub-element	Description
ort	Required. Port on which to run the LogServer.
output-prefix	Required. Path and prefix name for the LogServer output. For example, `output_prefix = c:\temp\wine` generates files that start with “wine” in `c:\temp`.
gzip	Required. Controls the archiving of log files. Possible values are true and false.
working-dir	Working directory for the process that is launched. If it is specified, it must be an absolute path. If any of the other properties of this component contain relative paths, they are interpreted as relative to the working directory. If working-dir is not specified, it defaults to `$ENDECA_CONF/work/<appName>/<componentName>` on UNIX, or `%ENDECA_CONF%\work\<appName>/<componentName>` on Windows.
startup-timeout	Specifies the amount of time in seconds that the eaccmd waits while starting the LogServer. If it cannot determine that the LogServer is running in this timeframe, it times out. The default is 60.
log-file	The path to the LogServer log file. If the log-file is not specified, the default is component working directory plus component name plus “.log”.

Example

The following example provisions a LogServer component based on the sample wine data.

```
<logserver component-id="wine_logserver" host-id="wine_indexer">
 <port>8002</port>
 <working-dir>
  C:\Endeca\PlatformServices\reference\sample_wine_data
 </working-dir>
 <output-prefix>.\logs\logserver_output\wine</output-prefix>
 <gzip>false</gzip>
 <startup-timeout>120</startup-timeout>
 <log-file>.\logs\wine_logserver.log</log-file>
</logserver>
```

ReportGenerator

The ReportGenerator component runs the Report Generator, which processes Log Server files into HTML-based reports that you can view in your Web browser and XML reports that you can view in Workbench.

Every Application Controller component contains the following attributes:

Attribute	Description
component-id	Required. The name of this instance of the component.
host-id	Required. The alias of the host upon which the component is running.
properties	An optional list of properties, consisting of a required name and an optional value.

The ReportGenerator component contains the following sub-elements:

Sub-element	Description
working-dir	Working directory for the process that is launched. If it is specified, it must be an absolute path. If any of the other properties of this component contain relative paths, they are interpreted as relative to the working directory. If working-dir is not specified, it defaults to `$ENDECA_CONF/work/<appName>/<componentName>` on UNIX, or `%ENDECA_CONF%\work\<appName>/<componentName>` on Windows.
input-dir-or-file	Required. Path to the file or directory containing the logs to report on. If it is a directory, then all log files in that directory are read. If it is a file, then just that file is read.
output-file	Required. Name the generated report file and path to where it is stored. For example: `C:\Endeca\reports\myreport.html` on Windows `/endeca/reports/myreport.html` on UNIX
stylesheet-file	Required. Filename and path of the XSL stylesheet used to format the generated report. For example: `%ENDECA_CONF%\etc\ report_stylesheet.xsl` on Windows `$ENDECA_CONF/etc/report_stylesheet.xsl` on UNIX
settings-file	Path to the report_settings.xml file. For example: `%ENDECA_CONF%\etc\report_settings.xml` on Windows `$ENDECA_CONF/etc/report_settings.xml` on UNIX
timerange	Sets the time span of interest (or report window). Allowed keywords: • Yesterday • LastWeek • LastMonth • DaySoFar • WeekSoFar • MonthSoFar These keywords assume that days end at midnight, and weeks end on the midnight between Saturday and Sunday.
start-date <date> stop-date <date>	These set the report window to the given date and time. The date format should be either `yyyy_mm_dd` or `yyyy_mm_dd.hh_mm_ss`. For example, `2009_10_23.19_30_57` expresses Oct 23, 2009 at 7:30:57 in the evening.

Sub-element	Description
time-series	Turns on the generation of time-series data and specifies the frequency, Hourly or Daily.
charts	Turns on the generation of report charts. Disabled by default.
log-file	The path to the ReportGenerator log file. If the log-file is not specified, the default is component working directory plus component name plus ".log".
java_binary	Should indicate a JDK 1.5.x or later. Defaults to the JDK that Endeca installs.
java_options	Command-line options for the java_binary setting. This command is primarily used to adjust the ReportGenerator memory, which defaults to 1GB. To set the memory, use the following: `java_options = -Xmx[MemoryInMb]m -Xms[MemoryInMb]m`
args	Command-line flags to pass to the ReportGenerator, expressed as a set of arg sub-elements.

Example

The following example provisions a ReportGenerator component based on the sample wine data.

```
<reportgenerator component-id="wine_gen_html_report" host-id="wine_indexer">
 <working-dir>
  C:\Endeca\PlatformServices\reference\sample_wine_data
 </working-dir>
 <input-dir-or-file>.\logs\logserver_output</input-dir-or-file>
 <output-file>.\reports\daily\daily_report.html</output-file>
 <stylesheet-file>.\etc\report_stylesheet.xsl</stylesheet-file>
 <settings-file>.\etc\report_settings.xml</settings-file>
 <timerange>day-so-far</timerange>
 <charts>true</charts>
 <log-file>.\logs\wine_gen_html_report.log</log-file>
</reportgenerator>
```

eaccmd command reference

The eaccmd tool contains commands for provisioning, resource configuration, and component use.

Provisioning commands

The provisioning commands make it possible for you to define and manage your applications from the command line.

Command	Description
define-app [--app app_id] [--def def_file]	Defines an application. Def_file takes an XML provisioning file, a sample of which, `sample_wine_definition.xml`, is located in the `%ENDECA_REFERENCE_DIR%\sample_wine_data\etc` directory on Windows, or the `$ENDECA_`

Command	Description
	REFERENCE_DIR\sample_wine_data\etc directory on UNIX. The provisioning file typically contains an application ID. If eaccmd specifies a different app_id for the same application, the eaccmd version overrides the one in provided in the provisioning file.
describe-app --app app_id [--canonical]	Describes an application. Returns an XML file in the format used by the def_file setting of define-app. If --canonical is specified, all paths are canonicalized.
remove-app [--force] --app app_id	Removes the named application. The optional --force flag indicates whether or not this remove operation should force any running components or services to stop before attempting the remove. Remove fails if any components or services are still running (that is, not forced to stop).
list-apps	Lists all defined applications.

Provisioning example

The following example defines an application called my_wine. (In this and all examples that follow we assume that the host and port are set in the eaccmd.properties file and so do not need to be included on the command line.)

```
eaccmd define-app --app my_wine --def sample_wine_definition.xml
```

Incremental provisioning commands

The incremental provisioning commands make it possible for you to add, remove, or update a host, component, or script without having to bring down the entire application.

Command	Description
add-component --app app_id [--comp comp_id] --def def_file	Adds a single component to an application. Def_file is a provisioning document. You can use a larger provisioning file for this purpose, or you can use one that specifies exactly one component or host. If you choose to use a larger provisioning file, then you must specify which component listed within it that you are adding, using the --comp flag.
add-host --app app_id [--host host_id] --def def_file	Adds a single host to an application. Def_file is a provisioning document. You can use a larger provisioning file for this purpose, or you can use one that specifies exactly one component or host. If you choose to use a larger provisioning file, then you must specify which host listed within it that you are adding, using the --host flag.
add-script --app app_id --script script_id (--def def_file | [--wd working_dir] [--log-file log_file] --cmd command [args...])	Adds a script to an application. Scripts can be added at any time. You can use --def to specify a definition file to start the script, or use the following settings:

Command	Description

	`--log-file` is the file for appended stdout/stderr output. If it is not specified, it defaults to `$ENDECA_CONF/logs/script/(app_id).(script_id).log`
	`--wd` is the working directory. If it is not specified, it defaults to `$ENDECA_CONF/working/(app_id)/`
	`--cmd` is the command that is used to start the script. If `--cmd` is omitted, the first unrecognized argument is taken as the start of your command. The `--log-file` and `--wd`, if used, should come before `--cmd`.

remove-component [--force] --app app_id --comp comp_id	Removes a single component from an application. The optional --force flag indicates whether or not this remove operation should force any running components or services to stop before attempting the remove. Remove fails if any components or services are still running (that is, not forced to stop).
remove-host [--force] --app app_id --host host_id	Removes a single host from an application. The optional `--force` flag indicates whether or not this remove operation should force any running components or services to stop before attempting the remove. Remove fails if any components or services are still running (that is, not forced to stop).
remove-script [--force] --app app_id --script script_id	Removes a script from an application. The optional `--force` flag indicates whether or not this remove operation should force a running script to stop before attempting the remove.
update-component [--force] --app app_id [--comp comp_id] --def def_file	Updates a component. Component properties can be updated at any time. Other changes cannot be made if the component is running or unreachable. The optional `--force` flag indicates that the Application Controller will attempt to force the conditions under which the specified updates can be made (by stopping stop a running component or utility invocation, for example). Regardless of whether or not the forced stop is successful, however, the update persists in the application provisioning, even if this leaves a dangling process somewhere.
update-host [--force] --app app_id [--host host_id] --def def_file	Updates a host. Host properties can be updated at any time. Other changes cannot be made if any components or services are running on the host, or if the host is unreachable. The optional `--force` flag indicates that the Application Controller will attempt to force the conditions under which the specified updates can be made (by stopping stop a running component or utility invocation, for example). Regardless of whether or not the forced stop is successful, however,

Command	Description
	the update persists in the application provisioning, even if this leaves a dangling process somewhere.

update-script [--force] --app app_id --script script_id (--def def_file | [--wd working_dir] [--log-file log_file] --cmd command [args...])	Updates a script. The optional `--force` flag indicates whether or not this update operation should force a running script to stop before attempting the update. You can use --def to specify a definition file to update the script, or use the following settings:
	`--wd` is the working directory. If it is not specified, it defaults to `$ENDECA_CONF/working/(app_id)/`
	`--log-file` is the file for appended stdout/stderr output. If it is not specified, it defaults to `$ENDECA_CONF/logs/script/(app_id).(script_id).log`
	`--cmd` is the command that is used to start the script. If `--cmd` is omitted, the first unrecognized argument is taken as the start of your command. The `--log-file` and `--wd`, if used, should come before `--cmd`.

Incremental provisioning example

The following example adds a Forge component to the my_wine application. Because this provisioning file contains only a single component, it is not necessary to use the `--comp` flag.

```
eaccmd add-component --app my_wine --def update_forge.xml
```

Synchronization commands

Synchronization commands are used by the Synchronization service (described below) to manage application-level flags that let users know when processes are in use.

Command	Description
set-flag --app app_id --flag flag	Sets a flag that demonstrates that a group of processes are in use. You specify the flag with the application name and a flag name, which may be arbitrary but should be well-known.
remove-flag --app app_id --flag flag	Removes the named flag and releases the reserved processes.
remove-all-flags --app app_id	Removes all flags in an application and releases all reserved processes.
list-flags --app app_id	Lists all flags in an application.

About the Synchronization service

The Synchronization service lets you create, query, and delete application-level flags on a series of processes. These flags indicate that the flagged processes are in use. The service creates flags on the fly at the user's request and deletes them when they are released. Using this service, multiple users can synchronize their

activities by obtaining and querying the flags. If two users attempt to flag the same processes at the same time an error occurs.

Synchronization service flags are identified by an application name/flag name pair. Because flag names are user-created and arbitrary, all users must be aware of flag names and consistent in their use. If a set of processes needs to be reserved, then everyone concerned needs to know the name of the flag.

Synchronization examples

The following example adds a flag called mkt1010 to the my_wine application:

```
eaccmd set-flag --app my_wine --flag mkt1010
```

The following example removes all flags in the my_wine application:

```
eaccmd remove-all-flags --app my_wine
```

Component and script control commands

The component and script control commands are used to start and stop components or scripts and retrieve their status.

