

Endeca® Information Access Platform

Administrator's Guide

ORACLE®

ENDECA

Contents

Preface.....	9
About this guide.....	9
Who should use this guide.....	10
Conventions used in this guide.....	10
Contacting Oracle Support.....	10
Chapter 1: Introduction.....	11
Taking ownership of your Endeca implementation.....	11
About the Endeca Application Controller.....	12
EAC architecture.....	12
EAC architecture example.....	14
About the Deployment Template.....	14
Directories created by the Deployment Template.....	15
About the AppConfig.xml file.....	16
Typical workflows.....	17
The control framework.....	17
Data and configuration workflow.....	18
Logging and reporting workflow.....	19
Chapter 2: Creating Multiple Server Environments with the Deployment Template.23	
About a multiple server environment.....	23
Overview of staging and production environments	24
Planning your server topology.....	26
Configuring the application on multiple servers.....	27
Changing server settings in AppConfig.xml	28
Adding MDEX Engine servers.....	30
Adding Dgraphs.....	33
Additional customization tasks.....	34
Chapter 3: Replicating application definitions across environments...37	
About replicating application definitions using the Deployment Template.....	37
Identifying the artifacts that make up an application.....	39
Creating a custom file for environment-specific settings.....	41
Controlling paths to ensure interoperability across environments.....	42
Automating the collection of files.....	44
Distributing application definitions between environments for the first time.....	44
Distributing an updated application definition with another environment.....	45
Approaches to avoiding synchronization conflicts.....	45
Chapter 4: Performing System Operations with the EAC.....47	
Options for provisioning the application.....	47
Updating the application provisioning.....	48
Backing up the EAC application provisioning with eaccmd.....	48
Options for running system operations.....	48
Checking the status of EAC components.....	49
Avoiding defunct EAC processes.....	50
EAC memory usage.....	51
Deployment Template and Endeca Workbench interaction.....	51
Archiving the Dgraph log files	53
Releasing locks set by the Deployment Template in the EAC.....	53
Removing components from your configuration.....	54
Removing components in a Deployment Template environment.....	56
Determining the state of the EAC with service URLs.....	56
Logs for the EAC Central Server.....	56
Changing the IP address for the EAC Central Server machine.....	57

Chapter 5: Administering Dgidx.....	59
Dgidx processing and memory usage.....	59
Running the Dgidx process with the Deployment Template.....	60
Running the Dgidx binary at the command prompt.....	60
Tips for speeding up indexing time.....	61
Troubleshooting Dgidx failures.....	61
Dgidx logs.....	63
Dgidx log details for text search indexing.....	64
Dgidx handling of records with missing or duplicate record spec values.....	65
Variations in Dgidx indexing time.....	66
Chapter 6: Administering the Dgraph.....	67
Checking Dgraph and Agraph with the ping command.....	67
Specifying arguments to the Dgraph in the Deployment Template.....	67
Collecting debugging information.....	68
The logs created by the Dgraph.....	68
The Agraph request log.....	70
Troubleshooting baseline update failures.....	70
Troubleshooting partial updates.....	71
Identifying connection errors.....	71
Troubleshooting socket and port errors with Dgraph.....	72
Managing the Dgraph core dump files.....	73
Managing Dgraph crash dump files on Windows.....	73
Managing Dgraph core dump files on Linux and Solaris.....	73
Chapter 7: Backing up Endeca applications.....	75
Required files for backup.....	75
Backing up CAS configurations.....	76
Backing up the Discovery Framework.....	77
What not to backup.....	77
When primary and recovery environments are different.....	78
Appendix A: Administrative and configuration operations and logging variables.	79
About administrative and configuration operations.....	79
List of administrative operations.....	79
List of configuration operations.....	86
About MDEX Engine logging variables.....	88
Logging variable operation syntax.....	88
List of supported logging variables.....	88
Appendix B: Endeca Flag Reference.....	91
Agidx flags.....	91
Agraph flags.....	92
Dgidx flags.....	94
Dgraph flags.....	96
Appendix C: XML Configuration Files.....	107
About the XML configuration files.....	107
Creating the XML configuration files.....	107
Changing the Deployment Template output prefix.....	108
Creating and modifying the XML configuration files.....	108
Appendix D: Transferring Endeca Implementations Between Environments.	109
For implementations using the Deployment Template.....	109
About transferring your implementation.....	109
Retrieving the Endeca Workbench instance configuration with Developer Studio.....	110
About emgr_update.....	110
emgr_update syntax reference.....	111
About transferring implementations using the emgr_update utility.....	113
Removing instance configuration files from Endeca Workbench.....	116
Sending the dimensions file produced by Forge to Endeca Workbench.....	116
Removing inactive rules from an instance configuration.....	117

Transferring auto-generated and external dimension value ID assignments.....118
Removing an application from Endeca IAP.....119



Copyright and disclaimer

Product specifications are subject to change without notice and do not represent a commitment on the part of Endeca Technologies, Inc. The software described in this document is furnished under a license agreement. The software may not be reverse engineered, decompiled, or otherwise manipulated for purposes of obtaining the source code. The software may be used or copied only in accordance with the terms of the license agreement. It is against the law to copy the software on any medium except as specifically allowed in the license agreement.

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of Endeca Technologies, Inc.

Copyright © 2009-2010 Endeca Technologies, Inc. All rights reserved. Printed in USA.

Portions of this document and the software are subject to third-party rights, including:

Outside In® Search Export Copyright © 2008 Oracle. All rights reserved.

Rosette® Globalization Platform Copyright © 2003-2005 Basis Technology Corp. All rights reserved.

Trademarks

Endeca, the Endeca logo, Guided Navigation, MDEX Engine, Find/Analyze/Understand, Guided Summarization, Every Day Discovery, Find Analyze and Understand Information in Ways Never Before Possible, Endeca Latitude, Endeca Profind, Endeca Navigation Engine, and other Endeca product names referenced herein are registered trademarks or trademarks of Endeca Technologies, Inc. in the United States and other jurisdictions. All other product names, company names, marks, logos, and symbols are trademarks of their respective owners.

The software may be covered by one or more of the following patents: US Patent 7035864, US Patent 7062483, US Patent 7325201, US Patent 7424528, US Patent 7567957, US Patent 7617184, Australian Standard Patent 2001268095, Republic of Korea Patent 0797232, Chinese Patent for Invention CN10461159C, Hong Kong Patent HK1072114, European Patent EP1459206B1, and other patents pending.

Endeca IAP Administrator's Guide • December 2010

Preface

Oracle Endeca's Web commerce solution enables your company to deliver a personalized, consistent customer buying experience across all channels — online, in-store, mobile, or social. Whenever and wherever customers engage with your business, the Oracle Endeca Web commerce solution delivers, analyzes, and targets just the right content to just the right customer to encourage clicks and drive business results.

Oracle Endeca Commerce 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, Oracle Endeca Commerce enables businesses to help guide and influence customers in each step of their search experience. At the core of Oracle Endeca Commerce is the MDEX Engine,™ a hybrid search-analytical database specifically designed for high-performance exploration and discovery. The Endeca 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. Endeca Assembler dynamically assembles content from any resource and seamlessly combines it with results from the MDEX Engine.

Oracle Endeca Experience Manager is a single, flexible solution that enables you to create, deliver, and manage content-rich, cross-channel customer experiences. It also enables non-technical business users to deliver targeted, user-centric online experiences in a scalable way — creating always-relevant customer interactions that increase conversion rates and accelerate cross-channel sales. Non-technical users can control how, where, when, and what type of content is presented in response to any search, category selection, or facet refinement.

These components — along with additional modules for SEO, Social, and Mobile channel support — make up the core of Oracle Endeca Experience Manager, a customer experience management platform focused on delivering the most relevant, targeted, and optimized experience for every customer, at every step, across all customer touch points.

About this guide

This guide describes tasks involved in administering and maintaining applications built upon the Endeca Information Access Platform.

It bridges the gap between the work performed by the Endeca Services team when your Endeca implementation was being initially deployed, and the issues that you, as a system administrator, may need to address to maintain the system.

About the contents of the guide

The guide introduces basic Endeca workflows and environments, and discusses the topology, indicating which physical servers should host specific Endeca components. It then identifies the control framework: the Endeca Application Controller (EAC) that is managed by Deployment Template scripts. In the following sections, the guide reviews EAC and the Deployment Template. Next, it discusses various administrative tasks related to the Deployment Template, EAC, Dgidx and Dgraph, such as operational tasks and logging.

This guide mainly focuses on Dgraph implementations, and only mentions the Agraph briefly. See the *Performance Tuning Guide* for a discussion about when you should use an Agraph rather than a set

of load-balanced Dgraphs. In addition, while this guide mentions updates, it is assumed that customers implementing updates will refer to the *Partial Updates Guide* for information.

The guide also contains the reference of all Endeca administrative and configuration operations, an Endeca flag reference, an overview of the XML configuration files, and an appendix that describes how to transfer an Endeca implementation between different environments if you are using the Endeca Workbench (and not the Deployment Template).

Who should use this guide

This guide is intended for system administrators who administer and maintain an Endeca implementation.

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

You should also be familiar with the Endeca concepts described in the *Concepts Guide*.

You can choose to read specific topics from this guide individually as needed while maintaining your Endeca implementation after it has been initially deployed.

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 important information regarding Oracle Endeca software, implementation questions, product and solution help, as well as overall news and updates.

You can contact Oracle Support through Oracle's Support portal, My Oracle Support at <https://support.oracle.com>.



Chapter 1

Introduction

This section describes the stage at which you take control of the operation and maintenance of your Endeca implementation. It also introduces basic administrative components of the system — the Endeca Application Controller (EAC), and the Deployment Template.

Taking ownership of your Endeca implementation

As a system administrator, you take ownership of the Endeca implementation at a certain stage. This topic describes the context in which you will perform administrative tasks to maintain the stable operation of a properly functioning Endeca implementation.

This guide assumes that by this point in using the Endeca software you or your team have done the following:

- Planned and provisioned the hardware needed for the staging and production environments.
- Installed the Endeca components, including the MDEX Engine, Platform Services, Endeca Workbench and the Deployment Template.
- Read the *Endeca Getting Started Guide*.
- Run the Deployment Template's sample scripts for the sample reference implementation on the development or staging servers.

When you complete your engagement with Endeca Professional Services, you obtain an Endeca Technical Specification for your Endeca deployment. This assumes the following:

- You have completed the process of extracting source information from your incoming data sources and feeding the data into the Endeca pipeline using the Content Acquisition System (CAS).
- You have used the Deployment Template scripts to run the offline Forge and Dgidx processes, thus creating the Endeca index files.
- You have deployed your Endeca solution in a staging environment, and are either preparing to deploy it in production, or have already deployed it in production.
- You have created a working prototype of your Endeca front-end application for your end users. This front-end application can be used to issue requests to the running MDEX Engine in a production environment.

Related Links

[The control framework](#) on page 17

This guide assumes that you are using the Deployment Template with the Endeca Application Controller (EAC) as your primary control framework.

[About the Endeca Application Controller](#) on page 12

The Endeca Application Controller (EAC) is a control system you can use to control, manage, and monitor components in your Endeca implementation.

[About the Deployment Template](#) on page 14

The Deployment Template provides a collection of operational components that serve as a starting point for development and application deployment.

About the Endeca Application Controller

The Endeca Application Controller (EAC) is a control system you can use to control, manage, and monitor components in your Endeca implementation.

The EAC provides the infrastructure to support Endeca projects from design through deployment and runtime. It replaces the deprecated Control Interpreter, while leaving the Endeca tools (Developer Studio and Endeca Workbench) largely intact.

The EAC uses open standards, such as the Web Services Descriptive Language (WSDL), which makes the Application Controller platform- and language-independent. As a result, the Application Controller supports a wide variety of applications in production. It allows you to handle complex operating environments that support features such as partial updates, delta updates, phased MDEX Engine updates, and more.

Related Links

[EAC architecture](#) on page 12

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

[EAC architecture example](#) on page 14

A typical Endeca 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.

EAC architecture

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

Depending on the role that the EAC plays in the Endeca implementation, each instance of the EAC can take one of two roles:

- EAC Central Server
- EAC Agent

You can communicate with the EAC and provide instance configuration and resource configuration information to the EAC Central Server, using any of the three methods:

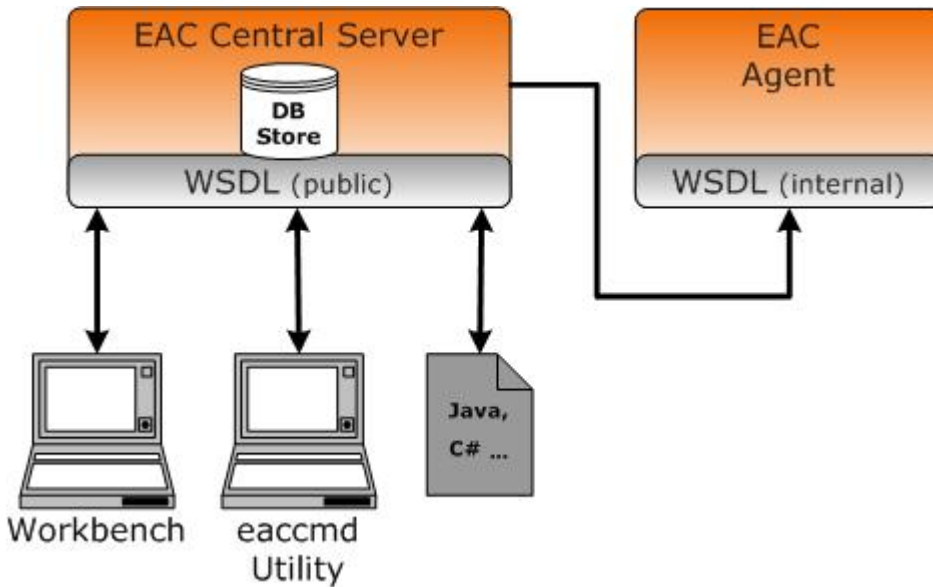
- Endeca Workbench. Endeca Workbench communicates through the WSDL interface to the EAC Central Server. Using Endeca Workbench you can provision, run, and monitor your application. For details, see the *Endeca Workbench Help*.
- The command line utility, `eaccmd`. `eaccmd` lets you script the EAC within a language such as Perl, shell, or batch.
- Direct programmatic control through the Endeca WSDL-enabled interface and languages, such as Java, that support Web services.



Note: The Endeca Deployment Template utilizes this method for communication with the EAC Central Server.

Using any of these methods, you can instruct the EAC to perform different operations in your Endeca implementations, such as start or stop a component (for example, Forge or Dgraph), or a utility (for example, Copy or Shell environment).

The following diagram describes the EAC architecture and means of communication with it, while the sections below describe the roles of the EAC Central Server and EAC Agents:



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 you communicate with the EAC. Communication is implemented with the standard Web services protocol, SOAP.

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 is managing.



Note: You should configure only one EAC Central Server for a given application. The EAC can run into issues when multiple Central Servers are provisioned with the same application on the same EAC Agents (for example, it can lead to confusing clean-up instructions being sent to the Agents from multiple Central Servers, which can interrupt scripts).

EAC Agents

All other instances of the EAC serve as Agents. The Agents instruct their host machines to do the actual work of an Endeca implementation, such as processing data with a Forge component, or coordinating the workings of multiple MDEX Engines with an Aggregated MDEX Engine component.