Command	Description
start --app app_id [--comp comp_id | --script script_id]	Starts a component or a script.
stop --app app_id [--comp comp_id | --script script_id]	Stops a component or a script.
status --app app_id [--comp comp_id | --script script_id]	Gets the status of a component (one of Starting, Running, NotRunning, or Failed) or a script (one of Running, NotRunning, or Failed).

Component control example

The following example starts a Dgraph named wine_dgraph in the my_wine application.

```
eaccmd start --app my_wine --comp wine_dgraph
```

Utility commands

The utility commands allow you to run and monitor Application Controller utilities through the eaccmd tool.

There are three kinds of Utility commands: Shell, Copy, and Archive.

General notes on Application Controller utilities

Keep in mind the following general points about Application Controller utilities.

- Utility naming: Be sure to name your utilities carefully. If you create a new utility that has the same name as a running utility, an error is issued. However, if there is an existing utility with the same name that is not running, the new utility overwrites it.
- System cleanup of utility output: Each instance of the Shell and Copy utilities stores status information and output logs. The Application Controller clears this information for non-running utilities instances every seven days (that is, 10,080 minutes) to save system resources. This setting can be modified in the eac.properties file.

The List Directory Contents (ls) command

The List Directory Contents command lets you see the contents of directories on remote machines. Its behavior is similar to that of ls on UNIX, although some non-ls restrictions, noted below, apply.

Command	Description
ls --app app_id --host host_id --pattern file_pattern	Returns a list of files matching the pattern input in file_pattern. Note the following: A file_pattern must start with an absolute path, such as C:\ or /. A file_pattern can contain . or .. as directory names, and expands * and ? wildcards. A file_pattern cannot contain the wildcard expressions .*, .?, or ..* as directory or file names. Bracketed wildcards, such as file[123].txt, are not supported. Wildcards cannot be applied to drive names. You cannot use .. to create paths that do not exist. For example, the path `/temp/../../a.txt` refers to a path that is above the root directory. This is an invalid path that causes the operation to fail.

Wildcard behavior

The List Directory Contents command expands the wildcards in a pattern. If the expansion results in a file, it returns a file. If the expansion results in a directory, it returns the directory non-recursively. Wildcard expansion can result in any combination of files and directories.

For example, assume that the following directories and files exist:

```
/home/endeca/reference/...
/home/endeca/install.log
/home/e.txt
```

The following command:

```
eaccmd ls --app my_wine --host my_host --pattern /home/e\\*
```

would list all of these files and directories, because they match the file_pattern.

Delimiting wildcard arguments

To prevent inappropriate expansion, any wildcard arguments you use with the List Directory Contents utility in eaccmd need to be delimited with double quotation marks. For example: On Windows, "C:*.txt". On UNIX, "/home/endeca/test/*.txt".

The Shell utility

The Shell utility allows you to run arbitrary commands in a host system shell.

Command	Description
start-util --type shell --app app_id [--token token] --host host_id [--wd working_dir] --cmd command [args...]	Starts a Shell utility with the specified command string. The token is a string. If you do not specify a token, one is generated and returned when you start the utility. The token is used to stop the utility or to get its status. --wd, which is optional, sets the working directory for the process that gets launched. If specified, it must be an absolute path. If wd is not specified, the setting defaults to `%ENDECA_CONF%\working\`

Command	Description
	`<appName>\shell` on Windows or `$ENDECA_CONF/working/ <appName>/shell` on UNIX. The --cmd arguments are passed in a single string. If --cmd is omitted, the first unrecognized argument is taken as the start of your command.
stop-util --app app_id --token token	Stops a Shell utility. The token is a string, either user-created or generated and returned when you start the utility, that eaccmd prints to screen. The token can be used to stop the utility or to get its status.
status-util --app app_id --token token	Gets the status of a Shell utility. The token is a string, either user-created or generated and returned when you start the utility, that eaccmd prints to screen. The token can be used to stop the utility or to get its status.

Shell utility examples

The first example deletes the Dgidx output after it has been copied in a separate action over to the Dgraph:

```
eaccmd start-util --type shell --app my_wine --host mkt1010
--cmd rm <dgidx-output-dir>/*.*
```

The second example performs a recursive directory copy:

```
eaccmd start-util --type shell --app myapp --host hosttorunon
--cmd cp–r /mysourcedir /mydestdir
```

Troubleshooting the Shell utility

In many cases, particularly cross-platform scenarios, the Shell command must be wrapped in double quotation marks. The error message returned, which occurs at the console level, is usually something similar to the following:

```
The system cannot find the path specified.
```

The Copy utility

The Copy utility uses an internal Web services interface to copy files or directories, either locally or between machines.

Commands	Description
start-util --type copy --app app_id [--token token] [--recursive] --from host_id --to host_id --src file_pattern --dest dest_path	As part of the Copy utility, starts a copy. You identify the hostname, port, and path for both the source and destination directories. If the copy is local, you do not need to specify the host_id. Keep in mind that you are not necessarily copying to the machine you are running eaccmd on. The hosts you are copying to and from are those you specified in your provisioning file. `--token` is a string used to stop the utility or get its status. If you do not specify a token, one is generated and returned when you start the utility.

Commands	Description
	If `--recursive` is specified, it indicates that the Copy utility recursively copies any directories that match the wildcard. If `--recursive` is not specified, the Copy utility does not copy directories, even if they match the wildcard. Instead, it creates intermediate directories required to place the copied files at the destination path. `--src` is a string representing the file, wildcard, or directory to be copied. A `--src` must start with an absolute path, such as C:\ or /. A --src can contain . or .. as directory names, and expands * and ? wildcards. Note the following: • You cannot use the wildcard expressions .*, .?, or ..* as directory or file names. • Bracket wildcards, such as file[123].txt, are not supported. • Wildcards cannot be applied to drive names. `--dest` is the full path to the destination file or directory. `--dest` must be an absolute path, and no wildcards are allowed. If `--dest` is a directory, that directory must exist, unless the following conditions are met: • The parent of the destination already exists. • You are copying only one thing.
stop-util --app app_id --token token	Stops a Copy utility. The token is a string, either user-created or generated and returned when you start the utility, that eaccmd prints to screen. The token can be used to stop the utility or to get its status.
status-util --app app_id --token token	Gets the status of a Copy utility. The token is a string, either user-created or generated and returned when you start the utility, that eaccmd prints to screen. The token can be used to stop the utility or to get its status.

Copy utility examples

This section illustrates several different Copy actions. For simplicity, the majority of the Copy actions are done on a single machine. The final example shows how to copy across machines.

First, assume the following directory structure exists on the source:

```
/
 endeca1/
  work/
   dgraphlogs/
    a.log
   forgelogs/
    b.log
 endeca2/
```

```
  work/
   dgraphlogs/
    c.log
   forgelogs/
    d.log
    e.log
 destination/
```

The following command copies one file to a new name:

```
eaccmd start-util --type copy --app myApp
   --src "/endeca1/work/dgraphlogs/a.log" --dest "/destination/out.log"
```

The resulting directory change would look like this:

```
 destination/
  out.log
```

The following command copies one file into an existing directory:

```
eaccmd start-util --type copy --app myApp
   --src "/endeca1/work/dgraphlogs/a.log" --dest "/destination"
```

The resulting directory change would look like this:

```
destination/
 a.log
```

The following command recursively copies a directory to a new name:

```
eaccmd start-util --type copy --app myApp
   --src "/endeca1/work/dgraphlogs" --dest "/destination/outlogs" --recursive
```

The resulting directory change would look like this:

```
destination/
 outlogs/
  a.log
```

The following command recursively copies a directory into an existing directory:

```
eaccmd start-util --type copy --app myApp
   --src "/endeca1/work/dgraphlogs" --dest "/destination"
   --recursive
```

The resulting directory change would look like this:

```
destination/
 dgraphlogs/
  a.log
```

The following command copies all files in a directory.

```
eaccmd start-util --type copy --app myApp
   --src "/endeca2/work/forgelogs/*" --dest "/destination"
```

The resulting directory change would look like this:

```
destination/
  d.log
  e.log
```

The following copy command demonstrates the use of multiple wildcards:

```
eaccmd start-util --type copy --app myApp
   --src "/e*/work/*logs/*.log" --dest "/destination"
```

The resulting directory change would look like this:

```
destination/
  a.log
  b.log
  c.log
  d.log
  e.log
```

The following copy demonstrates a recursive copy with wildcards:

```
eaccmd start-util --type copy --app myApp
   --src "/e*/work" --dest "/destination" --recursive
```

The resulting directory change would look like this:

```
destination/
 work/
  dgraphlogs/
   a.log
   c.log
  forgelogs/
   b.log
   d.log
   e.log
```

When copying to another machine, the syntax is as follows:

```
eaccmd start-util --type copy --app myApp --from ITLHost --to MDEXHost
   --src /full/path/to/file/src.txt --dest /full/path/to/file/dest.txt
```

Keep in mind that the hostnames are not IP addresses or DNS names, but rather are the hosts that are defined within the EAC. If you are using the Deployment Template, these are the hosts defined in the `AppConfig.xml` file with tags similar to this example:

```
<host id="ITLHost" hostName="itl.example.com" port="8888" />
<host id="MDEXHost" hostName="mdex.example.com" port="8888" />
```

Also make sure that you have a clear network path between hosts (if necessary, make the appropriate modifications in any firewall to allow traffic).

About the Copy utility

This topic provides details about how the Copy utility works.

The Copy utility supports wildcards (* and ?) and recursive copying. In some cases, the destination directory must already exist; in others, the utility automatically creates both the destination directory and any empty directories in the transfer.

Directories are copied first to a temporary directory on the destination machine before being copied one file at a time to the target location. You can configure the location of this temporary directory in the eac.properties file, using the optional `setting com.endeca.eac.filetransfer.fileTransferTempDir` as follows:

- If this setting is defined as an absolute path, the Copy utility uses it.
- If it is defined as a relative path, the Copy utility considers it to be relative to `%ENDECA_CONF%/state/`
- If it is not defined, the Copy utility uses the directory `%ENDECA_CONF%/state/file_transfer/`

If the Copy utility tries to copy a file to a location where another file already exists, the utility overwrites the preexisting file.

Note: The Copy utility supports both SSL and non-SSL communication, with SSL being off by default. For details on enabling SSL, see the *Oracle Commerce Guided Search Security Guide*.

Destination directories

In most cases, the destination directory where the copied files are placed has to exist already. However, there are a few exceptions where the destination directory does not have to exist prior to the copy:

- Copying just one file to the location of an existing file.
- Copying just one file to a new file name in an existing directory.
- Copying just one directory to a new directory name in an existing parent directory.

Failure and recovery

The following situations result in a failure of the Copy utility:

- The Copy utility tries to write to a directory it doesn't have permissions to.
- There is not enough disk space.
- There is no file at the source location.
- The wildcard expression matches no files.
- When there are mismatches between directories and files (for example, the Copy utility tries to copy a file to path where a directory with that name already exists, or tries to create a directory in the destination and a file with that name already exists).
- You cannot use .. to create paths that do not exist. For example, the path `/temp/../../a.txt` refers to a path that is above the root directory. This is an invalid path that causes the utility to fail.
- Asking for a copy that results in multiple files being written to the same location. For example, given the following directory structure on the source:

  ```
  /trunk/src/a.txt
  /testbranch/src/a.txt
  ```

  a copy from /t*/src/* to /temp would result in the Copy utility trying to write both a.txt files to the same location in the temp directory.

There is no recovery for copies. Therefore, if the transfer of a large file fails, the entire file must be transferred again. Likewise, if a multi-file transfer fails before completion, you must either re-run the entire transfer or request only those parts that did not transfer.