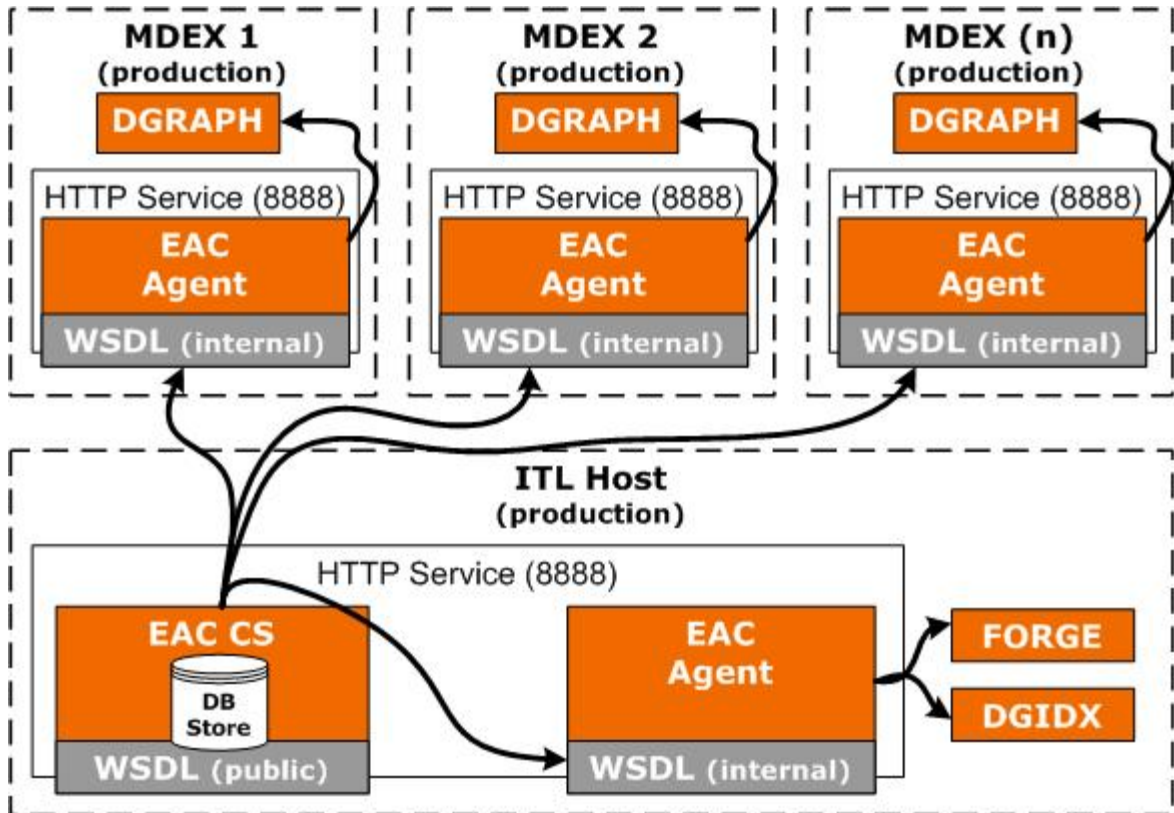
Each Agent also contains a small repository for its own use. The EAC Central Server communicates with its Agents through an internal Web service interface. You do not communicate directly with the Agents—all command, control, and monitoring functions are sent through the EAC Central Server.

EAC architecture example

A typical Endeca 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 development and application deployment.

The template includes the complete directory structure required for deployment, including Endeca Application Controller (EAC) scripts, configuration files, and batch files or shell scripts that wrap common script functionality.

The Deployment Template is the recommended method for building your application deployment environment.

Related Links

[Deployment Template and Endeca Workbench interaction](#) on page 51

This topic summarizes information about Deployment Template and Endeca Workbench interaction.

[Directories created by the Deployment Template](#) on page 15

The Deployment Template creates the following default directory structure. For each Endeca implementation that is deployed with the Deployment Template, look into these directories to identify currently used configuration options and scripts.

[About the AppConfig.xml file](#) on page 16

The `[appdir]/config/script/AppConfig.xml` file is the central configuration file of the Deployment Template. It defines the hosts and Endeca components that make up the Endeca implementation known to the EAC. It also defines the scripts that orchestrate updates by running the defined components.

Directories created by the Deployment Template

The Deployment Template creates the following default directory structure. For each Endeca implementation that is deployed with the Deployment Template, look into these directories to identify currently used configuration options and scripts.

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. Therefore, its `AppConfig.xml` file contains all the possible blocks and directories that you may need on your production servers.

For example, the Deployment Template has separate directories to ensure that the MDEX Engine operations are safely accessing only the information they need. Further, the default Deployment Template allows for configuring multiple Dgraphs, so additional directories are created to facilitate this task.

Directory	Contents
<code>config/lib</code>	Subdirectories to store any custom scripts or code for your Deployment Template project.
<code>config/pipeline</code>	The Developer Studio pipeline file and XML configuration files.
<code>config/report_templates</code>	Files required to generate an application's reports.
<code>config/script</code>	The <code>AppConfig.xml</code> file and related Deployment Template scripts responsible for defining the baseline update workflow and communication of different Endeca components with the EAC Central Server.
<code>control</code>	Shell (UNIX) or batch (Windows) scripts responsible for running different operations defined within <code>AppConfig.xml</code> .
<code>data/incoming</code>	The premodified incoming data files that are ready acquisition by the Endeca pipeline and should be processed.
<code>data/processing</code>	Temporary data and configuration files created and stored during the baseline update process.
<code>data/forge_output</code>	The data and configuration files that are output from the Forge process to the Dgidx process.
<code>data/dgidx_output</code>	The index files that are output from the Dgidx process.

Directory	Contents
data/dgraphs	The copy of the index files used by an instance of the MDEX Engine.
data/state	Autogenerated dimensions files.
data/complete_index_config	Merged configuration (that is, Developer Studio files from <code>config/pipeline</code> , with any Workbench- maintained files specified in the Deployment Template's <code>ConfigManager</code> component overwritten by files downloaded from the Workbench instance).
data/web_studio/config	Configuration files extracted from Workbench by the Deployment Template's <code>ConfigManager</code> component.
logs	Various log files within subdirectories, such as <code>Dgidx</code> logs.
reports	Generated reports.

About the AppConfig.xml file

The `[appdir]/config/script/AppConfig.xml` file is the central configuration file of the Deployment Template. It defines the hosts and Endeca components that make up the Endeca implementation known to the EAC. It also defines the scripts that orchestrate updates by running the defined components.

If you do not require all components in your staging or development environments, or need more than one of each component, you can modify the `AppConfig.xml` file to add or remove them.



Note: This guide describes most frequently used tasks you can do with `AppConfig.xml`. For detailed information about how each component is defined in `AppConfig.xml`, see the *Endeca Deployment Template Usage Guide*.

Related Links

[About the schema for AppConfig.xml](#) on page 16

The `eacToolkit.xsd` schema determines the valid syntax within `AppConfig.xml`.

About the schema for AppConfig.xml

The `eacToolkit.xsd` schema determines the valid syntax within `AppConfig.xml`.

The `eacToolkit.xsd` file is located at the top level of the `eacHandlers.jar` archive file. If any of your Deployment Template scripts fail due to XML syntax errors, you can look at the schema to learn which syntax options for attributes and values are allowed. You may decide to modify the schema to allow you to specify the options you need.

This archive file resides in the `config/lib/java` sub-directory of a deployed application. It also resides in the `data/eac-java/common/config/lib/java` directory of the Deployment Template installation.

To explore this file, use the following command at a prompt from the directory containing the `eacHandlers.jar` archive file:

- On UNIX: `$ENDECA_ROOT/j2sdk/bin/jar xvf eacHandlers.jar eacToolkit.xsd`
- On Windows: `%ENDECA_ROOT%\j2sdk\bin\jar xvf eacHandlers.jar eacToolkit.xsd`

Typical workflows

This section defines basic workflows that take place within the Endeca implementation.

Related Links

[The control framework](#) on page 17

This guide assumes that you are using the Deployment Template with the Endeca Application Controller (EAC) as your primary control framework.

[Data and configuration workflow](#) on page 18

The diagram in this topic describes a basic data and configuration workflow. This workflow reflects the tasks performed when you run the Deployment Template.

[Logging and reporting workflow](#) on page 19

The diagram in this topic describes a logging and reporting workflow.

The control framework

This guide assumes that you are using the Deployment Template with the Endeca Application Controller (EAC) as your primary control framework.

By now you know that the typical Endeca implementation involves many different components (such as CAS, Workbench, Forge, Dgidx, Dgraph) that can run on different physical servers. Each such server also hosts the EAC Agent or the EAC Central Server. In other words, the EAC is a control system that manages these components on each physical server for the Endeca implementation and coordinates communication between them.

The Deployment Template allows you to start and run the components through the Endeca Application Controller (EAC) as well as run baseline and partial updates.

The EAC is a mandatory component of the Endeca implementation, while the Deployment Template is optional and is used to communicate with the EAC. While you can use other methods to communicate with the EAC, the Deployment Template provides a convenient framework and a set of scripts that simplify these tasks and is the recommended method for managing your implementation.

Related Links

[About the Endeca Application Controller](#) on page 12

The Endeca Application Controller (EAC) is a control system you can use to control, manage, and monitor components in your Endeca implementation.

[About the Deployment Template](#) on page 14

The Deployment Template provides a collection of operational components that serve as a starting point for development and application deployment.

[Options for running system operations](#) on page 48

You can run system operations using two alternative approaches—scripts that utilize the Deployment Template, or your own scripts provisioned in the EAC Admin Console of the Endeca Workbench.

[Options for provisioning the application](#) on page 47

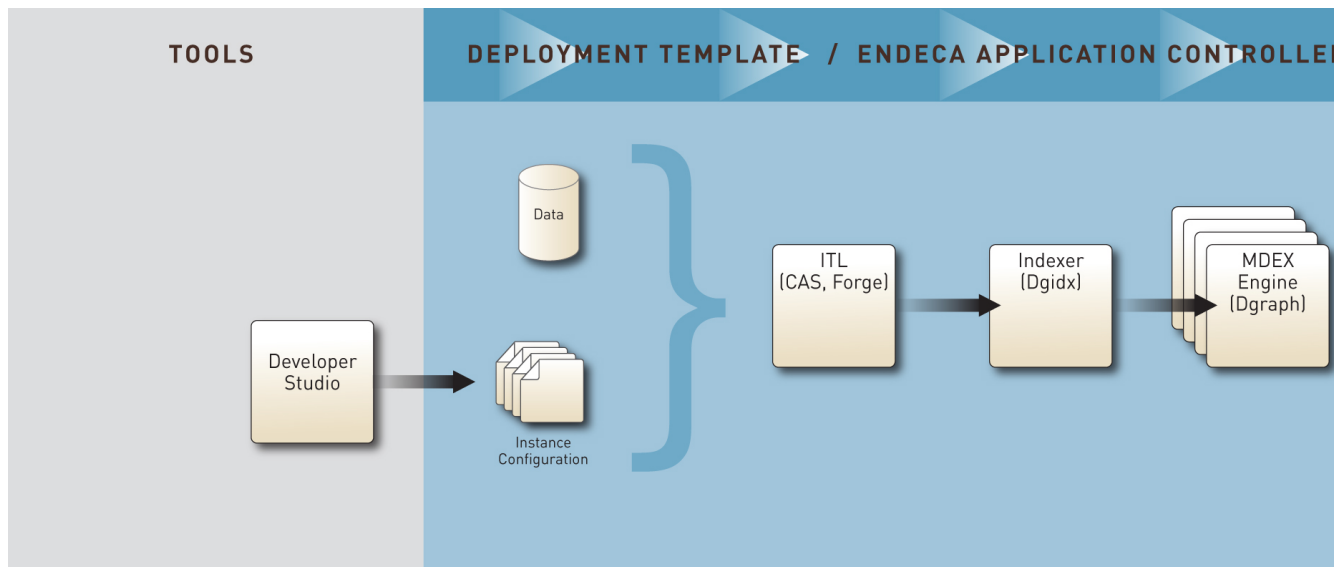
Provisioning is the task of defining the location and configuration of the Endeca resources (such as Forge, Dgidx and one or more Dgraphs) that control your Endeca application to the EAC.

Data and configuration workflow

The diagram in this topic describes a basic data and configuration workflow. This workflow reflects the tasks performed when you run the Deployment Template.

Basic workflow

The following diagram introduces the basic workflow that takes place when you run the Deployment Template:

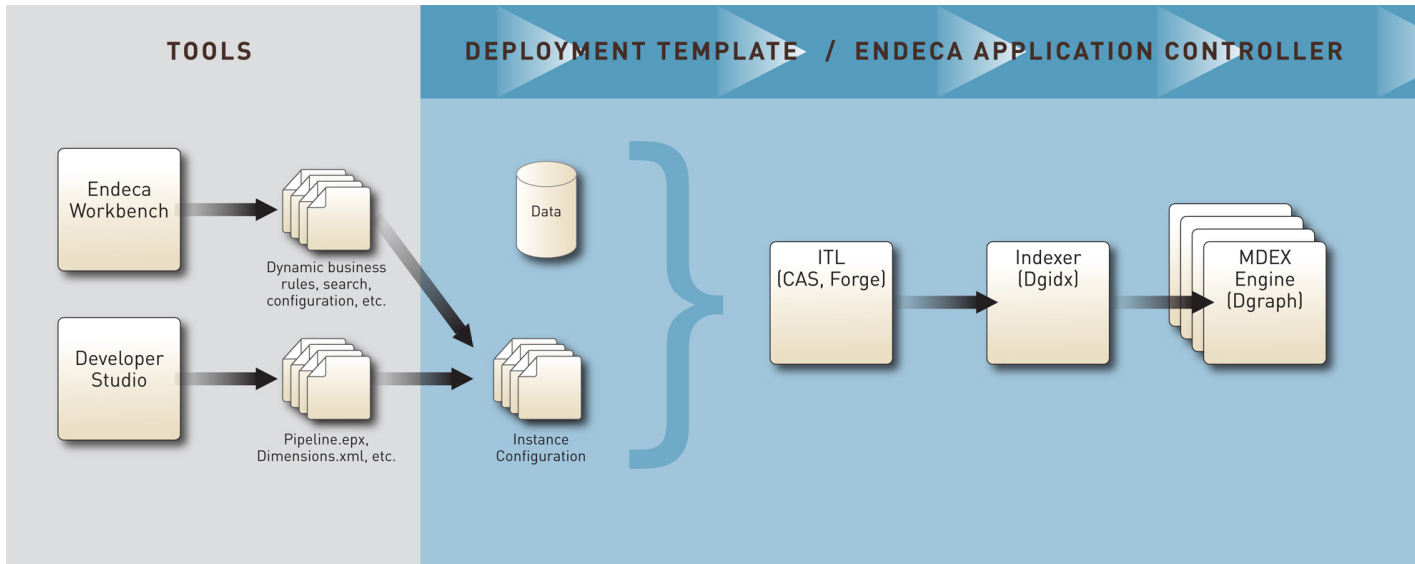


The Deployment Template creates the directory structure and adds the sample wine application data to the appropriate directories. (If you use your own data, the pipeline that you create in Developer Studio supplies the instance configuration files.)

Optionally, you can use the Content Acquisition System (CAS) to acquire the source data and prepare it for further processing in the pipeline. Next, the Deployment Template loads the data, provisions the application to the Endeca Application Controller (EAC) and runs the baseline update script. This ensures that the data is indexed by the MDEX Engine, which is now ready to process user queries.