Explicit machine naming

Keep in mind that when you are using the Copy utility, you are potentially working with three machines: the EAC Central Server, from which you issue eaccmd commands, the Agent machine you are copying data from, and the one you are copying data to. In such cases, the name localhost can be confusing. Unless you are using the Copy utility to move files on a single machine, you should use explicit machine names rather than simply localhost.

Delimiting wildcard elements

To prevent inappropriate expansion, any wildcard arguments you use with the Copy utility in eaccmd need to be delimited with double quotation marks. For example:

On Windows, "C:*.txt".

On UNIX, "/home/endeca/test/*.txt".

Copying across platforms

If you are copying files or directories between machines on different platforms, you have to wrap any Window paths on a Linux or Solaris shell in double quotation marks (for example, "C:*.txt").

The Archive utility

The Archive utility allows you to archive and roll back directories.

Using the Archive utility, you can save off and back up a set of component outputs, which later can be rolled back on demand. With the backup operation, you create back up copies of directories distinguished by time stamps. With the rollback operation, you replace the current version of a directory with the most recently backed-up version. The current version is then renamed with an .unwanted suffix.

Note: Do not start a backup or rollback operation while another such operation is in progress on the same directory. Unexpected behavior may occur if you do so.

Backup operations

Backup operations create an archive directory from an existing directory.

Backup operations create an archive directory from an existing directory. The archive directory has the same name as the original directory, but with a timestamp appended to the end. The timestamp reflects the time when the backup operation was performed.

For example, if the original directory is called logs and was backed up on October 11, 2008 at 8:00 AM, the backup operation creates a directory called logs.2008_10_11.08_00_00.

Command	Description
start-util --type backup --app app_id [--token token] --host host-id --dir dir [--method] <copy|move> [--backups num_backups]	Starts the backup operation. The token is a string. If you do not specify a token, one is generated and returned when you start the utility. The token is used to stop the utility or to get its status. The host and dir settings specify the path to the directory that will be archived. The method is either copy or move (the default). The optional backups setting specifies the maximum number of archives to store. This number does not include the original directory itself, so if backups is set to 3, you would have the original directory plus up to three archive directories, for a total of as many as four directories. The default num_backups is 5.
stop-util --app app_id --token token	Stops a backup operation. The token is a string, either user-created or system-generated when you start the utility. The token can be used to stop the utility or to get its status.
status-util --app app_id --token token	Gets the status of a backup operation. The token is a string, either user-created or system-generated when you start the utility. The token can be used to stop the utility or to get its status.

Backup operation example

In the following example, an archive version of the logs directory is created.

```
eaccmd start-util --type backup --app my_wine --host mkt1010
--dir c:\my_wine\data\logs --backups 2
```

Rollback operations

Rollback operations roll back the directory to the most recent backed up version.

For example, say you have a directory called logs, one called logs.2008_10_11.08_00_00, and other, older versions. When you roll back, the following things happen:

- logs is renamed logs.unwanted.
- logs.2008_10_11.08_00_00 is renamed logs.
- The older versions are left alone.

Note: There can only be a single .unwanted directory at a time. If you roll back twice, the .unwanted directory from the first rollback is deleted.

Command	Description
start-util --type rollback --app app_id [--token token] --host host_id --dir dir	Starts the rollback operation. The token is a string. If you do not specify a token, one is generated and returned when you start the utility. The token is used to stop the utility or to get its status. The host and dir settings specify the path to the directory that will be rolled back.
stop-util --app app_id --token token	Stops a rollback operation. The token is a string, either user-created or generated and returned when you start the utility, that eaccmd prints to screen. The token can be used to stop the utility or to get its status.
status-util --app app_id --token token	Gets the status of a rollback operation. The token is a string, either user-created or generated and returned when you start the utility, that eaccmd prints to screen. The token can be used to stop the utility or to get its status.

Rollback operation example

In the following example, the archived logs directory is rolled back.

```
eaccmd start-util --type rollback --app my_wine --host mkt1010
--dir c:\my_wine\data\logs
```

Appendix F

The Endeca Application Controller Development Toolkit

The EAC Development Toolkit provides a common set of objects, a standard and robust configuration file format and a lightweight controller implementation that developers can leverage in order to implement operational controller applications. The toolkit is designed to enable quick deployment, while providing complete flexibility for developers to extend and override any part of the implementation to create custom, project-specific functionality.

EAC Development Toolkit distribution and package contents

The EAC Development Toolkit is distributed as a set of JAR files bundled with the Deployment Template.

The toolkit consists of three JAR files has dependencies on two others, all of which are distributed with Tools and Frameworks. Details about classes and methods can be found in Javadoc distributed with the EAC Development Toolkit. These JAR files must be on the classpath of any application built using the EAC Development Toolkit.

`eacToolkit.jar`

This JAR contains the compiled class files for the core EAC Development Toolkit classes. These classes encompass core EAC functionality, from which all component implementations extend. Included are low-level classes that access the EAC's central server via SOAP calls to its Web Service interface as well as higher level objects that wrap logic and data associated with hosts, components, scripts and utilities. In addition, this JAR includes the controller implementation used to load the Toolkit's application configuration file, and to invoke actions based on the configuration and the user's command line input.

`eacComponents.jar`

This JAR contains the compiled class files for common implementations of Guided Search components. These classes extend core functionality in `eacToolkit.jar` and implement standard versions of Forge, Dgidx, Dgraph and other components of an Guided Search deployment.

`eacHandlers.jar`

This JAR contains the compiled class files for parsing application configuration documents. In addition, the EAC Dev Toolkit's application configuration XML document format is defined by an XSD file packaged with this JAR. Finally, the JAR includes files required to register the schema and the toolkit's namespace with Spring, the framework used to load the toolkit's configuration.

spring.jar

The toolkit uses the Spring framework for configuration management.

bsh-2.0b4.jar

The toolkit uses BeanShell as the scripting language used by developers to write scripts in their application configuration documents.

EAC Development Toolkit usage

The EAC Development Toolkit provides a library of classes that developers can use to develop and configure EAC scripts.

Classes in the library expose low level access to the EAC's web services and implement high level functionality common to many EAC scripts. Developers may implement applications by simply configuring functionality built in the toolkit or by extending the toolkit at any point to develop custom functionality.

This document discusses the toolkit's configuration file format, BeanShell scripting, command invocation and logging. This document does not provide a reference of the classes in the toolkit, or the functionality implemented in various objects and methods. Developers should refer to Javadoc in the `ToolsAndFrameworks\<version>\deployment_template\apidoc` directory for details about the implementation.

Application Configuration File

The EAC toolkit uses an XML configuration file to define the elements that make up an application. In most deployments, this document will serve as the primary interface for developers and system administrators to configure, customize, and maintain a deployed application.

The toolkit's controller allows multiple configuration files to be specified, enabling users to separate configuration and scripts into various files. All of the objects defined across all configuration files are loaded by the controller and made accessible to scripts.

Spring framework

The EAC Development Toolkit uses the Spring Framework's Inversion of Control container to load an EAC application based on configuration specified in an XML document.

A great deal of functionality and flexibility is provided in Spring's IoC Container and in the default bean definition XML file handled by Spring's `XmlBeanDefinitionReader` class. For details about either of these, refer to Spring Framework documentation and JavaDoc.

The EAC Development Toolkit uses a customized document format and includes a schema and custom XML handlers to parse the custom document format. It uses Spring to convert this customized configuration metadata into a system ready for execution. Specifically, the toolkit uses Spring to load a set of objects that represent an EAC application with the configuration specified for each object in the configuration document.

XML schema

A customized document format is used to provide an intuitive configuration format for EAC script developers and system administrators.

However, this customization restricts the flexibility of the configuration document. The following sections describe elements available in the custom namespace defined by the `eacToolkit.xsd` XML schema. Each element name is followed by a brief description and an example configuration excerpt. For details, refer to the `eacToolkit.xsd` schema file distributed within the file `eacHandlers.jar`.

Application elements

This section describes the application elements available in the custom namespace defined by the `eacToolkit.xsd` XML schema.

For more details, refer to the `eacToolkit.xsd` schema file distributed within the file `eacHandlers.jar`.

Element	Description
`app`	This element defines the global application settings inherited by all other objects in the document, including application name, EAC central server host and port, data file prefix, the lock manager used by the application and whether or not SSL is enabled. In addition, this object defines global defaults for the working directory and the logs directory, which can be inherited or overridden by objects in the document. `<app appName="myApp" eacHost="devhost.company.com" eacPort="8888" dataPrefix="myApp" sslEnabled="false" lockManager="LockManager" > <working-dir>C:\Endeca\apps\myApp</working-dir> <log-dir>./logs/baseline</log-dir> </app>`
`lock-manager`	This element defines a LockManager object used by the application to interact with the EAC's synchronization web service. Lock managers can be configured to release locks when a failure is encountered, ensuring that the system returns to a "neutral" state if a script or component fails. Multiple lock managers can be defined. `<lock-manager id="LockManager" releaseLocksOnFailure="true" />`

Hosts

This section describes the host element available in the custom namespace defined by the `eacToolkit.xsd` XML schema.

The `host` element defines a host associated with the application, including the `ID`, `hostname` and EAC agent port of the host. Multiple host elements can be defined.

```
<host id="ITLHost" hostName="itlhost.company.com" port="8888" />
```

Components

This section describes the component elements available in the custom namespace defined by the `eacToolkit.xsd` XML schema.

For more details, refer to the `eacToolkit.xsd` schema file distributed within the file `eacHandlers.jar`.

Element	Description

`forge`	This element defines a Forge component, including attributes that define the functionality of the Forge process as well as custom properties and directories used to configure the functionality of the Forge object's methods. Multiple `forge` elements can be defined.
	`<forge id="Forge" host-id="ITLHost">`
	`  <properties>`
	`  <property name="numStateBackups" value="10" />`
	`  <property name="numLogBackups" value="10" />`
	`  </properties>`
	`  <directories>`
	`  <directory name="incomingDataDir">./data/incoming</directo¬`
	`ry>`
	`  <directory name="configDir">./data/complete_config</direc¬`
	`tory>`
	`  <directory name="wsTempDir">./data/web_studio_temp_dir</di¬`
	`rectory>`
	`  </directories>`
	`  <args>`
	`  <arg>-vw</arg>`
	`  </args>`
	`  <input-dir>./data/processing</input-dir>`
	`  <output-dir>./data/forge_output</output-dir>`
	`  <state-dir>./data/state</state-dir>`
	`  <temp-dir>./data/temp</temp-dir>`
	`  <num-partitions>1</num-partitions>`
	`  <pipeline-file>./data/processing/pipeline.epx</pipeline-file>`
	`</forge>`

`forge-cluster`	This element defines a Forge cluster, including a list of ID references to the Forge components that belong to this cluster. This object can be configured to distribute data to Forge servers serially or in parallel.
	`<forge-cluster id="ForgeCluster" getDataInParallel="true">`
	`  <forge ref="ForgeServer" />`
	`  <forge ref="ForgeClient1" />`
	`  <forge ref="ForgeClient2" />`
	`</forge-cluster>`

`dgidx`	This element defines a Dgidx component, including attributes that define the functionality of the Dgidx process as well as custom properties and directories used to configure the functionality of the Dgidx object's methods. Multiple `dgidx` elements can be defined.
	`<dgidx id="Dgidx" host-id="ITLHost">`
	`  <args>`
	`  <arg>-v</arg>`
	`  </args>`
	`  <input-dir>./data/forge_output</input-dir>`
	`  <output-dir>./data/dgidx_output</output-dir>`
	`  <temp-dir>./data/temp</temp-dir>`
	`  <run-aspell>true</run-aspell>`
	`</dgidx>`

Element	Description

indexing-cluster	This element defines an indexing cluster, including a list of ID references to the Dgidx components that belong to this cluster. This object can be configured to distribute data to indexing servers serially or in parallel.
	`<indexing-cluster id="IndexingCluster" getDataInParallel="true">`
	`  <dgidx ref="Dgidx1" />`
	`  <dgidx ref="Dgidx2" />`
	`</indexing-cluster>`

dgraph	This element defines a Dgraph component, including attributes that define the functionality of the Dgraph process as well as custom properties and directories used to configure the functionality of the Dgraph object's methods. Multiple `dgraph` elements can be defined. Each `dgraph` element inherits, and potentially overrides, configuration specified in the `dgraph-defaults` element (see below).
	`<dgraph id="Dgraph1" host-id="MDEXHost" port="15000">`
	`  <properties>`
	`  <property name="restartGroup" value="A" />`
	`  <property name="updateGroup" value="a" />`
	`  </properties>`
	`  <log-dir>./logs/dgraphs/Dgraph1</log-dir>`
	`  <input-dir>./data/dgraphs/Dgraph1/dgraph_input</input-dir>`
	`  <update-dir>./data/dgraphs/Dgraph1/dgraph_input/updates</up¬`
	`date-dir>`
	`  </dgraph>`

dgraph-defaults	This element defines the default settings inherited by all `dgraph` elements specified in the document. This enables a single point of configuration for common Dgraph configuration such as command line arguments, and script directory configuration. Only one `dgraph-defaults` element can be defined.
	`<dgraph-defaults>`
	`  <properties>`
	`  <property name="srcIndexDir" value="./data/dgidx_output"`
	`/>`
	`  <property name="srcIndexHostId" value="ITLHost" />`
	`  <property name="numLogBackups" value="10" />`
	`  </properties>`
	`  <directories>`
	`  <directory name="localIndexDir">`
	`  ./data/dgraphs/local_dgraph_input`
	`  </directory>`
	`  </directories>`
	`  <args>`
	`  <arg>--threads</arg>`
	`  <arg>2</arg>`
	`  <arg>--spl</arg>`
	`  <arg>--dym</arg>`
	`  </args>`
	`  <startup-timeout>120</startup-timeout>`
	`</dgraph-defaults>`

Element	Description

dgraph-cluster	This element defines a Dgraph cluster, including a list of ID references to the Dgraph components that belong to this cluster. This object can be configured to distribute data to Dgraph servers serially or in parallel.
	`<dgraph-cluster id="DgraphCluster" getDataInParallel="true">`
	`  <dgraph ref="Dgraph1" />`
	`  <dgraph ref="Dgraph2" />`
	`</dgraph-cluster>`

logserver	This element defines a LogServer component, including attributes that define the functionality of the LogServer process as well as custom properties and directories used to configure the functionality of the LogServer object's methods. Multiple `logserver` elements can be defined.
	`<logserver id="LogServer" host-id="ITLHost" port="15002">`
	`  <properties>`
	`  <property name="numLogBackups" value="10" />`
	`  <property name="targetReportGenDir" value="./reports/input" />`
	`  <property name="targetReportGenHostId" value="ITLHost" />`
	`  </properties>`
	`  <log-dir>./logs/logserver</log-dir>`
	`  <output-dir>./logs/logserver_output</output-dir>`
	`  <startup-timeout>120</startup-timeout>`
	`  <gzip>false</gzip>`
	`</logserver>`

report-generator	This element defines a ReportGenerator component, including attributes that define the functionality of the ReportGenerator process as well as custom properties and directories used to configure the functionality of the ReportGenerator object's methods. Multiple `report-generator` elements can be defined.
	`<report-generator id="WeeklyReportGenerator" host-id="ITLHost">`
	`  <properties>`
	`  <property name="webStudioReportDir" value="C:\Endeca\MDEXEngine\workspace/reports/MyApp" />`
	`  <property name="webStudioReportHostId" value="ITLHost" />`
	`  </properties>`
	`  <log-dir>./logs/report_generators/WeeklyReportGenerator</log-dir>`
	`  <input-dir>./reports/input</input-dir>`
	`  <output-file>./reports/weekly/report.xml</output-file>`
	`  <stylesheet-file>`
	`  ./config/report_templates/tools_report_stylesheet.xsl`
	`  </stylesheet-file>`
	`  <settings-file>`
	`  ./config/report_templates/report_settings.xml`
	`  </settings-file>`
	`  <time-range>LastWeek</time-range>`
	`  <time-series>Daily</time-series>`
	`  <charts-enabled>true</charts-enabled>`
	`</report-generator>`

Element	Description

custom-component	This element defines a custom component, including custom properties and directories used to configure the functionality of the custom component object's methods. Multiple custom-component elements can be defined, though each must specify the name of the implemented class that extends com.Endeca.soleng.eac.toolkit.component.CustomComponent.
	The custom component is also used to implement the Configuration Manager, Workbench Manager, and IFCR components.
	`<custom-component id="IFCR" host-id="ITLHost" class="com.ende¬ca.soleng.eac.toolkit.component.IFCRComponent">`
	`  <properties>`
	`  <property name="repositoryUrl" value="http://local¬host:8006/ifcr" />`
	`  <property name="username" value="admin" />`
	`  <property name="password" value="admin" />`
	`  <property name="numExportBackups" value="3" />`
	`  </properties>`
	`  </custom-component>`

Utilities

This section describes the utility elements available in the custom namespace defined by the eacToolkit.xsd XML schema.

For more details, refer to the eacToolkit.xsd schema file distributed within the file eacHandlers.jar.

Element	Description

copy	This element defines a copy utility invocation, including the source and destination and whether or not the source pattern should be interpreted recursively. Multiple copy elements can be defined.
	`<copy id="CopyData" src-host-id="ITLHost" dest-host-id="ITLHost"`
	`  recursive="true" >`
	`  <src>./data/incoming/*.txt</src>`
	`  <dest>./data/processing/</dest>`
	`</copy>`

shell	This element defines a shell utility invocation, including the command to execute and the host on which the command will be executed. Multiple shell elements can be defined.
	`<shell id="ProcessData" host-id="ITLHost" >`
	`  <command>perl procesDataFiles.pl ./data/incoming/data.txt</command>`
	`</shell>`

backup	This element defines a backup utility invocation, including the directory to archive, how many archives should be saved and whether the archive should copy or move the source directory. Multiple backup elements can be defined.
	`<backup id="ArchiveState" host-id="ITLHost" move="true" num-back¬ups="5">`
	`  <dir>C:\Endeca\apps\myApp\data\state</dir>`
	`</backup>`

Element	Description
rollback	This element defines a rollback utility invocation, including the directory whose archive should be recovered. Multiple rollback elements can be defined. <rollback id="RollbackState" host-id="ITLHost"> <dir>./data/state</dir> </rollback>

Customization/extension within the toolkit's schema

Most configuration tasks are performed by simply altering an element in the configuration document, by adding elements to the document, or by removing elements from the configuration.

These three actions enable users to alter the behavior of objects in their application, change which objects make up their application and change the way scripts acts on the objects in their application.

In addition to these simple actions, users can customize the behavior of objects in their application or create new objects while continuing to use the EAC development toolkit's XML configuration document format. The following are examples of customization that are possible within the constructs of the XML schema defined in the eacToolkit.xsd schema file.

Implement a custom component

Users can develop new custom components by extending the class com.Endeca.soleng.eac.toolkit.component.CustomComponent. This class and its associated XML element allow any number of properties and directories to be specified and accessed by methods in the object. This customization method may be appropriate for cases where functionality needs to be developed that is not directly associated with an Oracle Endeca process.

Extend an existing object

Users can implement customizations on top of existing objects by creating a new class that extends an object in the toolkit. Most elements in the configuration document (with the notable exception of the "app" element, which specifies global configuration, but does not directly correspond to an object instance) can specify a class attribute to override the default class associated with each element. For example, a user could implement a MyForgeComponent class by extending the toolkit's ForgeComponent class.

```
package com.Endeca.soleng.eac.toolkit.component;

import java.util.logging.Logger;

import com.Endeca.soleng.eac.toolkit.exception.AppConfigurationException;
import com.Endeca.soleng.eac.toolkit.exception.EacCommunicationException;
import com.Endeca.soleng.eac.toolkit.exception.EacComponentControlException;

public class MyForgeComponent extends ForgeComponent
{
  private static Logger log =
    Logger.getLogger(MyForgeComponent.class.getName());

  protected void getIncomingData() throws AppConfigurationException,
      EacCommunicationException, EacComponentControlException,
      InterruptedException
  {

  // custom data retrieval implementation
```

```
  }
}
```

The new class can override method functionality to customize the behavior of the object. As long as the new object does not require configuration elements unknown to the `ForgeComponent` from which it inherits, it can continue to use the forge element in the XML document to specify object configuration.

```
<forge class="com.Endeca.soleng.eac.toolkit.component.MyForgeComponent"
  id="CustomForge" host-id="ITLHost">
...
</forge>
```

Implement custom functionality in BeanShell scripts

Users can implement custom functionality by writing new code in the XML document in new or existing BeanShell scripts. This form of customization can be used to add new functionality or to override functionality that is built in to toolkit objects. While this customization approach is very flexible, it can become unwieldy and hard to maintain and debug if a large amount of custom code needs to be written.

Customization/extension beyond the toolkit's schema

Customization approaches within the existing schema will be sufficient for the majority of applications, but some developers will require even greater flexibility than can be supported by the XML document exposed by the toolkit.

This type of customization can still be achieved, by switching out of the default `eacToolkit` namespace in the XML document and leveraging the highly flexible and extensible Spring Framework bean definition format.

As an example, a developer might implement a new class, `PlainOldJavaObject`, which needs to be loaded and accessed by EAC scripts. If the object is implemented, compiled and added to the classpath, it can be loaded based on configuration in the XML document by specifying its configuration using the "`spr`" namespace.

```
<spr:bean id="MyPOJO" class="com.company.PlainOldJavaObject">
  <spr:constructor-arg>true</spr:constructor-arg>
  <spr:property name="Field1" value="StrValue" />
  <spr:property name="Map1">
    <spr:map>
      <spr:key>one</spr:key>
      <spr:value>1</spr:value>
      <spr:key>two</spr:key>
      <spr:value>2</spr:value>
    </spr:map>
  </spr:property>
</spr:bean>
```

BeanShell Scripting

The EAC Development Toolkit uses BeanShell to interpret and execute scripts defined in the app configuration document. The following sections describe the toolkit's use of the BeanShell interpreter and provide sample BeanShell script excerpts.

Script implementation

In the toolkit, the `com.Endeca.soleng.eac.toolkit.script.Script` class implements scripts.

This class exposes simple execution logic that either uses a BeanShell interpreter to execute the script specified in the configuration file or, if no BeanShell script is specified in the script's configuration, uses the Script object's `scriptImplementation` method. By default, the `scriptImplementation` method has no logic and must be overridden by an extending class to take any action. This allows developers to leverage BeanShell to implement their scripts or to extend the Script object, overriding and implementing the `scriptImplementation` method.

By implementing scripts as BeanShell scripts configured in the toolkit's XML configuration document, developers can quickly develop and adjust scripts, and system administrators can adjust script implementations without involving developers. The scripting language should be familiar to any Java developer, as it is a Java based scripting language that can interpret strict Java code (i.e. code that could be compiled as a Java class). BeanShell also provides a few flexibilities that are not available in Java; for example, BeanShell allows developers to import classes at any point in the script, rather than requiring all imports to be defined up front. In addition, BeanShell allows variables to be declared without type specification.