Basic workflow with Endeca Workbench

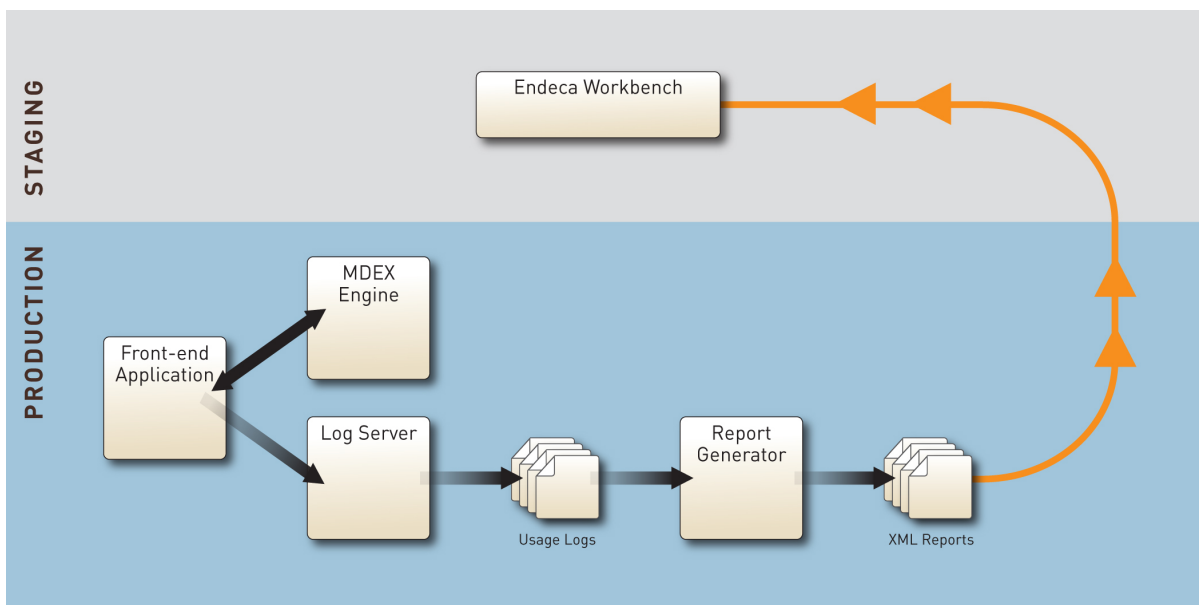
The following diagram introduces the basic workflow that takes place when you run the Deployment Template when the deployment includes Endeca Workbench:



In this diagram, the environment includes Endeca Workbench, which allows business users to change business rules, search configuration, thesaurus and other options.

Logging and reporting workflow

The diagram in this topic describes a logging and reporting workflow.



In this diagram:

- The front-end application uses the Endeca Logging API to communicate with the Log Server. (The front-end application uses the Endeca Presentation API, Web services and XQuery, or the RAD Toolkit for ASP.NET to communicate with the MDEX Engine.)
- Usage logs are generated for the Report Generator which creates XML reports for Endeca Workbench. Business users can use Endeca Workbench to analyze the reports and tune business rules search configuration settings accordingly.

For more information, see the *Log Server and Report Generator Guide*.

Related Links

[Refining the application based on production reports](#) on page 20

The diagram in this topic describes the logging and reporting workflow that takes place between staging and production environments. This process lets you obtain reports from the production environment and refine your Endeca application by changing the configuration in the staging environment and pushing it back to production.

[The logs created by the Dgraph](#) on page 68

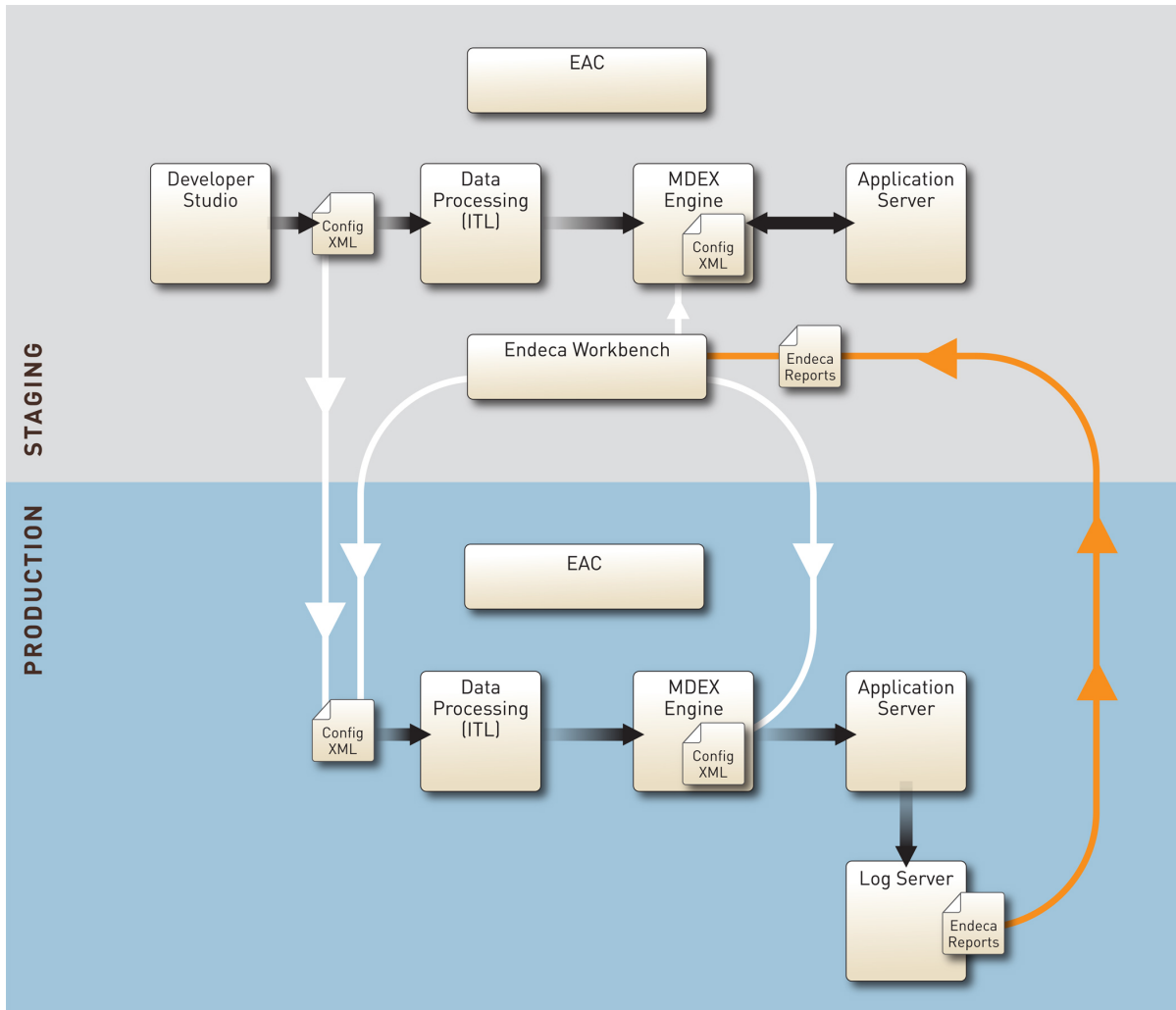
The Dgraph creates up to five logs, although some of these logs depend on your implementation and the Endeca components that you may be using. This topic provides a summary of these logs.

[Refining the application based on production reports](#) on page 20

The diagram in this topic describes the logging and reporting workflow that takes place between staging and production environments. This process lets you obtain reports from the production environment and refine your Endeca application by changing the configuration in the staging environment and pushing it back to production.

Refining the application based on production reports

The diagram in this topic describes the logging and reporting workflow that takes place between staging and production environments. This process lets you obtain reports from the production environment and refine your Endeca application by changing the configuration in the staging environment and pushing it back to production.



In this diagram, the following actions take place:

1. The project is established and runs in the staging environment.
2. The data and configuration information is pushed from staging to production. This is illustrated by arrows that go down from the staging to production environment.
3. The project is established and runs in the production environment.
4. The data from queries goes into the Logging Server, which generates Endeca Reports.
5. Endeca reports are consumed by Endeca Workbench. Based on this information, business users can make adjustments to the project configuration using Endeca Workbench.



Note: In addition to the Endeca reports that you can analyze in Endeca Workbench, query logs from the active running MDEX Engine in the production environment allow you to fine tune MDEX Engine performance in the staging environment and reproduce your changes on the MDEX Engine production servers. This cycle can be repeated to optimize performance.



Chapter 2

Creating Multiple Server Environments with the Deployment Template

Based on your project requirements, you can proceed to create staging and production environments. This typically means adding 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.

About a multiple server environment

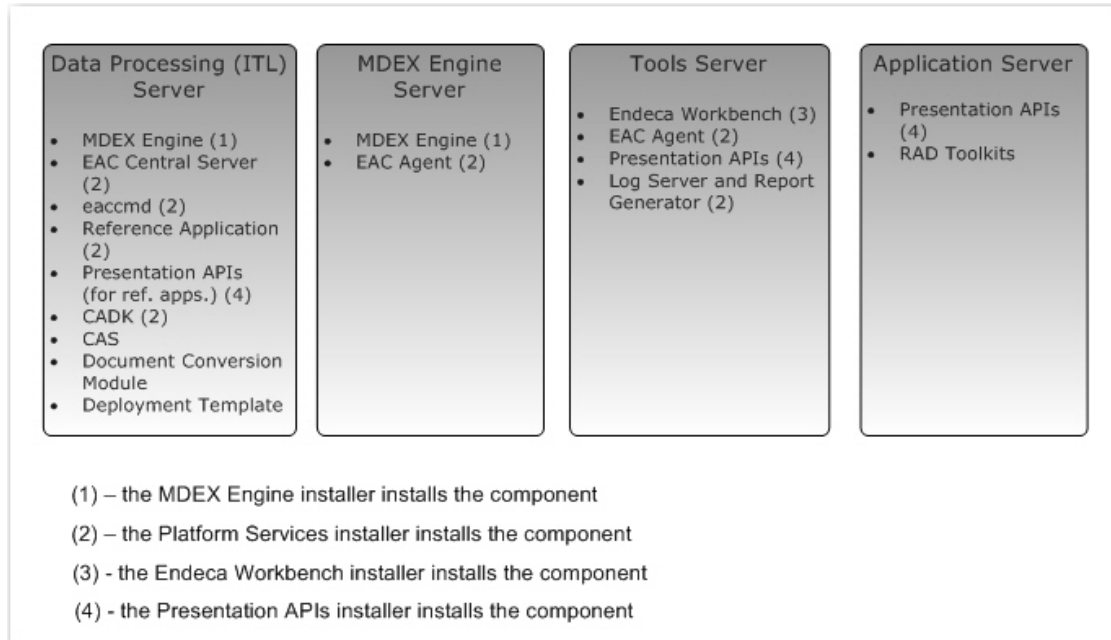
It is typical to establish a development environment (as well as staging and production environments) with multiple servers. The diagram in this topic illustrates which Endeca components must be installed in a multiple server environment.

In a multiple server environment, you can host:

- The MDEX Engine, the Platform Services package (which includes the EAC Central Server and Agent), the data for your application, and the Deployment Template on one server. This is the *Data Processing (ITL) server*.
- The MDEX Engine and the EAC Agent on one or more additional servers. These are the *MDEX Engine servers*.
- Endeca Workbench and the EAC Agent on a separate server. This is the *Tools server*.

You can also have a dedicated *Application server* to host the front-end application, or use one of the existing servers for this purpose. While you may or may not have an Application server in your development environment, in staging and production environments it is typical to set up one or more dedicated Application servers.

The following diagram illustrates which Endeca packages or their parts must be installed on the servers dedicated to the Endeca deployment:



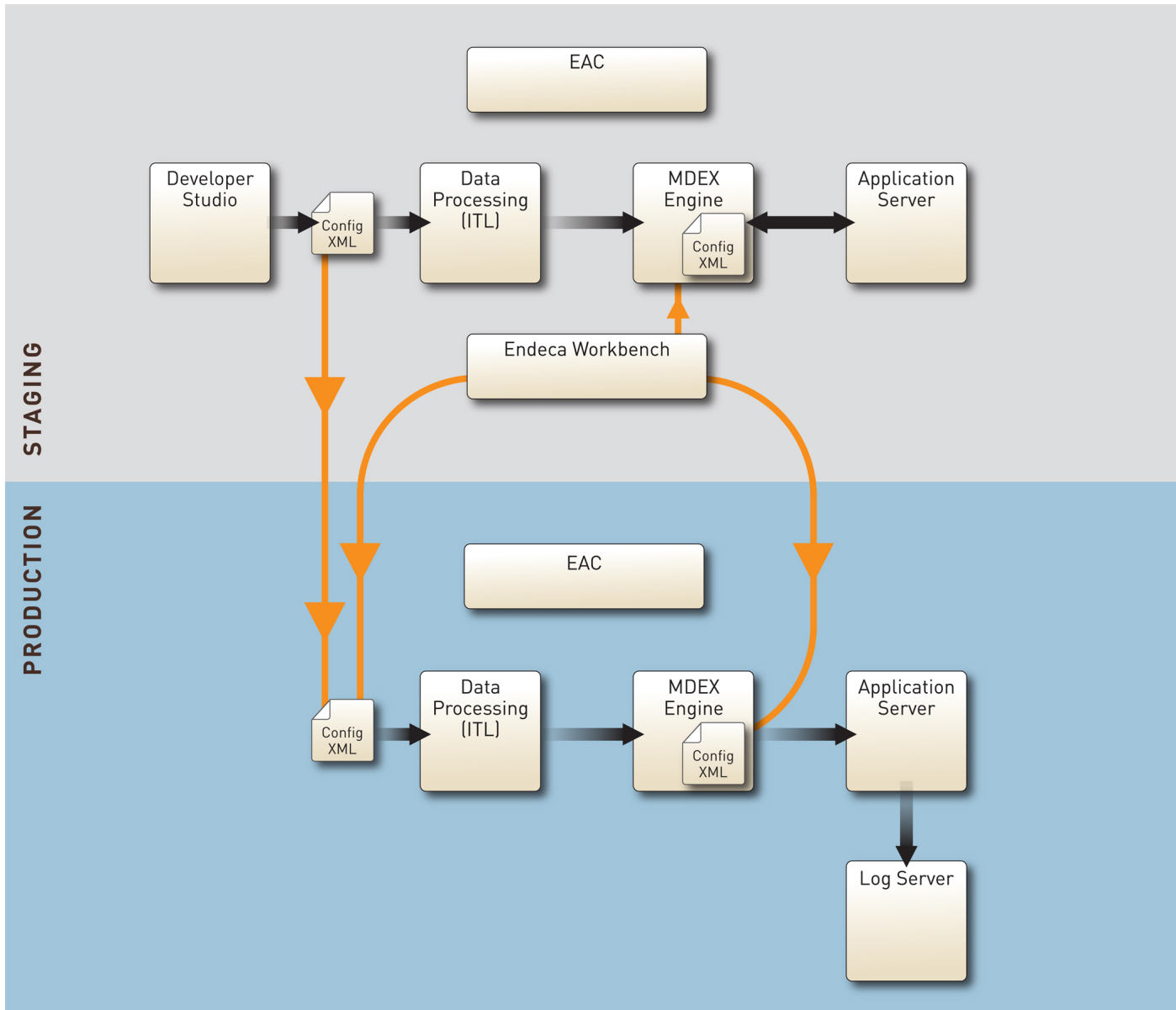
In this diagram:

- Each server plays a role in your Endeca environment and is represented as a single physical machine. However, you can configure more than one server for data processing (ITL), the MDEX Engine, and the front-end application.
- A Data Processing (ITL) server typically hosts the EAC Central Server, which is part of the Platform Services package.
- A Data Processing (ITL) server that runs on Windows also typically hosts Developer Studio.
- A Data Processing (ITL) server must host the Dgidx component that is part of the MDEX Engine installation package.
- An MDEX Engine server hosts the Dgraph component that is part of the MDEX Engine installation package.
- An MDEX Engine server and a Tools server must each host the EAC Agent, which is part of the Platform Services package.
- An Application server does not require the EAC Agent.