Note: For details about BeanShell and ways in which it differs from Java, developers should refer to BeanShell documentation and Javadoc.

BeanShell interpreter environment

The most common use of BeanShell scripts in the EAC Development Toolkit is to orchestrate the elements defined in the application configuration document.

More precisely, BeanShell scripts are used to orchestrate the execution of methods on the objects that are loaded from the configuration document. In order to enable this, when the toolkit constructs the BeanShell Interpreter environment, it sets internal variables associated with each element defined in the configuration document. While additional variables can be declared at any point in a script, this allows scripts to immediately act on objects defined in the document without declaring any variables.

Take, for example, the following configuration document:

```
<app appName="myApp" eacHost="devhost.company.com" eacPort="8888"
    dataPrefix="myApp" sslEnabled="false" lockManager="LockManager" >
  <working-dir>C:\Endeca\apps\myApp</working-dir>
  <log-dir>./logs/baseline</log-dir>
</app>

<host id="ITLHost" hostName="itlhost.company.com" port="8888" />

<copy id="CopyData" src-host-id="ITLHost" dest-host-id="ITLHost"
    recursive="true" >
  <src>./data/incoming/*.txt</src>
  <dest>./data/processing/</dest>
</copy>

<backup id="ArchiveState" host-id="ITLHost" move="true" num-backups="5">
  <dir>C:\Endeca\apps\myApp\data\state</dir>
</backup>

<forge id="Forge" host-id="ITLHost">
  <properties>
    <property name="numStateBackups" value="10" />
    <property name="numLogBackups" value="10" />
  </properties>
  <directories>
    <directory name="incomingDataDir">./data/incoming</directory>
    <directory name="configDir">./data/processing</directory>
```

```
  </directories>
  <args>
    <arg>-vw</arg>
  </args>
  <input-dir>./data/processing</input-dir>
  <output-dir>./data/forge_output</output-dir>
  <state-dir>./data/state</state-dir>
  <temp-dir>./data/temp</temp-dir>
  <num-partitions>1</num-partitions>
  <pipeline-file>./data/processing/pipeline.epx</pipeline-file>
</forge>
```

A BeanShell script defined in this document will have five variables immediately available for use: `ITLHost`, `CopyData`, `ArchiveState`, `Forge`, and `log`. Note that there is no variable associated with the `app` element in the document, as this element does not correspond to an object instance. Each of the other elements is instantiated, loaded with data based on its configuration and made available in the BeanShell interpreter. In addition, a special variable called log is always created for each script with a `java.util.Logger` instance.

A simple BeanShell script can then be written without importing a single class or instantiating a single variable.

```
<script id="SimpleForgeScript">
  <bean-shell-script>
    <![CDATA[
  log.info("Starting Forge script.");
  CopyData.run();
  Forge.run();
  ArchiveState.setNumBackups(Forge.getProperty("numStateBackups"));
  ArchiveState.run();
  log.info("Finished Forge script.");
    ]]>
  </bean-shell-script>
</script>
```

In addition to exposing objects defined in the document, the toolkit imports and executes a default script each time a BeanShell script is invoked. If a file named "`beanshell.imports`" is successfully loaded as a classpath resource, that file is executed each time a BeanShell script is executed. This allows a default set of imports to be defined. For example, the following default file imports all of the classes in the toolkit, exposing them to BeanShell scripts:

```
import com.Endeca.soleng.eac.toolkit.*;
import com.Endeca.soleng.eac.toolkit.application.*;
import com.Endeca.soleng.eac.toolkit.base.*;
import com.Endeca.soleng.eac.toolkit.component.*;
import com.Endeca.soleng.eac.toolkit.component.cluster.*;
import com.Endeca.soleng.eac.toolkit.exception.*;
import com.Endeca.soleng.eac.toolkit.host.*;
import com.Endeca.soleng.eac.toolkit.logging.*;
import com.Endeca.soleng.eac.toolkit.script.*;
import com.Endeca.soleng.eac.toolkit.utility.*;
import com.Endeca.soleng.eac.toolkit.utility.perl.*;
import com.Endeca.soleng.eac.toolkit.utility.webstudio.*;
import com.Endeca.soleng.eac.toolkit.utility.wget.*;
import com.Endeca.soleng.eac.toolkit.utils.*;
```

About implementing logic in BeanShell

BeanShell scripts will typically be used to orchestrate method execution for objects defined in the configuration document.

However, scripts can also implement logic, instantiating objects to provide a simple point of extension for developers to implement new logic without compiling additional Java classes.

For example, the following script excerpt demonstrates how a method can be defined and referenced in a script:

```
<script id="Status">
    <bean-shell-script>
      <![CDATA[

        // define function for printing component status
        import com.Endeca.soleng.eac.toolkit.component.Component;
        void printStatus( Component component ) {
          log.info(component.getAppName() + "." +
            component.getElementId() + ": " +
            component.getStatus().toString() );
        }

        // print status of forge, dgidx, logserver
        printStatus( Forge );
        printStatus( Dgidx );
        printStatus( LogServer );

        // print status for dgraph cluster
        dgraphs = DgraphCluster.getDgraphs().iterator();
        while( dgraphs.hasNext() ) {
          printStatus( dgraphs.next() );
        }

      ]]>
    </bean-shell-script>
  </script>
```

Command Invocation

The toolkit provides a simple interface for invoking commands from the command line.

The class `com.Endeca.soleng.eac.toolkit.Controller` exposes a main method that can be executed from the command line.

This method can be used to invoke a method on any object defined in the application configuration document. The Controller's usage information describes the command line arguments.

```
USAGE:
java Controller [options] --app-config <app config> <object> <method> [args]

The controller will invoke a method on an object defined in the app
configuration document. Method return values are not captured and
only String arguments may be passed as method parameters. The controller
will typically be used to run scripts, components or utilities. More
complex invocations should usually be wrapped in a BeanShell script.

By default, the controller will compare provisioning in the app config
document to the provisioning in the EAC. If any definition changes are
found, elements are re-provisioned.

  Available options:
    --help
        Displays this usage information. If app config
```

```
       document and object name specified, available
       methods will be displayed.
   --update-definition
       Updates application provisioning without invoking
       any action. Any specified object, method and args
       will be ignored.
   --skip-definition
       Skips the default provisioning check, invoking the
       requested action with the app definition currently
       provisioned in the EAC.
   --remove-app
       Removes the application from the EAC.
       WARNING: Any active components will be stopped.
   --print-status
       Displays the status of application components.
   --config-override <override props file>
       Name of an app configuration override properties
       file to read from the classpath. Multiple override
       files may be specified.

 <app config>
     Name of the app configuration document to read
     from the classpath. Multiple documents may be specified.
 <object>
     ID of object defined in app config document.
 <method>
     Method to invoke on the specified object. Default: run.
 [args]
     Arguments to pass to the specified method. Only String
     arguments are allowed. Methods requiring other
     argument types may be wrapped in BeanShell script.
```

Invoke a method on an object

By default, the controller tries to invoke a method called "`run`" with no arguments on the specified object.

The following simple command invokes the run method on the BaselineUpdate script object:

```
java Controller --app-config AppConfig.xml BaselineUpdate
```

If a method name is specified, the controller looks for a method with that name on the specified object and invokes it. For example, the following command executes the `applyIndex` method on the `DgraphCluster` object:

```
java Controller --app-config AppConfig.xml DgraphCluster applyIndex
```

In addition to no-argument method invocation, the controller allows any number of String arguments to be passed to a method. The following example shows the `releaseLock` method being invoked on the LockManager object with the single String argument "`update_lock`" specifying the name of the lock to release:

```
java Controller --app-config AppConfig.xml LockManager releaseLock
  update_lock
```

Identify available methods

In order to help users identify the objects and methods available for invocation, the controller provides a help argument that can be called to list all available objects or methods available on an object.

If specified with an app configuration document, the help command displays usage and available objects:

```
java Controller --app-config AppConfig.xml --help

...

The following objects are defined in document 'AppConfig.xml':
[To see methods available for an object, use the --help command line argument and
 specify the name of the object.]

  [com.Endeca.soleng.eac.toolkit.base.LockManager]
    LockManager
  [com.Endeca.soleng.eac.toolkit.component.ConfigManagerComponent]
    ConfigManager
  [com.Endeca.soleng.eac.toolkit.component.DgidxComponent]
    Dgidx
  [com.Endeca.soleng.eac.toolkit.component.DgraphComponent]
    Dgraph1
    Dgraph2
  [com.Endeca.soleng.eac.toolkit.component.ForgeComponent]
    Forge
    PartialForge
  [com.Endeca.soleng.eac.toolkit.component.LogServerComponent]
    LogServer
  [com.Endeca.soleng.eac.toolkit.component.ReportGeneratorComponent]
    WeeklyReportGenerator
    DailyReportGenerator
  [com.Endeca.soleng.eac.toolkit.component.cluster.DgraphCluster]
    DgraphCluster
  [com.Endeca.soleng.eac.toolkit.host.Host]
    ITLHost
    MDEXHost
  [com.Endeca.soleng.eac.toolkit.script.Script]
    BaselineUpdate
    DistributeIndexAndApply
    PartialUpdate
    DistributePartialsAndApply
    ConfigUpdate
```

The name of each object loaded from the configuration document is printed along with the object's class. To identify the available methods, the help command can be invoked again with the name of an object in the document:

```
java Controller --app-config AppConfig.xml --help DgraphCluster
...

The following methods are available for object 'DgraphCluster':
[Excluded: private, static and abstract methods; methods inherited from Object;
methods with names that start with 'get', 'set' or 'is'. For details, refer to
Javadoc for class com.Endeca.soleng.eac.toolkit.component.cluster.DgraphCluster.]


  start(), stop(), removeDefinition(), updateDefinition(), cleanDirs(),
  applyIndex(), applyPartialUpdates(), applyConfigUpdate(),
  cleanLocalIndexDirs(), cleanLocalPartialsDirs(),
  cleanLocalDgraphConfigDirs(), copyIndexToDgraphServers(),
  copyPartialUpdateToDgraphServers(),
  copyCumulativePartialUpdatesToDgraphServers(),
  copyDgraphConfigToDgraphServers(),  addDgraph(DgraphComponent)
```

Note that not all methods defined for the class `com.Endeca.soleng.eac.toolkit.component.clus¬ter.DgraphCluster` are displayed. As the displayed message notes, methods declared as private, static

or abstract are excluded, as are methods inherited from Object, getters and setters, and a few reserved methods that are known not to be useful from the command line. These restrictions are intended to make the output of this help command as useful as possible, but there are likely to be cases when developers will need to refer to Javadoc to find methods that are not displayed using the help command.

Update application definition

By default, the controller will test the application definition in the configuration document against the provisioned definition in the EAC and update EAC provisioning if the definition in the document has changed.

This will happen by default any time any method is invoked on the command line.

System administrators may find it useful to update the definition without invoking a method. To facilitate this, a flag has been provided to perform the described definition update and exit.

```
java Controller --app-config AppConfig.xml --update-definition
```

In addition, there may be a need to invoke a method without testing the application definition. This can be accomplished by using an alternate command line argument:

```
java Controller --app-config AppConfig.xml --skip-definition
  BaselineUpdate
```

Remove an application

The controller provides a convenience method for removing an application from the EAC's central store.

When invoked, this action checks whether the application loaded from the configuration document is defined in the EAC. If it is, all active components are forced to stop and the application's definition is completely removed from the EAC.

```
java Controller --remove-app --app-config AppConfig.xml
```

Display component status

The controller provides a convenience method for displaying the status of all components defined in the configuration document.

When the following method is invoked, the controller iterates over all defined components, querying the EAC for the status of each one and printing it.

```
java Controller --print-status --app-config AppConfig.xml
```

Appendix G

Endeca Application Controller API Reference

This section provides an overview of the interfaces and classes in the Endeca Application Controller API.

Using the Application Controller WSDL

You can use the Endeca Application Controller WSDL API to write your application in the language of your choice.

Using the Web Services tool of your choice (such as Axis for Java), do the following:

1. Run the WSDL through your tool to generate the stubs (that is, an API that your code can call).
2. Write your application, using that code to control the Application Controller.

Note:

- The Application Controller schema is defined in `eac.wsdl`, which is located in the `%ENDECA_ROOT%\lib\services` directory.
- You generate clent stubs (or proxies) using the `eac.wsdl` file located in the file system provided by the Guided Search installation. You cannot generate client stubs using the SOAP Web services addresses associated with each service within the WSDL file.

Simple types in the Application Controller WSDL

The Application Controller WSDL defines several data types that can be treated as simple data types.

- `IDType`, `TokenType`, `BackupMethodType`, `TimeRangeType`, and `TimeSeriesType` can be treated as Strings.
- `PortNumber` can be treated as an Integer.
- `TimeOut` can be treated as a Long.

Interface Reference

This section documents Endeca Application Controller interfaces.

The exact syntax of a class member depends on the output of the WSDL tool that you are using. Be sure to check the client stub classes that are generated by your WSDL tool for the exact syntax of the Application Controller API class members.

ComponentControl interface

The ComponentControl interface provides component management capabilities.

It consists of the following methods:

startComponent(FullyQualifiedComponentIDType startComponentInput)

Starts the named component.

FullyQualifiedComponentIDType parameters:

- applicationID identifies the application to use.
- componentID identifies the component to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

stopComponent(FullyQualifiedComponentIDType stopComponentInput)

Stops the named component.

FullyQualifiedComponentIDType parameters:

- applicationID identifies the application to use.
- componentID identifies the component to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Synchronization interface

The Synchronization interface manages application-level flags that let users know when processes are in use.

For example, your code could create a flag named update-running to ensure that a new baseline update does not start while another update is already in progress.

Typical usage is as follows:

```
if (setFlag(MY_FLAG_ID) == true)
 [perform action, such as a baseline update]
 removeFlag(MY_FLAG_ID)
else
 [signal error such as "an update is already in progress"]
```

setFlag(FullyQualifiedFlagIDType setFlagInput)

Creates a new flag, identified by flagID, that is associated with the named application.

FullyQualifiedFlagIDType parameters:

- applicationID identifies the application to use.
- flagID is a unique string identifier for this flag.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- A Boolean, false if the flag was already set, or true if it was not set meaning the method succeeded).

removeFlag(FullyQualifiedFlagIDType removeFlagInput)

Removes the named flag.

FullyQualifiedFlagIDType parameters:

- applicationID identifies the application to use.
- flagID is a unique string identifier for this flag.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

removeAllFlags(IDType removeAllFlagsInput)

Removes all flags in an application.

IDType parameter:

- applicationID identifies the application to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

listFlags(IDType listFlagsInput)

Lists the collection of flags in an application.

IDType parameter:

- applicationID identifies the application to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- flagIDList, a string collection of flagIDs.

Utility interface

The Utility interface allows you to manage the Application Controller utilities (Shell, Copy, and Archive) programmatically.

Note: Be sure to name your utilities carefully. If you create a new utility that has the same name as a running utility, an error is issued. However, if there is an existing utility with the same name that is not running, the new utility overwrites it.

The Utility interface consists of the following methods:

startBackup(RunBackupType startBackupInput)

Starts the backup operation of the Archive utility.

Backup operations create an archive directory from an existing directory. The archive directory has the same name as the original directory, but with a timestamp appended to the end. The timestamp reflects the time when the backup operation was performed.

For example, if the original directory is called logs and was backed up on October 11, 2008 at 8:00 AM, the backup operation creates a directory called logs.2008_10_11.08_00_00.

Note: Do not start a backup or rollback operation while another such operation is in progress on the same directory. Unexpected behavior may occur if you do so.

RunBackupType parameters:

- applicationID identifies the application to use.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- hostID is a unique identifier for the host. The hostID and dirName parameters specify the path to the directory that will be archived.
- dirName is the full path of the directory. The hostID and dirName parameters specify the path to the directory that will be archived.
- backupMethod is either copy or move.
- numBackups specifies the maximum number of archives to store. This number does not include the original directory itself, so if numBackups is set to 3, you would have the original directory plus up to three archive directories, for a total of as many as four directories. The default numBackups is 5.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- The string token assigned to this invocation.

startFileCopy(RunFileCopyType startFileCopyInput)

Launches the Copy utility, which copies files either on a single machine or between machines.

RunFileCopyType parameters:

- applicationID identifies the application to use.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- fromHostID is a unique identifier for the host from which you are copying.
- toHostID is a unique identifier for the host to which you are copying.
- sourcePath is a string representing the file, wildcard, or directory to be copied. A sourcePath must start with an absolute path, such as C:\ or /. A sourcePath can contain . or .. as directory names, and expands * and ? wildcards. Note the following:
- You cannot use the wildcard expressions .*, .?, or ..* as directory or file names.
- Bracket wildcards, such as file[123].txt, are not supported.
- Wildcards cannot be applied to drive names.
- destinationPath is the full path to the destination file or directory. destinationPath must be an absolute path, and no wildcards are allowed.

The destination directory must exist, unless the parent of the destination already exists and you are copying only one thing.

- recursive, when true, indicates that the Copy utility recursively copies any directories that match the wildcard.

  If recursive is false, the Copy utility does not copy directories, even if they match the wildcard. Instead, it creates intermediate directories required to place the copied files at the destination path.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- The string token assigned to this invocation.

startRollback(RunRollbackType startRollbackInput)

Rollback operations roll back the directory to the most recent backed up version.

For example, say you have a directory called logs, one called logs.2008_10_11.08_00_00, and other, older versions. When you roll back, the following things happen:

- logs is renamed logs.unwanted.
- logs.2008_10_11.08_00_00 is renamed logs.
- The older versions are left alone.

Note: There can only be a single .unwanted directory at a time. If you roll back twice, the .unwanted directory from the first rollback is deleted.

Note: Do not start a backup or rollback operation while another such operation is in progress on the same directory. Unexpected behavior may occur if you do so.

RunRollbackType parameters:

- applicationID identifies the application to use.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- hostID is a unique identifier for the host. The hostID and dirName parameters specify the path to the directory that will be archived.
- dirName is the full path of the directory. The hostID and dirName parameters specify the path to the directory that will be archived.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- The string token assigned to this invocation.

startShell(RunShellType startShellInput)

The startShell() method launches the Shell utility, which allows you to run arbitrary commands in a host system shell.

RunShellType parameters:

- applicationID identifies the application to use.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- hostID is a unique identifier for the host.
- cmd is the command line to execute.
- workingDir is the full path to the working directory.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- The string token assigned to this invocation.

stop(FullyQualifiedUtilityTokenType)

Takes a token returned by any of the start methods, and stops that invocation by terminating the process that is running it.

FullyQualifiedUtilityTokenType parameters:

- applicationID identifies the application to use.
- token identifies the token used to stop the utility.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

getStatus(String applicationID, String token)

Takes a token returned by any of the Utility start methods (startBackup(), startFileCopy(), startRollback(), or startShell()), and returns the current status of that utility.

Parameters:

- applicationID identifies the application to use.
- token identifies the token used to get the utility's status.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- A BatchStatusType object.

listDirectoryContents(ListDirectoryContentsInputType listDirectoryContentsInput)

Performs a list operation similar to UNIX ls on a remote host, with the following restrictions on the input file pattern.

- A filePattern must start with an absolute path, such as C:\ or /.
- A filePattern can contain . or .. as directory names, and expands * and ? wildcards.
- A filePattern cannot contain the wildcard expressions .*, .?, or ..* as directory or file names.
- Bracketed wildcards, such as file[123].txt, are not supported.
- Wildcards cannot be applied to drive names.

- You cannot use .. to create paths that do not exist. For example, the path /temp/../../a.txt refers to a path that is above the root directory. This is an invalid path that causes the operation to fail.

ListDirectoryContentsInputType parameters:

- applicationID (required) identifies the application to use.
- hostID (required) is a unique identifier for the host.
- filePattern (required) is the name of the directory, file, or wildcard combination of directory and file whose contents are to be listed.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails. Failure conditions correspond to bad input cases.

Returns:

- A FilePathListType object representing the contents of the requested directory.

Provisioning interface

The Provisioning interface allows you to define and manage your Guided Search applications programmatically.

It contains the following methods:

defineApplication(ApplicationType application)

Defines an application.

ApplicationType parameters:

- applicationID identifies the application to use.
- hosts is a collection of HostType objects, representing the hosts to define.
- components is a collection of ComponentType objects (such as ForgeComponentType, DgraphComponentType, and so on) representing the components to define.
- scripts is a collection of ScriptType objects.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

getApplication(IDType getApplicationInput)

Gets an application, which is composed of hosts, components, and scripts and identified by an application ID.

IDType parameter:

- applicationID identifies the application to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- An ApplicationType object.

getCanonicalApplication(IDType getCanonicalApplicationInput)

The getCanonicalApplication() method returns the provisioning just as getApplication() does, but with all paths canonicalized.

This process ensures that all paths are absolute, and that the working directory and log path settings are provided. It also prevents .. from being used in a path name.

IDType parameter:

- applicationID identifies the application to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- An ApplicationType object, as described on page 248.

listApplicationIDs(listApplicationIDsInput)

Lists the applications that are defined.

Returns:

- An ApplicationIDListType object.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

removeApplication(RemoveApplicationType removeApplicationInput)

Removes the named application.

RemoveApplicationType parameter:

- applicationID identifies the application to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

addComponent(AddComponentType addComponentInput)

Adds a single component to an application.

AddComponentType parameters:

- applicationID identifies the application to use.

- component is one of the following: Forge, Dgidx, Dgraph, LogServer, ReportGenerator

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

removeComponent(RemoveComponentType removeComponentInput)

Removes a single component from an application.

RemoveComponentType parameters:

- applicationID identifies the application to use.
- componentID identifies the component to use.
- forceRemove indicates whether or not a remove operation should force the component to stop before attempting the remove. If the component is running, and forceRemove is not set to true, then the remove call will fail.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

updateComponent(UpdateComponentType updateComponentInput)

Updates a running component.

UpdateComponentType parameters:

- applicationID identifies the application to use.
- component is one of the following: Forge, Dgidx, Dgraph, LogServer, ReportGenerator.
- forceUpdate indicates that the Application Controller will attempt to force the conditions under which the update can take place, by stopping running components.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

addHost(AddHostType addHostInput)

Adds a host to an application.

AddHostType parameters:

- applicationID identifies the application to use.
- host is a HostType object specifying the host to add.
- directories allows you to specify directories using a full path and a name. These directories are associated with hosts and created when the host is provisioned.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

updateScript(UpdateScriptType updateScriptInput)

Updates a running script.

UpdateScriptType parameters:

- applicationID identifies the application to use.
- script is a ScriptType object specifying the script to be updated.
- forceUpdate is a Boolean that indicates whether the Application Controller should force a running script to stop before attempting the update.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

removeHost(RemoveHostType removeHostInput)

Removes a single host from an application.