Overview of staging and production environments

While staging and production environments can have identical or very similar hardware setup, they differ from each other.

This diagram introduces the differences between staging and production environments and explains how they relate to each other:



In this diagram:

- The top portion of the diagram describes the data and configuration workflow within a staging environment. This workflow takes place when you create your pipeline, provision and initialize the application to the EAC, either with the Deployment Template or by using Endeca Workbench, and then run the baseline or partial update scripts. Next, the data is prepared for indexing and is processed by the MDEX Engine. The MDEX Engine receives queries from the front-end application and returns results to the front-end application on the Application server.
- The bottom portion of the diagram describes the data and configuration workflow within a production environment. This workflow takes place when you run updates on production servers and also enable the front-end application to send user queries to the MDEX Engine server for processing. Note that the production environment does not include Developer Studio and Endeca Workbench, because by this point, the configuration files and operational settings from the staging environment are replicated in the MDEX Engine running in the production environment.

- Most importantly, the arrows in this diagram that connect the staging and production environments describe what is involved in pushing your staging data into production. To push your project's data and configuration from staging to production, you typically perform these two high-level steps:
 1. Copy configuration files from the pipeline (Developer Studio) and Endeca Workbench (if you are using it) to the incoming directory on the Data Processing (ITL) server in the production environment from which Forge will consume it. You can accomplish this task by running the scripts within the Deployment Template.
 2. Promote configuration files (typically created in Endeca Workbench) to the MDEX Engine server in the production environment. This enables the MDEX Engine to use your project's configuration settings. The Configuration Manager of the Deployment Template lets you specify which files created in Endeca Workbench it needs to promote to the MDEX Engine server in the production environment.

Planning your 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 the *Deployment Template Usage Guide*.

To configure your server topology:

1. Analyze your goals for the performance of your Endeca application. Look at the projected size of the data set and other characteristics. For detailed information on hardware benchmarking and performance, see the *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.

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.

Related Links

[Adding Dgraphs](#) on page 33

You can add one or more additional Dgraphs on an already configured MDEX Engine server in your environment, by adjusting the `AppConfig.xml` file.

[Adding MDEX Engine servers](#) on page 30

You can add additional MDEX Engine servers to your environment by adjusting the Deployment Template `AppConfig.xml` file.

Configuring the application on multiple servers

To configure an application on multiple servers run the Deployment Template `deploy` script.

Before running the Deployment Template, verify that:

- On the Tools server, you have installed Endeca Workbench and the EAC Agent.
- On the Data Processing (ITL) server, you have installed the Platform Services (this includes the EAC Central Server and EAC Agent) and the MDEX Engine. The data is typically hosted on this server as well.
- On the Data Processing (ITL) server, you have downloaded the Deployment Template and set up a directory for your deployment, such as `C:\Endeca\apps` on Windows, or `localdisk/apps` on UNIX.
- On the MDEX Engine server, you have installed the MDEX Engine and the EAC Agent. (You can have multiple physical servers each running the MDEX Engine, or more than one MDEX Engine running on the same machine.)
- On all servers, the Endeca HTTP service is running. This starts the EAC.

To configure the application on multiple servers:

1. Go to the `C:\Endeca\Solutions\deploymentTemplate-<version>\bin` directory on Windows or `/usr/local/Endeca/Solutions/deploymentTemplate-<version>/bin` on UNIX and run the `deploy.bat` or `deploy.sh` script.

This script creates the project directories and configuration files.

2. Confirm the correct version of the Platform Services installation package (the template verifies the `ENDECA_ROOT` variable), and answer `Yes` to proceed.
3. Select the deployment type, `Dgraph`.
4. Specify the name of the application: `MyApp` and the location of the application directory: `C:\Endeca\apps` on Windows or `/localdisk/apps` on UNIX.



Note: In this guide, the directory for each of your applications is referred to by the `[appDir]` abbreviation. With the paths above, this is equal to `C:\Endeca\apps\MyApp` on Windows and `/localdisk/apps/MyApp` on UNIX.

5. Specify the EAC port (the Endeca HTTP service port) or accept the default port: `8888`
6. For **Enable IAP Workbench integration**, specify `Yes`.



Note: This configuration also applies to any Endeca Workbench edition.

7. Specify the Endeca IAP Workbench port (this is the Endeca Tools Service port for your Endeca Workbench edition) or accept the default port: `8006`.
8. Specify other necessary ports:
 - a) For the `Dgraph1`, specify the `Dgraph1` user query port or accept the default: `15000`
 - b) For the `Dgraph2`, specify the `Dgraph2` user query port or accept the default: `15001`
 - c) For the Endeca Logging and Reporting Server, specify the server port or accept the default: `15010`



Note: The Logging Server port number can be no larger than `32767`. If you plan to use the reference implementation and verify the Logging Server, you can set the Logging Server to run on port `15002` (for `Dgraph1`) or on port `15003` (for `Dgraph2`), and the reference implementation will work by default when connected to an MDEX Engine running on ports

15000 and 15001, respectively. These settings assume that the Logging Server runs on the same machine as the MDEX Engines. If you are using a different port for your Dgraph with the JSP reference implementation, specify a port equal to `Dgraph_port_number + 2`. This is because the Logging Server for the JSP reference implementation submits log entries to a port 2 above the Dgraph port.

Related Links

[Changing server settings in AppConfig.xml](#) on page 28

To accommodate a deployment on multiple servers, change the settings in the `AppConfig.xml` file of the Deployment Template. An example of changes to this file is included in this topic.

[Adding Dgraphs](#) on page 33

You can add one or more additional Dgraphs on an already configured MDEX Engine server in your environment, by adjusting the `AppConfig.xml` file.

[Adding MDEX Engine servers](#) on page 30

You can add additional MDEX Engine servers to your environment by adjusting the Deployment Template `AppConfig.xml` file.

[Additional customization tasks](#) on page 34

After you finish adjustments to the Deployment Template workflow so that it knows the location of your incoming data and correctly reflects the topology of the servers in your environment, you can start working on the baseline update script for your project. You can later run this script within the Deployment Template.

Changing server settings in AppConfig.xml

To accommodate a deployment on multiple servers, change the settings in the `AppConfig.xml` file of the Deployment Template. An example of changes to this file is included in this topic.

In a multiple server environment, the `[appDir]/config/script/AppConfig.xml` file should point to one or more MDEX Engine servers, a Data Processing (ITL) server, and a Tools server.

To change server settings in the `AppConfig.xml` file:

1. If you are using one or more separate servers to host one or more MDEX Engines, change the `MDEXHost` name to the name of the server in your deployment that runs each MDEX Engine.
2. Similarly, if you are using a separate Tools server, change the Endeca Workbench (referred to as "webstudio" in the `AppConfig.xml` file of the Deployment Template) host name to the name of the machine you are using for that server, and specify the port number on which the EAC agent is running on that machine.
3. Edit the rest of the `AppConfig.xml` file to change the other machine characteristics in a similar way. For example, for the Data Processing (ITL) server, change the name of the `ITLHost` and the port number on which the EAC agent is running on the ITL server. Change the "WebStudioHost" property in the `ConfigManager` section of this file.

You have adjusted the Deployment Template configuration to reflect the multiple server environment. Now you need to initialize the application.

Example of the changes to the AppConfig.xml file

The following example shows an abbreviated version of the [appDir]/config/script/AppConfig.xml file with the changes required to accommodate multiple servers:

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

# This file contains settings for an EAC application.
...
#####

# Global variables
#
-->
<app appName="myApp" eachHost="PlatformServicesServer.MyCompany.com" eachPort="8888"
...
</app>

<!--
#####

# Servers/hosts
...
-->
<host id="ITLHost" hostName="PlatformServicesServer.MyCompany.com"
port="8888" />
<host id="MDEXHost" hostName="MDEXEngineServer.MyCompany.com" port="8888"
/>
<host id="webstudio" hostName="ToolsServer.MyCompany.com" port="8888" >
  <directories>
    <directory name="webstudio-report-dir">./reports</directory>
  </directories>
</host>
.....
<!--
#####

# Config Manager.
...
-->
<custom-component id="ConfigManager" host-id="ITLHost"
class="com.endeca.soleng.eac.toolkit.component.ConfigManagerComponent">
  <properties>
    <property name="webStudioEnabled" value="true" />
    <property name="webStudioHost" value="ToolsServer.MyCompany.com" />
    <property name="webStudioPort" value="8006" />
    ....
  </properties>
  ....
<!--
#####

# Forge
#
-->
<forge id="Forge" host-id="ITLHost">
...

```

```

</forge>
....
<!--
#####

# Dgidx
#
-->
<dgidx id="Dgidx" host-id="ITLHost">
....
</dgidx>

<!--
#####

# Dgraph Cluster
#
-->
<dgraph-cluster id="DgraphCluster" getDataInParallel="true">
  <dgraph ref="Dgraph1" />
  <dgraph ref="Dgraph2" />
</dgraph-cluster>

....

<!--
#####

# Dgraphs
#
-->
<dgraph id="Dgraph1" host-id="MDEXHost" port="15000">
....
</dgraph>

<dgraph id="Dgraph2" host-id="MDEXHost" port="15001">
....
</dgraph>

....

<!--
#####

# LogServer
#
-->
<logserver id="LogServer" host-id="ITLHost" port="15010">
....

</logserver>

....

```

Adding MDEX Engine servers

You can add additional MDEX Engine servers to your environment by adjusting the Deployment Template `AppConfig.xml` file.

When you initially run the Deployment Template, it sets up an environment in which two Dgraphs are running on the same MDEX Engine server. You can optionally change this configuration and add one or more MDEX Engine servers, with one or more Dgraphs.

To add MDEX Engine servers:

1. Adjust the `[appDir]/config/script/AppConfig.xml` file and add `MDEX_Server2.MyCompany.com` as shown in the following example.

```
<!--
#####

# Servers/hosts
...
-->
<host id="ITLHost" hostName="PlatformServicesServer.MyCompany.com"
port="8888" />
<host id="MDEXHost1" hostName="MDEXServer1.MyCompany.com" port="8888"
/>
<host id="MDEXHost2" hostName="MDEXServer2.MyCompany.com" port="8888"
/>
<host id="webstudio" hostName="ToolsServer.MyCompany.com" port="8888"/
>
  <directories>
    <directory name="webstudio-report-dir">./reports</directory>
  </directories>
</host>
...

```

2. Add the Dgraph to the Dgraph cluster:

```
<!--
#####

# Dgraph Cluster
#
-->
<dgraph-cluster id="DgraphCluster" getDataInParallel="true">
  <dgraph ref="Dgraph1" />
  <dgraph ref="Dgraph2" />
</dgraph-cluster>
....

```

3. Point the Dgraph to the new host:

```
<!--
#####

# Dgraphs
#
-->
<dgraph id="Dgraph1" host-id="MDEXHost1" port="15000">
  ....
</dgraph>

<dgraph id="Dgraph2" host-id="MDEXHost2" port="15001">
  ....
</dgraph>

```

....

Example

In this example, an additional MDEX Engine server is specified, `<host id="MDEXHost2" hostName="MDEXServer2.MyCompany.com" port="8888" />`.

In addition, Dgraph2 is now pointing to MDEXHost2, as in: `<dgraph id="Dgraph2" host-id="MDEXHost2" port="15001">`.

In this configuration Dgraph1 runs on MDEXServer1.MyCompany.com and Dgraph2 runs on MDEXServer2.MyCompany.com.



Note: The ports specified in the MDEXHost1 and MDEXHost2 definitions are the ports on which the EAC Agent is running on those machines, not the MDEX Engine query ports.

This example shows an abbreviated version of the file and highlights the required changes:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
#####
# This file contains settings for an EAC application.
...
#####
# Global variables
#
-->
<app appName="MyApp" eachHost="PlatformServicesServer.MyCompany.com" eachPort="8888"
...
</app>

<!--
#####
# Servers/hosts
...
-->
<host id="ITLHost" hostName="PlatformServicesServer.MyCompany.com"
port="8888" />
<host id="MDEXHost1" hostName="MDEXServer1.MyCompany.com" port="8888" />

<host id="MDEXHost2" hostName="MDEXServer2.MyCompany.com" port="8888" />

<host id="webstudio" hostName="ToolsServer.MyCompany.com" port="8888" />

    <directories>
        <directory name="webstudio-report-dir">./reports</directory>
    </directories>
</host>
...

<!--
#####
# Dgraph Cluster
#
```



```

-->
<dgraph-cluster id="DgraphCluster" getDataInParallel="true">
  <dgraph ref="Dgraph1" />
  <dgraph ref="Dgraph2" />
</dgraph-cluster>

....

<!--
#####

# Dgraphs
#
-->
<dgraph id="Dgraph1" host-id="MDEXHost1" port="15000">
  ....
</dgraph>

<dgraph id="Dgraph2" host-id="MDEXHost2" port="15001">
  ....
</dgraph>

....

```

Now that you have added an additional MDEX Engine server, you can add an additional Dgraph on any of these servers.

Adding Dgraphs

You can add one or more additional Dgraphs on an already configured MDEX Engine server in your environment, by adjusting the `AppConfig.xml` file.

When you initially run the Deployment Template, it sets up an environment in which two Dgraphs are running on the same MDEX Engine server. You can optionally change this configuration and add or remove Dgraphs from this server.