RemoveHostType parameters:

- applicationID identifies the application to use.
- hostID is a unique string identifier for this host.
- forceRemove indicates whether or not the Application Controller should force any running components or services to stop before attempting the remove. If a component or service is running, and forceRemove is not set to true, then the remove call will fail.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

updateHost(UpdateHostType updateHostInput)

Updates a running host.

UpdateHostType parameters:

- applicationID identifies the application to use.
- host is a HostType object specifying the host to add.
- directories allows you to specify directories using a full path and a name. These directories are associated with hosts and created when the host is provisioned.
- forceUpdate indicates that the Application Controller will attempt to force the conditions under which the update can take place, by stopping running components or services.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

addScript(AddScriptType addScriptInput)

Adds a script to an application.

AddScriptType parameters:

- applicationID identifies the application to use.
- script is a ScriptType object (see page 269) specifying the script to add.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

removeScript(RemoveScriptType removeScriptInput)

Removes a script from an application.

RemoveScriptType parameters:

- applicationID identifies the application to use.
- scriptID is a unique string identifier for this host.
- forceRemove indicates that the Application Controller will attempt to force the conditions under which the remove can take place.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

ScriptControl interface

The ScriptControl interface provides programmatic script management capabilities.

It contains the following methods:

startScript(FullyQualifiedScriptIDType startScriptInput)

Starts the named script.

FullyQualifiedScriptIDType parameters:

- applicationID identifies the application to use.
- scriptID identifies the script to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

stopScript(FullyQualifiedScriptIDType stopScriptInput)

Stops the named script.

FullyQualifiedScriptIDType parameters:

- applicationID identifies the application to use.
- scriptID identifies the script to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

getScriptStatus(FullyQualifiedScriptIDType getScriptStatusInput)

Returns the status of a script.

FullyQualifiedScriptIDType parameters:

- applicationID identifies the application to use.
- scriptID identifies the script to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- A ScriptStatus object (a sub-class of the StatusType class). This status may be Running, NotRunning, or Failed. (Failure results from a failure error code or internal EAC errors).

Class Reference

This section describes the Endeca Application Controller API classes.

The classes and their properties are documented here. However, the exact syntax of a class member depends on the output of the WSDL tool that you are using. Typically, a Java WSDL tool translates these classes into get and set methods. For example, the ApplicationIDType class would generate getApplicationID() and setApplicationID(String[] applicationID) methods.

The Microsoft .NET WSDL tool translates these classes into .NET properties. Be sure to check the client stub classes that are generated by your WSDL tool for the exact syntax of the Application Controller API class members.

The Application Controller contains the following classes:

AddComponentType class

A class that describes a component to be added to a named application during incremental provisioning.

AddComponentType properties

- applicationID (required) identifies the application to use.
- component (required) is one of the following: Forge, Dgidx, Dgraph, LogServer, or ReportGenerator.

AddHostType class

A class that describes a host to be added to a named application during incremental provisioning.

AddHostType properties

- applicationID (required) identifies the application to use.
- host (required) is a description of the host to add.
- directories allows you to specify directories using a full path and a name.

AddScriptType class

A class that describes a script to be added to a named application during incremental provisioning.

AddScriptType properties

- applicationID (required) identifies the application to use.
- script (required) is a description of the script to add.

ApplicationIDListType class

A class that describes a returned value of a list application call to the Provisioning service.

ApplicationIDListType encapsulates the list of applications running on this EAC Central Server.

ApplicationIDListType properties

- applicationID identifies the application to use.

ApplicationType class

A class that describes an application to be deployed by the Application Controller. An application is composed of a set of components residing on a set of hosts.

You can construct an ApplicationType object as a full specification of the application, including all hosts and components. Alternatively, you can start with an empty an ApplicationType object and incrementally fill in the

hosts, components, and scripts. In the latter case, order matters, because a host must be added before you add a component that lives on that host.

ApplicationType properties

- applicationID identifies the application to use.
- hosts is a list of hosts.
- components is a list of components.
- scripts is a list of scripts.

BackupMethodType class

In relation to the Archive utility, this class serves as an identifier for the type of backup you want the utility to perform, Copy or Move.

BackupMethodType fields

The enumeration of possible values is as follows:

- Copy.
- Move.

BatchStatusType class

Based on the StatusType class, a BatchStatusType object describes the status of a batch component.

Batch components include Forge, Dgidx and ReportGenerator..

BatchStatusType properties

- StateType – (required) An enumeration of the following fields:

  Starting, which only applies to server components (Dgraph or LogServer)

  Running

  NotRunning

  Failed

- startTime – (required) The time the batch component started; for example, 10/11/08 3:58 PM.
- failureMessage – The failure message, which tells you that a failure has occurred in the execution of the component. failureMessage is empty unless state is FAILED. (This is different from EACFault, which tells you that a problem has occurred while processing the Web Service request to get the status.)
- duration – (required) The length of time the batch component has been running; for example, 0 days 0 hours 0 minutes 6.96 seconds.

ComponentListType class

A class that describes a list of components, such as ForgeComponentType and DgraphComponentType.

ComponentListType properties

- component (required) A collection of components comprising this ComponentListType object.

ComponentType class

A class that describes the base type for all components within an application.

ComponentType properties

Each component contains these properties, as well as some others.

- componentID (required) identifies the component to use.
- hostID (required) is a unique string identifier for this host.
- workingDir is a string identifying the working directory for this component.
- logFile is a string identifying the log file for this component.
- properties is a string identifying any properties associated with this component.

DgidxComponentType class

A class that describes a Dgidx component within an application.

A Dgidx component sends the finished data prepared by Forge to the Dgidx program, which generates the proprietary indices for each Dgraph.

DgidxComponentType properties

- componentID (required) identifies the component to use.
- hostID (required) is a unique string identifier for this host.
- workingDir is a string identifying the working directory for this component. Any relative paths in component properties are be interpreted as relative to the component's workingDir. The workingDir property, if specified, must be an absolute path.
- logFile is a string identifying the log file for this component.
- args is a list of command-line flags to pass to Dgidx.
- appConfigPrefix is the path and file prefix that define the input for Dgidx.
- inputPrefix (required) is the path and prefix name for the Forge output that Dgidx indexes.
- outputPrefix (required) is the path and prefix name for the Dgidx output.
- runAspell prepares the Aspell files for the Dgraph. The default is true, which causes the Dgidx component to run dgwordlist and to copy the Aspell files to its output directory, where the Dgraph component can access them. Note that your stub generation tool may generate a Boolean property (for example, runAspellSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.
- tempDir is the path to the temporary directory that Dgidx uses.

DgraphComponentType class

A class that describes a Dgraph component within an application.

A Dgraph element launches the Dgraph (MDEX Engine) software, which processes queries against the indexed Endeca records.

DgraphComponentType properties

- componentID (required) identifies the component to use.
- hostID (required) is a unique string identifier for this host.
- workingDir is a string identifying the working directory for this component. Any relative paths in component properties are be interpreted as relative to the component's workingDir. The workingDir property, if specified, must be an absolute path.
- logFile is a string identifying the log file for this component.
- args is a list of command-line flags to pass to the Dgraph.
- port (required) is the port the Dgraph listens at. The default is 8000.
- appConfigPrefix is the path and file prefix that define the input for Dgraph.
- inputPrefix (required) is the path and prefix name for the Dgidx output that the Dgraph uses as an input.
- reqLogFile is the path to and name of the request log.
- spellDir, if specified, is the directory in which the Dgraph will look for Aspell files. If it is not specified, the Dgraph will look for Aspell files in the Dgraph's input directory (that is, inputPrefix without the prefix). For example, if inputPrefix is /dir/prefix and all the Dgraph input files are /dir/prefix.*, the Dgraph will look for the Aspell files in /dir/).
- startupTimeout specifies the amount of time in seconds that the Application Controller will wait while starting the Dgraph. Note that your stub generation tool may generate a Boolean property (for example, startupTimeoutSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.
- sslConfiguration sets SSL usage for this Dgraph.
- updateDir is the directory from which Dgraph reads partial update files. For more information, see the *Endeca Partial Updates Guide*.
- updateLogFile specifies the file for update-related log messages.
- tempDir is the path to the temporary directory that the Dgraph uses.

DirectoryListType class

A class that represents a collection of DirectoryType objects.

DirectoryListType property

- directory (required) is a collection of DirectoryType objects.

DirectoryType class

A class used by the HostType class to define a directory while provisioning a host.

DirectoryType properties

- dirID (required) is a unique identifier for this directory.
- dir (required) is a full path for this directory.

EACFault class

The class that creates the EACFault. EACFault is the error message returned by the Application Controller when the method fails.

EAC Fault property

- error is the error message.

FilePathListType class

An array of FilePathTypes that describes a returned value of a listDirectoryContents call.

FilePathListType operates on the application level.

FilePathListType property

- filePaths (required) describe a file on a remote host.

FilePathType class

A class that describes a file on a remote host.

FilePathType properties

- path (required) is the full path to the file.
- directory (required) indicates whether the path is a directory.

FlagIDListType class

A class that describes a returned value of a list flags call. FlagIDListType operates on the application level.

FlagIDListType property

- flagID is a unique string identifier for this flag.

ForgeComponentType class

A class that describes a Forge component within an application.

A Forge element launches the Forge (Data Foundry) software, which transforms source data into tagged Endeca records.

ForgeComponentType properties

- componentID (required) identifies the component to use.
- hostID (required) is a unique string identifier for this host.
- workingDir is a string identifying the working directory for this component. Any relative paths in component properties are be interpreted as relative to the component's workingDir. The workingDir property, if specified, must be an absolute path.
- logFile is a string identifying the log file for this component.
- args is a list of command-line flags to pass to Forge.
- stateDir is the directory where the state file is located.
- inputDir is the path to the Forge input.
- outputDir is the directory where the output from the Forge process will be stored.
- outputPrefixName is the prefix, without any associated path information, that Forge uses to save its output files. These files are located in the directory specified by outputDir.
- numPartitions is the number of partitions. Note that your stub generation tool may generate a Boolean property (for example, numPartitionsSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.
- pipelineFile (required) is the name of the Pipeline.epx file to pass to Forge.
- tempDir is the temporary directory that Forge uses.
- webServicePort is the port used by the Forge metrics Web service, which provides progress and performance metrics for Forge. For details, see "The Forge Metrics Web Service" in the *Endeca Forge Guide*. Note that your stub generation tool may generate a Boolean property (for example, webServicePortSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

FullyQualifiedComponentIDType class

A class that serves as an input to the start, stop, get status, and remove component commands.

FullyQualifiedComponentIDType properties

- applicationID (required) identifies the application to use.
- componentID (required) identifies the component to use.

FullyQualifiedFlagIDType class

In relation to the Synchronization service, this class serves as an input to an acquire or release flag method.

FullyQualifiedFlagIDType properties

- applicationID (required) identifies the application to use.
- flagID (required) is a unique string identifier for this flag.

FullyQualifiedHostIDType class

A class that identifies a host so that it can be used as an input to another command, such as remove host.

FullyQualifiedHostIDType properties

- applicationID (required) identifies the application to use.
- hostID (required) is a unique string identifier for this host.

FullyQualifiedScriptIDType class

A class that identifies a script so that it can be used as an input to another command, such as startScript().

FullyQualifiedScriptIDType properties

- applicationID (required) identifies the application to use.
- scriptID (required) is a unique string identifier for this script.

FullyQualifiedUtilityTokenType class

In relation to the Utility service, this object represents the token.

FullyQualifiedUtilityTokenType properties

- applicationID (required) identifies the application to use.
- token (required) identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.

HostListType class

A class that represents a collection of HostType objects.

HostListType property

- host (required) is a unique identifier comprising a hostname, port, and hostID.

HostType class

A class that describes a host within an application.

Along with components, a collection of HostType objects define an application.

HostType properties

- hostname (required) is the name of the host.
- port (required) is the connection port.
- hostID is a unique string identifier for this host.
- directories allows you to specify directories using a full path and a name.

ListApplicationIDsInput class

An empty object you pass into the Web services interface to get back a list of applications.

ListDirectoryContentsInputType class

An object that serves as an input to the listDirectoryContents object.

ListDirectoryContentsInputType properties

- applicationID (required) identifies the application to use to look up the host.
- hostID (required) is a unique identifier for the host within that application.
- filePattern (required) is the pattern that listDirectoryContents() expands the wildcards in a pattern. If the expansion results in a file, it returns a file. If the expansion results in a directory, it returns the directory non-recursively. Wildcard expansion can result in any combination of files and directories.

LogServerComponentType class

A class that describes a LogServerComponent within an application.

The LogServer component controls the use of the Endeca Log Server.

LogServerComponentType properties

- componentID (required) identifies the component to use.
- hostID (required) is a unique string identifier for this host.
- workingDir is a string identifying the working directory for this component. Any relative paths in component properties are be interpreted as relative to the component's workingDir. The workingDir property, if specified, must be an absolute path.
- logFile is a string identifying the log file for this component.
- port (required) is the port on which to run the LogServer.
- outputPrefix (required) is the path and prefix name for the LogServer output.
- gzip (required) controls the archiving of log files. Possible values are true and false.
- startupTimeout (required) specifies the amount of time in seconds that the Application Controller will wait while starting the LogServer. Note that your stub generation tool may generate a Boolean property (for example, startupTimeoutSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

PropertyListType class

A class that represents a collection of PropertyType objects.

PropertyListType property

- properties is a collection of name/value pairs.

PropertyType class

The PropertyType class allows you to add arbitrary properties (that is, name/value pairs) to host and all component elements.

PropertyType properties

- name (required) is a non-null string.
- value is a string.

ProvisioningFault class

An extension of EACFault, the ProvisioningFault class is thrown when there are fatal errors during provisioning.

ProvisioningFault properties

- errors is a list of provisioning errors.
- warnings is a list of provisioning warnings.

RemoveApplicationType class

Related to the Provisioning service, this class serves as input to the incremental remove command.

RemoveApplicationType properties

- applicationID (required) identifies the application to use.
- forceRemove indicates whether or not a remove operation should force any running components or services to stop before attempting the remove. Note that your stub generation tool may generate a Boolean property (for example, forceRemoveSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

RemoveComponentType class

Related to the Provisioning service, this class serves as input to the incremental remove command.

RemoveComponentType properties

- FullyQualifiedComponentIDType (required) identifies the component to use.
- forceRemove indicates whether or not a remove operation should force the component to stop before attempting the remove. Note that your stub generation tool may generate a Boolean property (for example, forceRemoveSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

RemoveHostType class

Related to the Provisioning service, this class serves as input to the incremental remove command.

RemoveHostType properties

- FullyQualifiedHostIDType (required) is a unique string identifier for this host.
- forceRemove is a Boolean that indicates whether or not a remove operation should force any running components or services to stop before attempting the remove. Note that your stub generation tool may generate a Boolean property (for example, forceRemoveSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

RemoveScriptType class

Related to the Provisioning service, this class serves as input to the incremental remove command.

RemoveScriptType properties

- applicationID (required) identifies the application.
- scriptID (required) identifies the script to remove.
- forceRemove is a Boolean that indicates whether or not a remove operation should force any running components or services to stop before attempting the remove. Note that your stub generation tool may generate a Boolean property (for example, forceRemoveSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

ReportGeneratorComponentType class

A class that describes a ReportGenerator component within an application.

The ReportGenerator component runs the Report Generator, which processes Log Server files into HTML-based reports that you can view in your Web browser and XML reports that you can view in Workbench.

ReportGeneratorComponentType properties

- componentID (required) identifies the component to use.
- hostID (required) is a unique string identifier for this host.
- workingDir is a string identifying the working directory for this component. Any relative paths in component properties are be interpreted as relative to the component's workingDir. The workingDir property, if specified, must be an absolute path.
- logFile is a string identifying the log file for this component. args is a list of command-line flags to pass to the ReportGenerator.
- javaBinary, if used, should indicate a JDK 1.5.x or later. Defaults to the JDK that Endeca installs.
- javaOptions are the command-line options for the javaBinary parameter. This parameter is primarily used to adjust the ReportGenerator memory, which defaults to 1GB. To set the memory, use the following: java_options = -Xmx[MemoryInMb]m -Xms[MemoryInMb]m inputDirOrFile (required) is the path to the file or directory containing the logs to report on. If it is a directory, then all log files in that directory are read. If it is a file, then just that file is read.
- outputFile (required) is the name the generated report file and path to where it is stored.
- stylesheetFile (required) is the filename and path of the XSL stylesheet used to format the generated report.
- settingsFile is the path to the report_settings.xml file.

- timerange sets the time span of interest (or report window). Allowed keywords: Yesterday, LastWeek, LastMonth, DaySoFar,WeekSoFar, and MonthSoFar. These keywords assume that days end at midnight, and weeks end on the midnight between Saturday and Sunday. Note that your stub generation tool may generate a Boolean property (for example, timerangeSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.
- startDate set the report window to the given date and time. The date format should be either yyyy_mm_dd or yyyy_mm_dd.hh_mm_ss.
- stopDate sets the report window to the given date and time. The date format should be either yyyy_mm_dd or yyyy_mm_dd.hh_mm_ss. timeSeries turns on the generation of time-series data and specifies the frequency, Hourly or Daily.
- charts turns on the generation of report charts. Note that your stub generation tool may generate a Boolean property (for example, chartsSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

RunBackupType class

A child of the RunUtilityType class, this class provides all the information you need to perform a backup operation to the Archive utility.

RunBackupType properties

- applicationID (required) is the unique identifier for this application.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- hostID (required) is a unique identifier for the host. The hostID and dirName parameters specify the path to the directory that will be archived.
- dirName (required) is the full path of the directory. The hostID and dirName parameters specify the path to the directory that will be archived.
- backupMethod is either Copy or Move. Note that your stub generation tool may generate a Boolean property (for example, backupMethodSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.
- numBackups specifies the maximum number of archives to store. This number does not include the original directory itself, so if numBackups is set to 3, you would have the original directory plus up to three archive directories, for a total of as many as four directories. The default numBackups is 5. Note that your stub generation tool may generate a Boolean property (for example, numBackupsSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

RunFileCopyType class

A child of the RunUtilityType class, this class provides all the information you need to run the Copy utility.

RunFileCopyType properties

- applicationID (required) identifies the application to use.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- fromHostID (required) is the unique identifier for the host you are copying the data from. toHostID (required) is the unique identifier for the host you are copying the data to.

- sourcePath (required) is the full path to the source file or directory. If sourcePath contains no wildcards, then destinationPath must be the destination file or directory itself, rather than the parent directory.
- destinationPath (required) is the full path to the destination file or directory.
- recursive, when specified, downloads the directories recursively. Note that your stub generation tool may generate a Boolean property (for example, recursiveSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

RunRollbackType class

A child of the RunUtilityType class, this class provides all the information you need to perform a rollback operation to the Archive utility.

RunRollbackType properties

- applicationID (required) identifies the application to use.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- hostID (required) is a unique identifier for the host. The hostID and dirName parameters specify the path to the directory that will be archived.
- dirName (required) is the full path for the directory. The hostID and dirName parameters specify the path to the directory that will be archived.

RunShellType class

A child of the RunUtilityType class, this class provides all the information you need to run the Shell utility.

RunShellType properties

- applicationID (required) identifies the application to use.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- hostID (required) is a unique identifier for the host.
- cmd (required) is the command(s). workingDir is the full path for the working directory.

RunUtilityType class

Parent class of the other Utility classes.

RunUtilityType properties

- applicationID (required) identifies the application to use.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.

ScriptListType class

A class that describes a list of scripts.

ScriptListType properties

- script (required) is a collection of scripts comprising this ScriptListType object.

ScriptType class

A class that describes the base type for all scripts within an application.

ScriptType properties

- scriptID (required) is a unique string identifier for the script.
- cmd (required) is the command that is used to start the script.
- logFile is the file for appended stdout/stderr output. It defaults to $ENDECA_CONF/logs/script/(app_id).(script_id).log.
- workingDir is the working directory. It defaults to $ENDECA_CONF/working/(app_id)/.

SSLConfigurationType class

A class used by the DgraphComponentType class to enable SSL on the resulting components.

SSLConfigurationType properties

- certFile (required) specifies the path of the eneCert.pem certificate file that is used by the Dgraph process to present to any client.

  The file name can be a path relative to the component's working directory.
- caFile (required) specifies the path of the eneCA.pem Certificate Authority file that the Dgraph process uses to authenticate communications with other Guided Search components. The file name can be a path relative to the component's working directory.
- cipher specifies a cryptographic algorithm that Dgraph will use during the SSL negotiation. If you omit this setting, the Dgraph chooses a cryptographic algorithm from its internal list of algorithms. See the *Endeca Commerce Security Guide* for more information.

StateType class

A class used by the StatusType class to describe the state of a component.

StateType fields

An enumeration of the following fields:

- Starting Starting only applies to server components (Dgraph or LogServer).
- Running
- NotRunning
- Failed

StatusType class

Describes the status of a server component in the Application Controller.

Server components include the Dgraph and LogServer. All other components (Forge, Dgidx, and ReportGenerator) are batch components. Their status is described by the BatchStatusType class.

StatusType properties

- StateType – (required) An enumeration of the following fields: Starting (which only applies to server components (Dgraph or LogServer), Running, NotRunning, or Failed.
- startTime – (required) The time the component started; for example, 10/25/07 3:58 PM.
- failureMessage – The failure message, which tells you that a failure has occurred in the execution of the component. failureMessage is empty unless state is FAILED. (This is different from EACFault, which tells you that a problem has occurred while processing the Web Service request to get the status.)

TimeRangeType class

A class used by the ReportGeneratorComponentType class to set the time span of interest (or report window).

TimeRangeType fields

The enumeration of possible values is as follows:

- Yesterday
- LastWeek
- LastMonth
- DaySoFar
- WeekSoFar
- MonthSoFar

TimeSeriesType class

A class used by the ReportGeneratorComponentType class to turn on the generation of time-series data and specify the frequency, hourly or daily.

TimeSeriesType fields

The enumeration of possible values is as follows:

- Hourly
- Daily

UpdateComponentType class

A class that describes a component to be updated during incremental provisioning.

UpdateComponentType properties

- applicationID (required) identifies the application.
- component (required) identifies the component to update.
- forceUpdate indicates whether or not the Application Controller should force the component to stop before attempting the update. Note that your stub generation tool may generate a Boolean property (for example, forceUpdateSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

UpdateHostType class

A class that describes a host to be updated during incremental provisioning.

UpdateHostType properties

- applicationID (required) identifies the application.
- host (required) identifies the host to update.
- forceUpdate indicates whether the Application Controller should force any components or services running on the host to stop before attempting the update. Note that your stub generation tool may generate a Boolean property (for example, forceUpdateSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

UpdateScriptType class

A class that describes a script to be updated during incremental provisioning.

UpdateScriptType properties

- applicationID (required) identifies the application.
- scriptID (required) identifies the script to update.
- forceUpdate indicates whether the Application Controller should force any components or services running on the host to stop before attempting the update. Note that your stub generation tool may generate a Boolean property (for example, forceUpdateSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X