To add a Dgraph to an MDEX Engine server:

1. Add a line similar to the following `<dgraph id="Dgraph3" host-id="MDEXHost" port="15001">` to the Dgraphs portion of the `[appDir]/config/script/AppConfig.xml` file.
2. Make changes to the Dgraph Cluster block and the Dgraphs block, as shown in the following example. Add any new Dgraphs that you have provisioned to this block in order for a copy of the index to be pushed out to them during updates. In this abbreviated example of `AppConfig.xml`, three Dgraphs are configured on the same MDEX Engine server:

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
#####
# This file contains settings for an EAC application.
...
#####
# Global variables
#
-->
<app appName="MyApp" eachHost="PlatformServicesServer.MyCompany.com"

```

```

eacPort="8888"
...
</app>

<!--
#####
# Servers/hosts
...
-->
<host id="ITLHost" hostName="PlatformServicesServer.MyCompany.com"
port="8888" />
<host id="MDEXHost" hostName="MDEXServer.MyCompany.com" port="8888" />

<host id="webstudio" hostName="ToolsServer.MyCompany.com" port="8888"
>
  <directories>
    <directory name="webstudio-report-dir">./reports</directory>
  </directories>
</host>
...
<!--
#####
# Dgraph Cluster
#
-->
<dgraph-cluster id="DgraphCluster" getDataInParallel="true">
  <dgraph ref="Dgraph1" />
  <dgraph ref="Dgraph2" />
  <dgraph ref="Dgraph3" />
</dgraph-cluster>
....
<!--
#####
# Dgraphs
#
-->
<dgraph id="Dgraph1" host-id="MDEXHost" port="15000">
  ....
</dgraph>

<dgraph id="Dgraph2" host-id="MDEXHost" port="15001">
  ....
</dgraph>
<dgraph id="Dgraph3" host-id="MDEXHost" port="15007">
  ....
</dgraph>
....

```

Additional customization tasks

After you finish adjustments to the Deployment Template workflow so that it knows the location of your incoming data and correctly reflects the topology of the servers in your environment, you can start working on the baseline update script for your project. You can later run this script 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 the Deployment Template customization and capabilities, see the *Deployment Template Usage Guide*.



Chapter 3

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 using the Deployment Template

The ability to replicate application definitions across environments helps organizations support a well-structured application life cycle, where applications can be developed, tested, deployed, modified, re-tested and re-deployed with the least effort.

Replicating application definitions also simplifies administrative processes like the backup and recovery of Endeca applications.

This section discusses the planning and procedures that you can use to replicate applications, even the target environments that have very different characteristics.

Endeca administrators and developers may want to use the same application definition in several hardware and software environments:

- A primary development environment, where the application configurations and files are created and maintained.
- A staging and test environment, where applications are tested and tweaked.
- A production environment.
- Secondary development environments, where the application definition can be re-used as the basis for new applications.
- A backup or disaster recovery environment, where the application can be redeployed and restarted in the event of a problem with the main production environment.

These environments may have very different characteristics, including variations in the numbers and types of servers, operating systems, and network architectures. 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 diverse systems for the ITL server, the MDEX Engine, and multiple application servers and log servers.

Replicating any type of application across diverse environments is challenging. Fortunately, with planning, some scripting, and the use of certain procedures, you can automate most of the process of replicating Endeca application definitions across environments with different characteristics.

Part of the solution is to develop with the Endeca Deployment Template. As described in the previous section, *Creating Multiple Server Environments with the 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.

But other procedures are necessary to successfully replicate application definitions between environments.

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).

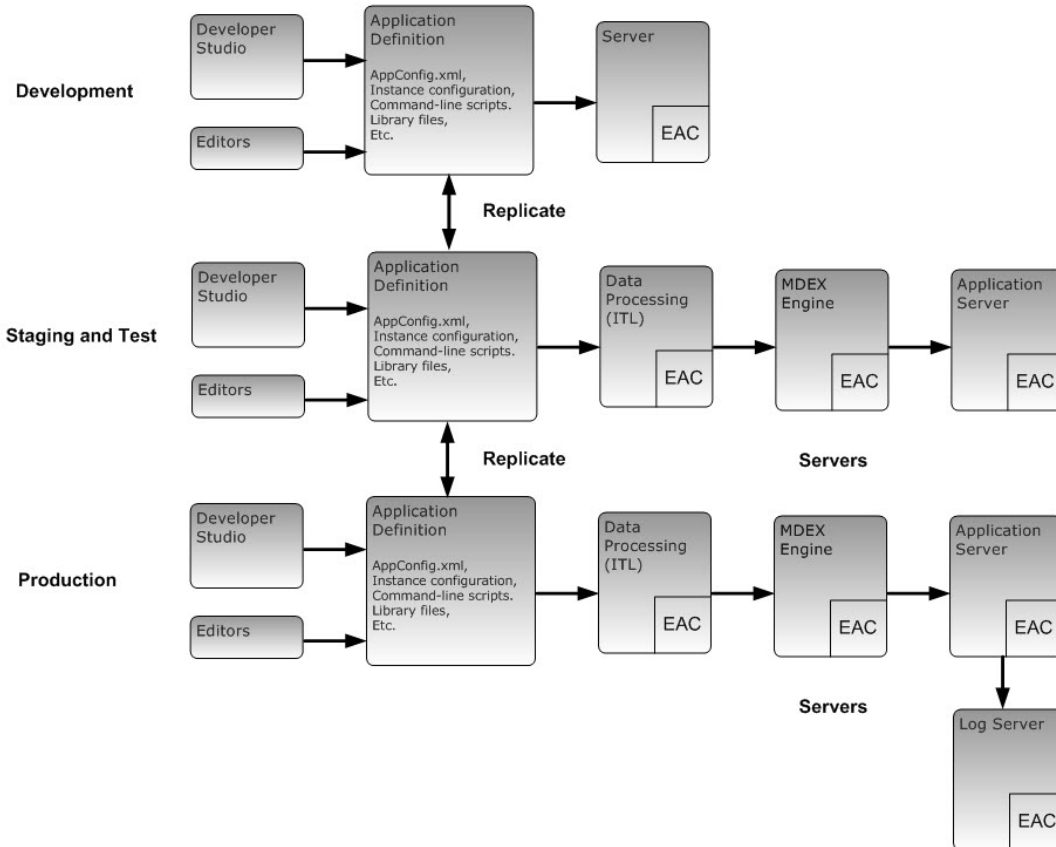
The administrator is developing with the Deployment Template, so when a new or modified application definition is moved to the EAC Central Server in each environment, the EAC takes care of propagating the necessary parts of the application definition across the servers in that environment.

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.



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.

Identifying the artifacts that make up an application

The artifacts that make up a typical Endeca application definition include the following:

The AppConfig.xml file

The `AppConfig.xml` file describes each of the application's provisioning information and is stored in the EAC Central Server. The Deployment Template control scripts use `AppConfig.xml` as the authoritative source for application definition. The Deployment Template stores a copy of the `AppConfig.xml` file in the `[appdir]/config/script` directory.

Although you can modify an application configuration with the Workbench, we recommend that modifications only be made in the `AppConfig.xml` file. That way, the application configuration will be saved on disk, ready for sharing between environments. You can use the Workbench for other

tasks that do not involve modifying the configuration, such as reviewing the configuration, and starting or stopping individual components.



Note: 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 more information about how to create this file, see the topic about *Creating a custom file for environment-specific settings*.

The instance configuration

The instance configuration is a set of files that control the ITL process and the data loaded into the MDEX Engine servers. The instance configuration files are controlled by the Developer Studio, and optionally by the Workbench.

These files include configuration data such as dimension definition, search configuration, the Forge pipeline, and Page Builder landing pages.

Page Builder templates

Page Builder templates are used to drive dynamic landing pages that can be created in Page Builder. They are defined by xml files stored by the Workbench, and accessed through the `emgr_update` command utility.

Command-line scripts

An application deployment typically includes command-line scripts that perform common operations related to the application's functionality. By convention, these scripts are stored in the Deployment Template's `[appdir]/control` directory.

The Deployment Template includes scripts such as `baseline_update` and `set_baseline_data_ready_flag`.

You can create additional scripts under the `[appdir]/control` directory. These scripts, together with their input data and output directories, are a part of the application definition. For example, a developer might create scripts to crawl web pages in preparation for a baseline update. These scripts might take as input a seed-list file, and create an output file in a custom directory under `[appdir]`.

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

Library files

Many parts of an application use library files. For example, a Forge pipeline using a Java or Perl manipulator typically requires access to library files implementing those manipulators. BeanShell scripts may use application- or Endeca-specific Java classes. By convention, library files are kept under `[appdir]/config/lib`.

Forge state files

Forge state files reside in the `[appdir]/data/state` directory.

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 the 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 dynamic business rules, Page Builder landing pages, explicit dimension ordering, or in precedence rules).

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" eachHost="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 Web Studio 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>

```

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 will 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 [appdir]/data/processing, which contains the content of [appdir]/data/incoming before the pipeline is run. Therefore any files from [appdir]/data/incoming can be referenced directly by name, without specifying a directory.

Java manipulator paths are relative to [appdir]. 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 [appdir].

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 [appdir]/control. If this script uses paths relative to [appdir]/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 [appdir], 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 [appdir]. 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.

Automating the collection of files

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

To create a script that collects these files:

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 [appdir] directory.

Running the script results in collecting and storing all of the artifacts in subdirectories under [appdir], 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 [appdir] 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.

Distributing application definitions between environments for the first time

After you collect all the files and artifacts for an application definition and store them in a central location, you can distribute them by copying them to the [appdir] directory on the EAC Central Servers of the target environment. A few additional steps are required when an application is being deployed to an environment for the first time.

To replicate an application definition the *first time* an environment is provisioned, for example when a new application is first moved from development to staging, or from staging to production:



Note: These steps assume that your environment operates under a version control system.

1. On the EAC Central Server of the source environment, run the `collect-app.bat` script to store all of the application configuration artifacts in `[appdir]` directory.
2. Commit the contents of the `[appdir]` directory.
3. In the target environment, create the `[appdir]` directory on the ITL Server.
4. Copy the contents of the `[appdir]` directory of the source environment into the `[appdir]` directory of the target environment.
5. Edit the environment-specific parts of the definition and supply the correct values for this environment. For more information, see the *Using techniques to ensure interoperability across environments* topic.
6. Edit the environment-specific parts of the definition and supply the correct values for this environment. For more information, see the *Using techniques to ensure interoperability across environments* topic.
7. On the EAC Central Server of the target environment, run the Deployment Template's `initialize_services` script in the `[appdir]/control` directory.
8. On the EAC Central Server of the target environment, run the Workbench `upload.bat` script to ensure that the target environment's Workbench is updated.

Distributing an updated application definition with another environment

To distribute an updated application definition with another environment:



Note: These steps assume that your environment operates under a version control system.

1. On the EAC Central Server of the source environment, run the `collect-app.bat` script to store all of the application configuration artifacts in the `[appdir]` directory.
2. Commit the content of the relevant `[appdir]` subdirectories.
3. Update the `[appdir]` directory on the EAC Central Server in the target environment.
4. On the EAC Central Server of the target environment, run the `upload.bat` script to ensure that the target environment's Workbench is updated.

Approaches to avoiding synchronization conflicts

In some situations, several users might be modifying an application definition concurrently. For example, several application developers might be working on the same application in their own development environments, or business users might be changing application configurations in the staging environment while developers are making other changes in the development environment. To avoid synchronization conflicts, review the following approaches.

A token system

Token systems prevent synchronization conflicts by allowing only one person in an environment to modify an application configuration. Only the person holding the "token" can make changes. The token

is passed from one environment to another during synchronization: the target environment acquires the token from the source environment when the application definition is propagated. Any changes performed before obtaining the token will be run over without warning.

Token systems are simple, because it is easy to tell which environment is entitled to make changes at any given moment, and any part of the application is allowed to change in the token-holding environment. However, a token system prevents concurrent development.

Separation of concerns

A separation of concerns method restricts the scope of changes allowed in an environment. Each environment is concerned with a distinct part of the configuration. For example, one environment may be in charge of Page Builder templates, while another is in charge of dimension definitions. Changes to the area of concern can only be made in the appropriate environment. Therefore changes made in one environment never overwrite changes made in another.

Separation of concern methods facilitate concurrent development, but the separation of concerns must be well defined in advance and enforced consistently.

Manual resolution

Conflicts arising from unrestricted concurrent changes in multiple environments can be resolved by manually comparing all the conflicting files and treating each difference individually.

However, manual resolution processes are time-consuming, require extensive knowledge of Endeca components and internal formats, are prone to human error, and are not scalable. Therefore, you should not consider manual resolution for synchronizing Endeca environments.



Chapter 4

Performing System Operations with the EAC

This section discusses operational tasks related to your Endeca application. It focuses on the recommended practice of using the Deployment Template to interact with the EAC, but mentions alternative methods of performing some tasks. It also lists essential administrative information related to EAC operations, such as pointers to EAC logs.

Options for provisioning the application

Provisioning is the task of defining the location and configuration of the Endeca resources (such as Forge, Dgidx and one or more Dgraphs) that control your Endeca application to the EAC.

You can provision an application in three ways:

Method	Description
By running the <code>initialize_services</code> script of the Deployment Template	This is the easiest way to provision an application. For information, see this guide, along with the <i>Deployment Template Usage Guide</i> .
Using Endeca Workbench	For information, see the <i>Endeca Workbench Help</i> .
Directly to the EAC using the <code>eaccmd</code> tool	For information, see the <i>EAC Guide</i> .

Related Links

[Updating the application provisioning](#) on page 48

When you make changes to the `AppConfig.xml` file, the application provisioning in the Deployment Template configuration must be updated as well. You can update the application provisioning definition in the Deployment Template's environment without having to run a baseline update.

[Backing up the EAC application provisioning with `eaccmd`](#) on page 48

If you use `eaccmd` (and not the Deployment Template), to back up an existing application's provisioning, you can run the `eaccmd describe-app` command and redirect its output to a file. This is a useful technique for testing, since it allows you to easily revert your application to its original provisioning if you make unwanted changes.

Updating the application provisioning

When you make changes to the `AppConfig.xml` file, the application provisioning in the Deployment Template configuration must be updated as well. You can update the application provisioning definition in the Deployment Template's environment without having to run a baseline update.

To update the `AppConfig.xml` definition in the Deployment Template configuration:

Run the `runcommand` script with `--update-definition` option as follows:

Option	Description
Windows	<code>C:\apps\myApplication\control\runcommand.bat --update-definition</code>
UNIX	<code>/apps/myApplication/control/runcommand.sh --update-definition</code>

The `--update-definition` option updates the application provisioning.

Backing up the EAC application provisioning with `eaccmd`

If you use `eaccmd` (and not the Deployment Template), to back up an existing application's provisioning, you can run the `eaccmd describe-app` command and redirect its output to a file. This is a useful technique for testing, since it allows you to easily revert your application to its original provisioning if you make unwanted changes.

It is possible to use the `eaccmd describe-app` command for this purpose, because the output of this command is in the same XML format used by the `eaccmd define-app` command.

For more information on the `eaccmd` commands see the *"Provisioning commands"* section in the *"Using the Eaccmd Tool"* chapter of the *EAC Guide*.

To back up provisioning for an existing application with `eaccmd`:

Run the following command: `eaccmd describe-app --app application_name -canonical>application.xml`

This command writes the XML to a file that can easily be used later by the `define-app` command. The `eaccmd describe-app -canonical` command has a required `-app` parameter that takes the name of the application to be described. 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. This makes reading the application description significantly easier.

You can later use the resulting application description file as a definition file for the `eaccmd define-app` command.

Options for running system operations

You can run system operations using two alternative approaches—scripts that utilize the Deployment Template, or your own scripts provisioned in the EAC Admin Console of the Endeca Workbench.

This guide assumes that you use the Deployment Template as a primary means of running operational tasks in your Endeca implementation, and provides basic information about the EAC.

Method	Description
Customized scripts within the Deployment Template environment	<p>The sample scripts provided with the Deployment Template control the Endeca operational tasks through the EAC. You can use these scripts, or create custom scripts based on them.</p> <p>The scripts typically run such processes as routine baseline and partial updates. You can also add specific scripts that run before the Dgraph is stopped or after it is started.</p>
Your own Java scripts within the Endeca Workbench environment	<p>You can create your own operational scripts for running baseline and partial updates, and for other purposes, and provision them in the Endeca Workbench for running from there.</p> <p>Your scripts should communicate with the EAC Central Server which controls the processes on each of your servers.</p> <p>If you would like to use your own custom Java scripts that directly communicate with the EAC and need more information on how to write them, see the <i>EAC Guide</i>.</p>

Checking the status of EAC components

You can use several methods to check the status of your application's components that are provisioned to the EAC, such as Dgidx, Forge, and Dgraph.

To check the status of EAC components, use one of these methods:

- **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 <app_id> --comp <comp_id>
```



Note: You can also substitute the `--comp <comp_id>` argument with the `--script <script_id>` argument to confirm the status of an EAC script.

- **The Deployment Template.** Run the following command at a command line:

```
Windows:
<PROJECT_DIR>\control\runcommand.bat --print-status
UNIX :
<PROJECT_DIR>/control/runcommand.sh --print-status
```

This commands prints the status for each component in the current application.

- **Your application.** You can build your application to query EAC via programmatic Web service calls. For more details about using the `GetComponentStatus()` API method, see the "*Endeca Application Controller API Interface Reference*" section of the *Endeca Application Controller Guide*.

Avoiding defunct EAC processes

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 child processes in a `<defunct>` state.

To avoid defunct EAC processing, consider the following recommendations:

- For a Dgraph or Agraph, 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.

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 *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.

-

Deployment Template and Endeca Workbench interaction

This topic summarizes information about Deployment Template and Endeca Workbench interaction.

If you use Endeca Workbench in your environment in addition to the Deployment Template, consider the following points that summarize their interaction:



Note: This topic provides a summary of interaction between these two components and lists recommendations for administrative practices. For detailed information on how to implement both Endeca Workbench and the Deployment Template in staging and production environments, see the *Deployment Template Usage Guide*.

- **Manage changes to the operational environment through the Deployment Template.**

Endeca strongly recommends managing changes to your EAC component configuration, such as command line arguments to Forge, Dgidx and Dgraph, through the Deployment Template (once you choose to use it for your operational tasks), and not through EAC Admin Console in Endeca Workbench.

- **Manually upload changes to the instance configuration to Endeca Workbench.**

If, after you have initially deployed your application, you make changes to the instance configuration files, such as add a new dimension, or define a new zone for dynamic business rules, and then run the Deployment Template scripts for baseline and partial updates, the Deployment Template does not automatically upload changed configuration files and settings to Endeca Workbench.

You can manually upload configuration updates to Workbench through a Deployment Template script, when needed. This may be necessary when changes to dynamic business rule zones and styles are updated, or dimensions are added or removed, and Workbench needs to be updated to allow business users to maintain rules based on the updated configuration.

Typically, it is useful to synchronize changes with Workbench before your Endeca implementation is deployed in the production environment, or during updates to your Endeca implementation. Use the `update_web_studio_config.[sh|bat]` script to deploy these pipeline changes through Workbench.

Calling `update_web_studio_config.[sh|bat]` requires all locks in Workbench to be available. This means all users must be logged out of the system and not holding any locks on resources. When you do run `update_web_studio_config.[sh|bat]`, you are responsible for resolving any locking issues via the Workbench interface.

- **Manage some settings in Endeca Workbench.**

In general, once you use the Deployment Template framework, its scripts become your designated environment through which you manage all changes to the EAC components and instance configuration files.

However, if you also use Workbench for managing rules, keyword redirects, search, dimension order, and reports, you can enable the Configuration Manager in the Deployment Template. The Configuration Manager informs the Deployment Template that for these changes, the Deployment Template should use Endeca Workbench, and not Developer Studio.



Note: The Deployment Template does not actually create, modify, or manage configuration files and operational settings; Developer Studio and Endeca Workbench perform these functions. However, when running its scripts, the Deployment Template needs to know which settings and files to use: from Developer Studio (this is the default assumption), or from Workbench. To decide whether changes that originated from Developer Studio or Workbench should be used, the Deployment Template uses its Configuration Manager. However, if you specify in the Configuration Manager the files as being maintained by Endeca Workbench, you ensure the Developer Studio version of these files is not used in scripts that run within the Deployment Template framework.

To summarize, if you prefer to maintain some changes through Endeca Workbench, enable the Configuration Manager component to ensure that these Workbench-maintained configuration files are used when running the scripts from the Deployment Template. For details on using this component, see the "Application Configuration" chapter of the *Deployment Template Usage Guide*.

Related Links

[Releasing locks set by the Deployment Template in the EAC](#) on page 53

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.

Archiving the Dgraph log files

Your Dgraph files are archived automatically once you run a baseline update script with the Deployment Template. In addition, 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`

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 Endeca 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:	<p>Go to the <code>.<AppDir>/control/</code> directory.</p> <p>Run the following Deployment Template command:</p> <p>On Windows: <code>.\runcommand.bat LockManager releaseLock update_lock</code></p> <p>On UNIX: <code>./runcommand.sh LockManager releaseLock update_lock</code></p>
Using the <code>eaccmd</code> tool:	<p>Windows: <code>eaccmd.bat remove-all-flags -app <your application></code></p> <p>UNIX: <code>eaccmd.sh remove-all-flags -app <your application></code></p> <p>The LockManager is internally using EAC flagging functionality, so the <code>remove-all-flags</code> function of <code>eaccmd</code> effectively cleans the Deployment Template locks.</p>


This releases the locks in the EAC.

Removing components from your configuration

To remove inactive components you can use several options — stop and remove them in the Deployment Template, run `initialize_services`, use `eaccmd`, or use the EAC Console of Endeca Workbench.

To stop and remove a component from the EAC definition:

Use any of the following options:

Option	Description
Use the Deployment Template <code>runcommand</code>.	<p>The <code>runcommand</code> of the Deployment Template stops the components if they are running and removes their definition. After that, you can remove the components from <code>AppConfig.xml</code>.</p> <ol style="list-style-type: none"> 1. If the component is running, stop it: <code>[appDir]/control/runcommand <i>component_name</i> stop</code> 2. Remove the component's definition: <code>[appDir]/control/runcommand <i>component_name</i> removeDefinition</code> 3. Remove the inactive component from the <code>AppConfig.xml</code> file.
Run the Deployment Template <code>initialize_services</code> script.	<p>This script resets your provisioning to components specified in <code>AppConfig.xml</code>. It stops any running components (such as Dgraphs or log servers), removes the application from the EAC, and provisions the entire application again to the EAC.</p> <p> Important: Re-running the <code>initialize_services</code> script will wipe out any configuration that exists in Endeca Workbench, such as configuration of rules (if you have created them in Workbench).</p>
Use the EAC Admin Console of the Endeca Workbench.	<p>The Endeca Workbench communicates with the EAC, stops and removes the component.</p>
Use the <code>eaccmd</code> tool to stop and remove the component.	<p>The <code>eaccmd</code> tool stops and removes the inactive components from the EAC.</p> <ol style="list-style-type: none"> 1. Stop the component: <ul style="list-style-type: none"> • On Windows: <code>eaccmd.bat host:port stop --app <appname> --comp <component id></code> • On UNIX: <code>eaccmd.sh host:port stop --app <appname> --comp <component id></code> 2. Remove the component: <ul style="list-style-type: none"> • On Windows: <code>eaccmd.bat host:port remove-component --app <appname> --comp <component id></code> • On UNIX: <code>eaccmd.sh host:port remove-component --app <appname> --comp <component id></code>

Related Links

[Removing components in a Deployment Template environment](#) on page 56

This topic explains how the Deployment Template treats components that have been removed from its `AppConfig.xml` file. Incremental provisioning in the Deployment Template updates existing components, but does not remove components that are not defined in the `AppConfig.xml`.

Removing components in a Deployment Template environment

This topic explains how the Deployment Template treats components that have been removed from its `AppConfig.xml` file. Incremental provisioning in the Deployment Template updates existing components, but does not remove components that are not defined in the `AppConfig.xml`.

In general, running `initialize_services` removes an application from both the EAC and the Endeca Workbench store, and re-provisions the application, so at that point you can be sure that the EAC has only the components provisioned in the Deployment Template.

When the Deployment Template performs its provisioning check, it verifies that all of the components currently listed in the `AppConfig.xml` are provisioned and up-to-date with the EAC copy of the components. Any new or modified components are provisioned or have their provisioning updated in the EAC at this time.

However, the Deployment Template does not consider components that are not declared in `AppConfig.xml` (including ones that were declared there at some point but since removed). This means that components that are provisioned in the EAC (including components previously provisioned with the Deployment Template) may remain—the Deployment Template supports an environment where it is not the only tool interacting with the EAC.

Example scenario

Following the provisioning step, whenever the Deployment Template runs a baseline update, it checks whether the definition in the `AppConfig.xml` file has changed compared to what is stored in the EAC Central Server. It picks up changes such as new flags or additional components.

However, if you modify the baseline update script by removing a Dgraph in the `AppConfig.xml` file, the `baseline_update` script honors the change, but does not affect the provisioning configuration stored in the EAC Central Server. It does issue a message about the discrepancy when running the script. The removed Dgraph continues running, even though it is no longer listed in the `AppConfig.xml` file.

Determining the state of the EAC with service URLs

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`

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 1G size limit, the system starts to write to `main.1.log`. Once `main.1.log` reaches the 1G 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 <code>main.0.log</code>	<p>Most EAC logging goes into this log file. For example, exceptions thrown when invoking a shell or component are logged in it.</p> <p>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"</p>
Process log, such as <code>process.0.log</code>	<p>Logs generated by the EAC process control module go into this file. This log contains messages associated with process control and recovery.</p> <p>These messages include information about starting and stopping scripts, components and utilities, recovering failed processes and rebinding to active processes.</p>
Invocation log, such as <code>invoke.0.log</code>	<p>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.</p>
Tomcat/Catalina logs, such as: <ul style="list-style-type: none"> • <code>catalina.out</code> • <code>catalina.[date].log</code> • <code>tomcat_[stdout/stderr].log</code> 	<p>These logs are useful when errors occur while loading the EAC.</p> <p>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.</p> <p>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"</p>

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 MDEXHost 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 MDEXHost.
2. Change the `networkaddress.cache.ttl` in the `java.security` file to 0 on both machines. This file is located by default in `%ENDECA_ROOT%\j2sdk\jre\lib\security` on Windows and `$ENDECA_ROOT/j2sdk/jre/lib/security` on UNIX.
3. Change the IP address of the EAC Central Server, making sure that the MDEXHost operating system can resolve the EAC Central Server at the new IP address.
4. Restart the Endeca HTTP service on both machines.
5. Optionally, you may go back to the `java.security` file mentioned in step 2 above, change the `networkaddress.cache.ttl` back to -1, and restart the Endeca HTTP service on both machines to avoid subsequent DNS spoof attacks.



Administering 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.

Dgidx processing and memory usage

Dgidx is the component of the MDEX Engine that organizes the acquired records into a structure, partially precomputes some results that will be 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 *Performance Tuning Guide*.

Related Links

[Variations in Dgidx indexing time](#) on page 66

When you analyze Dgidx logs, you may notice that periodically indexing times are longer than you might expect.

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	<code>runcommand.bat MyDgidx run</code>
UNIX	<code>./runcommand.sh MyDgidx run</code>

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.

Related Links

[Running the Dgidx binary at the command prompt](#) on page 60

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.

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, Endeca 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	<code>%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</code>
UNIX	<code>\$(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</code>

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 `prtcnf` 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.

Related Links

[Dgidx logs](#) on page 63

To locate your application's Dgidx logs, consult the Dgidx definition in the `AppConfig.xml` file of the Deployment Template.

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, Endeca 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 Description: 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.

Related Links

[Dgidx log details for text search indexing](#) on page 64

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.

[Dgidx handling of records with missing or duplicate record spec values](#) on page 65

When Dgidx processes records with missing or duplicate record specifier (or spec) values, it completes successfully, but produces a very large log file.

[Variations in Dgidx indexing time](#) on page 66


When you analyze Dgidx logs, you may notice that periodically indexing times are longer than you might expect.

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 <code>recsearch_indexes.xml</code> 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[0xcelbd0] "Ripe"
Property["P_DateReviewed"]: Value[0xcel470] "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[0xcel870] "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.



Chapter 6

Administering the Dgraph

This section describes basic administrative tasks for the Dgraph. It also contains Dgraph troubleshooting tips, and describes Dgraph and Agraph logs.

Checking Dgraph and Agraph with the ping command

A quick way of checking the health of a Dgraph or an Agraph is by accessing the URL as described in this topic.

To check the aliveness of a Dgraph or Agraph:

For a Dgraph, access:

```
http://DgraphServerNameOrIP:DgraphPort/admin?op=ping
```

or for an Agraph, access:

```
http://AgraphServerNameOrIP:AgraphPort/admin?op=ping
```

The Dgraph or Agraph quickly returns a lightweight HTML response page with the following content:

```
dgraph host:port responding at date/time
```

or

```
agraph 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.

Related Links

[The logs created by the Dgraph](#) on page 68

The Dgraph creates up to five logs, although some of these logs depend on your implementation and the Endeca components that you may be using. This topic provides a summary of these logs.

The logs created by the Dgraph

The Dgraph creates up to five logs, although some of these logs depend on your implementation and the Endeca 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 `--update-log` 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.update-log`. For example, `Dgraph1.update-log` 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 `$component.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*.

The Agraph request log

You can examine the Agraph request log to track performance of a specific Agraph query.

The MDEX Engine generates unique IDs for each Agraph request, communicates these IDs to its child Dgraphs, and maintains these IDs if it needs to requery the child Dgraph for the same request. This means that the IDs generated by an Agraph are listed in all the corresponding Dgraph log entries related to the same request.

For example, assume that the Agraph log contains an entry with request ID A1. The child Dgraphs log entries for this request repeat this ID from the Agraph. This allows you to trace queries in the Dgraph logs back to queries in the Agraph log.

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 `[AppDir]/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 inaccessible 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 Endeca implementation issue the following error and warning messages:

- The .NET API throws the following exception:

```
Endeca.Navigation.ENException:
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.HttpENException.Query(ENQuery req)
```

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.

Related Links

[Managing Dgraph crash dump files on Windows](#) on page 73

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.)

[Managing Dgraph core dump files on Linux and Solaris](#) on page 73

Endeca 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).

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

Endeca 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 *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 Endeca Support to troubleshoot the crash, you will need to reproduce the crash while capturing the full core file.



Chapter 7

Backing up Endeca applications

This section provides a list of the directories and files in your Endeca application that you will want to back up. The application directory structure is based on an environment that was initially established using the Deployment Template. If your implementation varies from the structure described, then the lists of directories and files will give you a starting point for building your backup and recovery strategy.

Required files for backup

For completeness, you can back up an application by making a copy of the entire deployment template application folder. If you want to be more selective, back up the directories in the following table.

The directory structure below is based on an environment managed by the Deployment Template. The variables in paths have the following meanings:

- [appdir] indicates the path into which your application was deployed when you ran the deploy script. For example, if your application name is MyApp, specifying the deployment directory as C:\Endeca\apps installs the template for your application into C:\Endeca\apps\MyApp.
- ENDECA_CONF indicates the path of the workspace directory for the Endeca HTTP service. In default installation, this value is C:\Endeca\PlatformServices\workspace (on Windows) or endeca/PlatformServices/workspace (on UNIX).
- ENDECA_TOOLS_ROOT indicates the path of the workspace directory for the Endeca Tools Service. In default installation, this value is C:\Endeca\Workbench\workspace (on Windows) or endeca/Workbench/workspace (on UNIX).

Product	Directory	Contents	Comments
Endeca application	[appdir]\config	Configuration and pipeline files	
	[appdir]\control	Scripts	
	[appdir]\logs	Log files	The system requires that you copy this directory the first time you back up. Subsequent backups of this directory is optional.
	[appdir]\data\dgidx_output	Initial index	

	[appdir]\data\partials\cumulative_partials	Partial update files	You must apply these files to the initial index.
	[appdir]\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.
	[appdir]\test_data	Test data files	These files are optional.
Platform Services	ENDECA_CONF\conf		Pagebuilder templates are also included in this directory.
	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_ROOT\conf	Includes Workbench extensions, and Workbench user definitions.	
	ENDECA_TOOLS_ROOT\state\emanager		



Note: For more information about backing up indices, see the *MDEX Engine Partial Updates Guide* and the *MDEX Engine Migration Guide*.

Backing up CAS configurations

Backing up CAS configurations requires a different approach than you would perform for copying and archiving files.

To extract your CAS configurations, you must run specific commands that are described in the *Administering Content Acquisition* section of the *CAS Developer's Guide*. 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

Backing up the Discovery Framework

The directories and files you should back up for the Discovery Framework are listed in the following table.



Note: Always stop your Discovery Framework server before backing up the Discovery Framework files.

Directory or files	Contents	Comments
endeca-portal/data		
endeca-portal/Endeca-data-sources	Data sources	
endeca-portal/portal-ext.properties	Portal properties	
endeca-portal/ee/license	License for the Discovery Framework	
endeca-portal/tomcat/version/webapps	Deployed portlets	These files should be backed up if environment-specific configurations have been applied to deployed portlets.
endeca-portal/tomcat/version/webapps/corda/WEB-INF/classes/Corda60/chart_root/appfiles	Appearance files	

Backing up custom or extended portlets

If you customized or extended any portlets, you should back up the source code and the WAR files for the portlets themselves and also any JAR files that the portlets reference.

Backing up the Discovery Framework database

If you are using MYSQL, DB2 or another relational database management system, you should back up the Discovery Framework database using the backup procedures and tools provided by the vendor.

If your development environment uses the Hypersonic SQL database that is packaged with Liferay, you can back up the database by copying up the `hsql` directory, in which it resides.



Note: For information on backing up the Discovery Framework, see the *Discovery Framework Installation Guide*.

What not to backup

The following table lists files that you do not need to back up for recovery.

File Type	Files	Comments
Archives	Time-stamped archive files such as <code>forge_output</code> , <code>dgidx_output</code> , and logs are optional.	
Processing files	<code>[appdir]/data/complete_data_config</code> <code>[appdir]/data/forge_output</code> <code>[appdir]/data/incoming</code> <code>[appdir]/data/partials</code> <code>[appdir]/data/processing</code> <code>[appdir]/data/temp</code> <code>[appdir]/data/web_studio</code>	You should backup <code>data/partials/cumulative_partials</code> .
Dgraph instances	<code>[appdir]/data/dgraphs</code>	

When primary and recovery environments are different

In some scenarios, the recovery environment may differ from the primary production environment, with different number of servers, operating systems and network architectures.

By using the Deployment Template and following certain procedures, you can ensure that Endeca applications running in one environment can be smoothly replicated in other environments with different characteristics.

Techniques for replicating Endeca applications across heterogeneous environments are described in Chapter 3 of this guide.

Backing up customized files

Endeca recommends that administrators store files created for customized applications, such as command-line scripts and library files, under `[appdir]`. However, if you choose not to follow this practice, then you must keep track of where these files are stored, and include their file locations in backup processes.

One option is to create a script that collects these files and stores copies under `[appdir]`. Running this script immediately before a backup ensures that copies of the customized files will be backed up together with your application files.

A sample script, `collect-app.bat` is available in Chapter 3 of this guide. You can write a similar script to fit your Endeca environment.



Appendix A

Administrative and configuration operations and logging variables

The MDEX Engine supports many administrative and configuration operations that you can access through simple URLs. You can use these operations and their logging variables to control the behavior of the MDEX Engine cleanly from within the system.

About 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. It indicates which URLs are supported by the Agraph.

The syntax of administrative and configuration operations

In the following listings, `<host>` refers to the hostname or IP address of the MDEX Engine and `<port>` refers to the port on which the MDEX Engine is listening. 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.

For administrative operations, the syntax is:

```
http://<host>:<port>/admin?op=<supported-operation>
```

For configuration operations, the syntax is:

```
http://<host>:<port>/config?op=<supported-operation>
```



Note: If you are using HTTPS mode, use `https` in the URL.

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
<code>/admin?op=help</code>	Returns the usage page for all of the admin operations.
<code>/admin?op=ping</code>	Checks the aliveness of an MDEX Engine and returns a lightweight message.
<code>/admin?op=flush</code>	Specifies when the MDEX Engine should flush its dynamic cache.
<code>/admin?op=exit</code>	Stops a running MDEX Engine.
<code>/admin?op=restart</code>	Restarts the MDEX Engine.
<code>/admin?op=audit</code>	Returns the MDEX Engine Auditing page.
<code>/admin?op=auditreset</code>	Resets the MDEX Engine Auditing page.
<code>/admin?op=stats</code>	Returns the MDEX Engine Statistics page.
<code>/admin?op=statsreset</code>	Resets the MDEX Engine Statistics page.
<code>/admin?op=logroll</code>	Forces a query log roll, with the side effect of remapping stdout.
<code>/admin?op=update</code>	Applies any partial update files to the MDEX Engine.
<code>/admin?op=updateaspell</code>	Rebuilds the aspell dictionary for spelling correction from the data corpus without stopping and restarting the MDEX Engine.
<code>/admin?op=updatehistory</code>	Shows a list of the update files that the MDEX Engine has processed recently.
<code>/admin?op=reload-services</code>	A Web services operation that reloads the application's main and library modules.

Related Links

[help](#) on page 81

`/admin?op=help` returns the usage page for all of the administrative operations.

[ping](#) on page 81

`/admin?op=ping` checks the aliveness of an MDEX Engine and returns a lightweight message.

[flush](#) on page 81

`/admin?op=flush` flushes the Dgraph cache.

[exit](#) on page 82

`/admin?op=exit` stops a running MDEX Engine.

[restart](#) on page 82

`/admin?op=restart` restarts the Dgraph.

[audit](#) on page 82

`/admin?op=audit` returns the MDEX Engine Auditing page.

[auditreset](#) on page 83

`/admin?op=auditreset` resets the MDEX Engine Auditing page.

[stats](#) on page 83

`/admin?op=stats` returns the MDEX Engine Statistics page.

[statsreset](#) on page 84

`/admin?op=statsreset` resets the MDEX Engine Statistics page.

[logroll](#) on page 84

`/admin?op=logroll` forces a query log roll, with the side effect of remapping `stdout`.

[update](#) on page 84

`/admin?op=update` applies any partial update files to the Dgraph.

[updateaspell](#) on page 85

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.

[updatehistory](#) on page 86

`/admin?op=updatehistory` shows a list of the update files that the Dgraph has processed since it was started.

[reload-services](#) on page 86

`/admin?op=reload-services` is 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.

Agraph support

The `help` operation is supported in the Agraph.

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 aliveness of a Dgraph or an Agraph 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 health or heartbeat of the MDEX Engine, and as a health check for load balancers.

Agraph support

The `ping` operation is supported in the Agraph.

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...
```

Agraph support

The `flush` operation is not supported in the Agraph, because the Agraph does not contain a 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 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.

Agraph support

The `exit` operation is supported in the Agraph.

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
```



Agraph support

The `restart` operation is supported in the Agraph.

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.

Address  http://localhost:15001/admin?op=audit Links » 

Endeca MDEX Engine (dgraph) Audit Statistics

Product: Endeca Information Access Platform version 6.1.0.298155, 6.1.0.298155

Audit Stats **General Information**

▼ Query Load Expand All
Collapse All

Period	Period Length	Peak Interval	Interval Length	Peak Value
Sun Jan 18 00:00:00 2009	weeks	Sun Jan 18 00:00:00 2009	hours	0
Sun Jan 25 00:00:00 2009	weeks	Sun Jan 25 00:00:00 2009	hours	0



Note: For details about the MDEX Engine Auditing page, see the *Performance Tuning Guide*.

Agraph support

The audit operation is supported in the Agraph.

auditreset

`/admin?op=auditreset` resets the MDEX Engine Auditing page.

It returns the following message:

```
resetting auditing stats...
```

Agraph support

The `auditreset` operation is not supported in the Agraph.

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*.

Agraph support

The `stats` operation is supported in the Agraph, but returns a smaller set of statistics.

statsreset

`/admin?op=statsreset` resets the MDEX Engine Statistics page.

The `statsreset` operation returns the following message:

```
resetting server stats...
```

Agraph support

The `statsreset` operation is supported in the Agraph.

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.
```

Agraph support

The `logroll` operation is supported in the Agraph.

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.

Agraph support

The `/admin?op=update` operation is not supported in the Agraph.

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.
```

Agraph support

The `admin?op=updateaspell` is not supported in the Agraph.

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_in-
put\updates\"

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"
```

Agraph support

The `updatehistory` operation is not supported in the Agraph.

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-ser- vices`. 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.

Agraph support

The `reload-services` operation is not supported in the Agraph.

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
<code>/config?op=help</code>	Returns the usage page for all of the config operations.
<code>/config?op=update</code>	Loads and applies updated MDEX Engine configuration files.

Config operation	Description
<code>/config?op=log-enable</code>	Enables verbose logging for one or more specified variables.
<code>/config?op=log-disable</code>	Disables verbose logging for one or more specified variables.
<code>/config?op=log-status</code>	Returns verbose logging status.

Related Links

[help](#) on page 87

`/config?op=help` returns the usage page for all of the config operations.

[update](#) on page 87

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.

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.



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.
```

Agraph support

The `config?op=update` operation is not supported in the Agraph.

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.

Related Links

[Logging variable operation syntax](#) on page 88

MDEX Engine logging variables are toggled using the `/config?op=log-enable&name=<variable-name>` and `/config?op=log-disable&name=<variable-name>` operations.

[List of supported logging variables](#) on page 88

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 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.

Related Links

[log-enable](#) on page 89

The `log-enable` operation lets you turn on verbose logging.

[log-disable](#) on page 89

The `log-disable` operation lets you turn off verbose logging.

[log-status](#) on page 90

The `log-status` operation returns a list of all logging variables with their values (true or false).

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
```



Appendix B

Endeca Flag Reference

This appendix provides a description of the flags (options) used by the Agidx, Agraph, Dgidx, and Dgraph programs. For information on Forge flags, see the *Forge Guide*.

Agidx flags

Agidx is a program that runs in a distributed environment. It creates a set of Agidx indices and aggregates the Agraph index with the current data subset.

Agidx has the following usage:

```
agidx [-v] [--agidx_out <input db_prefix>]
      [--help]
      [--config]
      [--out <stdout/stderr file>] [--version]
      <input db_prefix list> <output db_prefix>
```

where `<db_prefix>` specifies the path to the directory, and the prefix used for the files in your Endeca application.

Agidx contains the following options:

Flag	Description
<code>--agidx_out <input db_prefix></code>	Prefix for output generated previously by Agidx that should now be used as input. This option helps you incrementally build the Agidx index, allowing you to run Agidx against individual data subsets that have been generated by Dgidx.
<code>--config</code>	Specify a configuration file to read on startup. Configuration file should contain arguments of the same format used on the command line (it ignores whitespace, including new lines). A command line that includes the <code>--config</code> argument is processed as if the contents of the config file were inserted in place of the <code>--config</code> option.
<code>--help</code>	Print this usage information and exit.

Flag	Description
-v	Verbose mode.
--out <stdout/stderr file>	Specify file path to which <code>stdout/stderr</code> should be remapped (default is to use default <code>stdout/stderr</code> for the process).
--version	Print version information and exit.

Agraph flags

A distributed configuration requires a program called Agraph.

The Agraph program is responsible for receiving requests from clients, forwarding the requests to the distributed MDEX Engines, and coordinating the results. From the perspective of the Endeca API, the Agraph program behaves identically to a Dgraph program.

The Agraph has the following usage:

```
agrapph [-v] [--flags] <db_prefix>
```

where <db_prefix> specifies the path to the directory, and the prefix used for the files in your Endeca application.

The Agraph uses the following flags:

Flag	Description
-v	Verbose mode
-A	<i>Deprecated.</i> Disallow server shutdown and restart operations through <code>admin?op=exit</code> and <code>admin?op=restart</code> URL commands sent to the Dgraph.
--back_compat <api-version>	Enable backwards compatibility, so that the Agraph can communicate with previous versions of the Presentation API. The following full versions are supported: 6.0.x, 5.1.x, 5.0.x and 4.8.x. Therefore, the value for <api-version> must be one of the following: <ul style="list-style-type: none"> • 601 for all 6.0.x versions of the API. • 510 or 500 for all corresponding versions of the API, 5.1.x. • 480 for the 4.8.x versions of the API, including the Perl API.
--child <host>:<port>	Specify the location of a child Dgraph or Agraph process.
--config <filename>	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).
--explicit_no_keep_alive	Explicitly set the "connection:close" header on all HTTP responses.
--fork	(UNIX only) Causes the Agraph to fork off a new process to handle each request.
--fork-max <max-fork-children>	(UNIX only) Set the maximum number of live child processes in <code>--fork</code> mode. Default value is 4.
--help	Print usage information and exit.

Flag	Description
<code>--log <path></code>	Change the path for the request log file (<code>./agraph.reqlog</code> is the default value)
<code>--net-close-timeout</code>	Set default maximum wait time (in seconds) for client connection shutdown. The default value is 1 second.
<code>--net-timeout</code>	Specify the maximum number of seconds the Agraph waits for the client to download data across the network. The default network timeout value is 30 seconds.
<code>--nodnscache</code>	Disable caching of hostname to IP number lookups for child Dgraphs. By default, the Agraph caches these name lookups to improve performance.
<code>--noimplicit</code>	<p>Disable inclusion of implicit refinement dimension values in computed refinement sets. Implicit refinements are dimension values that are assigned to all records in the current result set, and whose selection therefore does not narrow the results.</p> <p>This flag sets the following conditions:</p> <ul style="list-style-type: none"> • Implicit dimensions values are not displayed. • Dimensions with only implicit dimension values are not displayed. • Implicit dimension values trigger precedence rules.
<code>--nomerch</code>	Do not process dynamic business rule results from children. These are processed by default.
<code>--no-partial</code>	Do not return results if any child fails to respond.
<code>--out <stdout/stderr file></code>	Specify file path to which stdout/stderr should be remapped. (The default is to use default stdout/stderr for the process.)
<code>--pidfile <pidfilename></code>	Specify the file to which to write the process ID (pid). If unspecified, the default name of the pid file depends on how the Agraph starts. Running the Agraph in a Control System environment (deprecated) or from the command line creates a default named <code>agraph.pid</code> . Running the Agraph in an Endeca Application Controller environment creates a default named <code>agraph-S0-R0.pid</code> .
<code>--port <num></code>	Specify the port that the Agraph listens to for user queries on the associated host. Default is 8888.
<code>--radius <num></code>	Specify initial record list radius (tuning parameter; the default is 100).
<code>--stat-brel</code>	Create dynamic record properties indicating the relevance rank assigned to record search results.
<code>--sslcertfile</code>	Specify the name of the SSL certificate file. If not given, SSL is not enabled for Agraph communications.
<code>--sslcafile</code>	Specify the name of the SSL Certificate Authority file. If not given, CA verification is not performed.
<code>--sslcipher</code>	Only allow the ciphers specified in the colon-separated list of cipher names following this option.

Flag	Description
--version	Print version information and exit.

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 Endeca application.

Dgidx supports the following flags:

Flag	Description
-q	Quiet mode.
-v	Verbose mode.
--autogenerate-dval-specs	Specify the auto-generation of dimension value specs. If this flag is specified, then during both baseline and partial updates, any dimension value that does not have a dimension value spec is assigned one.
--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 <i>MDEX Engine Basic Development Guide</i> .
--equivopt	<i>Deprecated.</i> The MDEX Engine ignores this flag if it is specified. Compute dimension value equivalence classes as a space-saving optimization. This adds time to the indexing phase, but reduces the size of the index. The default is to search leaf assignments only.
--help	Print the help message and exit.
--lang <lang-id>	Assume all documents are in the specified language. The default for <lang-id> is en.
--latin1	<i>Deprecated.</i> Ignore character accents when indexing text. Use ISO Latin 1 character mappings for international characters when performing search indexing. Note that the accents are folded down before indexing, so only a single form is indexed.
--ngram_min <value>	<i>Deprecated.</i> The MDEX Engine ignores this flag if it is specified.

Flag	Description
<code>--noimplicit</code>	<p>Disable computation of implicit refinement dimension values. Implicit refinements are dimension values that are assigned to all records in the current result set, and whose selection therefore does not narrow the results.</p> <p>In addition, this flag disables computation of dimension values for disabled refinements.</p>
<code>--nostrictattrs</code>	<p>Disable strict attribute checking. Allows records to retain property values for properties with no property (or <code>PROP_REF</code> element) defined in the navigation configuration file, and in the Properties view of Developer Studio.</p>
<code>--noxmlvalidate</code>	<p>Do not perform XML validation while reading the XML export file. This option only makes a difference if the export file is in XML format.</p>
<code>--numbins <num></code>	<p>Limit the number of records that Dgidx reads.</p>
<code>--out <stdout/stderr file></code>	<p>Specify file path to which stdout/stderr should be remapped (the default is to use default stdout/stderr for the process).</p>
<code>--sort <spec></code>	<p>Specify a default sort specification for the data set. The format of <code><spec></code> is (including the quotation marks):</p> <pre>"key dir"</pre> <p>where <i>key</i> is the name of a property or dimension on which to sort and <i>dir</i> is either <code>asc</code> for ascending or <code>desc</code> for descending (if not specified, the order will be ascending).</p> <p><i>key</i> can also be a geocode property, as in this example:</p> <pre>"Location(43,73) desc"</pre> <p>You can specify multiple sort keys in the format:</p> <pre>"key_1[dir_1] key_2[dir_2] ... key_n[dir_n]"</pre> <p>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.</p> <p>Note that if you are using the Endeca Application Controller (EAC) to control your environment, you must omit the quotation marks from the <code>--sort</code> flag. Instead, use the following syntax:</p> <pre>--sort key_1 dir_1 key_2 dir_2 ... key_n dir_n</pre>
<code>--spellmode <mode></code>	<p>Specify the spelling correction mode for the application. Supported modes are:</p> <ul style="list-style-type: none"> • default • aspell • espell • aspell_OR_espell • aspell_AND_espell

Flag	Description
<code>--spellnum</code>	In spelling modes that enable the <code>espell</code> module, include non-word terms (numbers, symbols, and so on) in the <code>espell</code> dictionary. By default, such terms are not included.
<code>--stemming-updates</code> <code><file></code>	Specify an optional XML file of stemming updates to apply to a default stemming dictionary. See the <i>MDEX Engine Advanced Development Guide</i> for XML examples and file name requirements.
<code>--threads <num></code>	<p>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.</p> <p>Note that Dgidx runs in multithreaded mode by default. In addition to the number of sorting threads that you can control with the <code>--threads</code> flag, Dgidx may use additional maintenance threads that run in the background by default, and are not used for sorting data.</p> <p>To improve indexing performance, Endeca 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.</p> <p>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 <i>N</i> sorting threads does not speed up Dgidx by <i>N</i> times.</p>
<code>--version</code>	Print version information and exit.
<code>--verbose-language-mapping</code>	<i>Deprecated.</i> The MDEX Engine ignores this option if it is specified. Report which record properties are mapped to which languages.

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 [-?Adv] [--flags] <db_prefix>
```

where `<db_prefix>` specifies the path to the directory, and the prefix used for the files in your Endeca application.

Flag	Description
<code>?</code>	Print the help message and exit.


Flag	Description
-A	<i>Deprecated.</i> Disallow server shutdown and restart operations through <code>admin?op=exit</code> and <code>admin?op=restart</code> URL commands sent to the Dgraph.
-d	Start in debug mode.
-v	Verbose mode. Print information about each request to <code>stdout</code> .
--ancestor_counts	Compute counts for root dimension values and any intermediate dimension value selections. By default, the Dgraph only computes refinement counts for proper refinements (in other words, for actual refinement dimension values). It does not compute counts for root dimension values or for any intermediate dimension value selections.
--back_compat <api-version>	<p>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 full versions are supported: 6.0.x, 5.1.x, 5.0.x and 4.8.x. Therefore, the value for <api-version> must be one of the following:</p> <ul style="list-style-type: none"> • 601 for all 6.0.x versions of the API. • 510 or 500 for all corresponding versions of the API, 5.1.x and 5.0.x. • 480 for the 4.8.x versions of the API, including the Perl API. <p> Note: Starting with version 6, the Endeca Presentation API is part of the Platform Services package. For the version of the Platform Services that is compatible with the current version of the MDEX Engine, see the <i>MDEX Engine Installation Guide</i>.</p>
--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 <code>--cmem</code> 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).
--deadends	<i>Deprecated.</i> The MDEX Engine ignores this flag if it is specified.
--diacritic-folding	Ignore character accents when processing search requests. For details about how characters with diacritical marks are mapped to their ASCII equivalents, see the <i>MDEX Engine Basic Development Guide</i> .


Flag	Description
<code>--disable_fast_aspell</code>	<p>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.</p> <p>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.</p>
<code>--disable_web_services</code>	Suppress the automatic loading of XQuery modules at startup.
<code>--dtag <data-tag></code>	Specify the data tag to send with all result XML objects. The default is to use <code>db_prefix</code> as the data tag.
<code>--dym</code>	Enable DYM (Did You Mean?) explicit query spelling suggestions for full-text search queries.
<code>--dym_hthresh <thresh></code>	Specify the threshold number of hits at or above which DYM (Did You Mean?) suggestions will not be generated. The default is 20.
<code>--dym_nsug <count></code>	Specify the maximum number of DYM (Did You Mean?) query suggestions to return for any query. The default is 1.
<code>--dym_sthresh <thresh></code>	Specify the threshold spelling correction score for words used by the DYM (Did You Mean?) engine. The default is 175.
<code>--dynrank_consider_collapsed</code>	<p>Use this flag to force the MDEX Engine to consider intermediate collapsible dimension values as candidates for dynamic ranking.</p> <p>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.</p> <p>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.</p> <p>If you would like the MDEX Engine to consider intermediate dimension values (that are configured as collapsible) for dynamic ranking, use this flag.</p>
<code>--esampmin <num></code>	<p>Specify the minimum number of records to sample during refinement computation. The default is 0. Tuning recommendations:</p> <ul style="list-style-type: none"> • 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.
<code>--ethresh <num></code>	<i>Deprecated.</i>

Flag	Description
<code>--explicit_no_keep_alive</code>	<i>Deprecated.</i> If specified, triggers a deprecation warning but is otherwise ignored.
<code>--failedupdatedir <dir></code>	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 <code><updatedir>/failed_updates/</code> .
<code>--help</code>	Print the help message and exit.
<code>--implicit_exact</code>	<i>Deprecated.</i> The MDEX Engine ignores this flag if it is specified.
<code>--implicit_sample</code>	<i>Deprecated.</i> The MDEX Engine ignores this flag if it is specified.
<code>--lang <lang-id></code>	Assume all queries are in the specified language. The default is <code>en</code> (US English).
<code>--latin1</code>	<i>Deprecated.</i> Ignore character accents when handling search requests, and use ISO Latin 1 character mappings when processing search requests.
<code>--log <path></code>	Specify the path for the Dgraph request log file. The default log file is named <code>dgraph.reqlog</code> .
<code>--log_stats <path></code>	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.
<code>--log_stats_thresh <value></code>	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 <code>s</code> to the number, such as <code>1s</code> for 1 second. The default is 60000 milliseconds (1 minute). Note that this flag is dependent on the <code>--log_stats</code> flag being used.
<code>--memusage</code>	<i>Deprecated.</i> The MDEX Engine ignores this flag if it is specified.
<code>--mergepolicy <policy></code>	Set the default merge policy of the MDEX Engine for partial updates. The value for <code><policy></code> must be either <code>balanced</code> or <code>aggressive</code> . If this flag is not used, <code>balanced</code> will be the default merge policy. For details on the merge policy, see the <i>Partial Updates Guide</i> .
<code>--net-close-timeout</code>	<i>Deprecated.</i> Prior to version 6.1, this flag set the default maximum wait time (in seconds) for client connection shutdown. The MDEX Engine now uses the <code>FIN_WAIT_2</code> timeout interval to set the number of seconds that the HTTP server waits after sending the response for the client to close down its end of the socket. If this timeout expires, the server forcibly shuts down the connection. The default value varies by operating system: for Linux it is 60s; for Solaris, it is 675000ms; and for Windows it is 240s. For details on changing the default value in your operating system, see the <i>Performance Tuning Guide</i> .

Flag	Description
<code>--net-timeout</code>	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.
<code>--noctrct</code>	Do not return information about implicit dimensions with node results, when displaying refinements in navigation results. This flag lets you optimize performance for applications where it is not necessary to present the implicit dimensions to the users in navigation results. If you specify this flag, the MDEX Engine still computes the implicit dimensions with node results, but they are not included in the navigation results that are displayed to the users.
<code>--nomrf</code>	Disable filtering for dynamic business rules.
<code>--out <stdout/stderr file></code>	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 Endeca Application Controller environment creates a default file named <code>dgraph-S0-R0.out</code> .
<code>--pcmem</code>	<i>Deprecated.</i> The MDEX Engine ignores this flag if it is specified.
<code>--persistdir</code>	Direct the Dgraph audit persistence file to a directory of your choice. By default, the file is written to a directory called <code>persist</code> that is located in the application's working directory. For details about the audit persistence file, see the <i>Endeca Performance Tuning Guide</i> . Important: Use the <code>--persistdir</code> flag only when you first start the Dgraph. Do not move or rename this directory after it has been created.
<code>--phrase_max <num></code>	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.
<code>--pidfile <pidfile-path></code>	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 a Control System environment or from the command line creates a default named <code>dgraph.pid</code> . Running the Dgraph in an Endeca Manager environment creates a default named <code>dgraph-S0-R0.pid</code> .
<code>--port <num></code>	Specify the port to use in server (non-interactive) mode. The default is 5555.
<code>--search_max <num></code>	Specify the maximum number of terms for text search. Default is 10.
<code>--snip_cutoff <num></code>	Limit the number of words in a property that the MDEX Engine evaluates to identify the snippet. If a match is not found within <code><num></code> words, the MDEX Engine does not return a snippet, even if a match occurs later in the property value.

Flag	Description
	If the flag is not specified, or <code><num></code> is not specified, the default is 500.
<code>--snip_disable</code>	Globally disable snippeting.
<code>--spellpath <path></code>	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 <code>--dym</code> and <code>--spl</code> 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 <code>.spell.dat</code> file for the indexed data set. The Dgraph typically requires write permissions in this directory, unless a correct or writable <code>.pwli</code> file is already available in this directory.
<code>--spell_bdgt <num></code>	Set maximum number of variants considered for spelling and DYM (Did You Mean?) correction (the default is 32).
<code>--spell_glom</code>	Allow cross-property suggestions, and count cross-property matches when evaluating the frequencies of suggestions. Normally, suggestions must match results in a single property value.
<code>--spell_nobrk</code>	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.
<code>--spl</code>	Enable auto-suggest spelling corrections for record (full text) and dimension search.
<code>--spl_hthresh <thresh></code>	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.
<code>--spl_nsug <count></code>	Specify the maximum number of auto-correct suggestions to return. The default is 1.
<code>--spl_sthresh <thresh></code>	Specify the threshold spelling correction score for words used as auto-correct suggestions. The default is 125.
<code>--sslcertfile <certfile-path></code>	Specify the path of the <code>eneCert.pem</code> 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.
<code>--sslcafile <CA-certfile-path></code>	Specify the path of the <code>eneCA.pem</code> 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

Flag	Description
	<p>authentication purposes. If not given, SSL mutual authentication is not performed.</p> <p> Note: If you need to establish a secure but not authenticated connection, use the <code>--sslcertfile</code> flag without the <code>--sslcafile</code> flag.</p>
<code>--sslcipher <cipher-list></code>	Set one or more cipher names (such as RC4-SHA) that specify the minimum cryptographic algorithm that the Dgraph will use during the SSL negotiation. If multiple ciphers are specified, the names must be separated by colons.
<code>--stat-all</code>	Enable all available dynamic dimension value attributes. Note that this option has performance implications and is not intended for production use.
<code>--stat-abins</code>	<p>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.</p> <p>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.</p> <p>In general, the MDEX Engine calculates refinement counts as follows:</p> <ul style="list-style-type: none"> • 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 regular records are returned by the MDEX Engine as the <code>Dgraph.Bins</code> property. • When returning aggregated record results, the <code>--stat-abins</code> 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 <code>Dgraph.AggrBins</code> property. <p>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.</p>
<code>--stat-bins-cutoff <num></code>	<i>Deprecated.</i> Set the cutoff for record counts. Once there are this many records associated with a refinement dimension value, the record count algorithm stops and returns this number or a number higher than it.
<code>--stat-bins-thresh <thresh></code>	<i>Deprecated.</i> Set the threshold for the maximum number of records above which the MDEX Engine stops computing record counts. By

Flag	Description
	default, the MDEX Engine returns refinement counts for records with no threshold.
--stat-brel	Create dynamic record attributes indicating the relevance rank assigned to fulltext search result records.
--stat-rel	Create dynamic dimension value attributes indicating the relevance ranking score (for dimension value search results).
--syslog	Direct all output to syslog.
--thesaurus_cutoff <limit>	<p>Set a limit on the number of words in a user's search query that are subject to thesaurus replacement.</p> <p>The default value of <i><limit></i> 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.</p> <p>This option is intended as a performance guard against very expensive thesaurus queries. Lower values improve thesaurus engine performance.</p>
--thesaurus_multiword_nostem	<p>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.</p> <p>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.</p>
--threads <num>	<p>Specify the number of threads in the MDEX Engine threading pool. The default is 1 (multithreaded mode). The multithreaded mode cannot be disabled.</p> <p>The recommended number of threads for the MDEX Engine is typically equal to the number of cores on the MDEX Engine server.</p> <p>If you specify a value greater than 0, the Dgraph runs the specified number of threads for processing client requests (queries and partial updates), and other CPU-intensive operations related to query processing. The MDEX Engine prioritizes tasks assigned to threads in its threading pool.</p> <p> Note: If the specified value is 0, the Dgraph interprets it as 1 and still runs in multithreaded mode with the number of threads in its threading pool set to 1.</p> <p>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).</p>

Flag	Description
<code>--tmpdir <dir></code>	Specify the path to a temporary directory to be used to hold temporary files (the default is the base directory of <code>db_prefix</code>).
<code>--unctrct</code>	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.
<code>--updatedir <dir></code>	Specify the directory into which completed partial update files will be placed. Partial update files are also read from this directory.
<code>--updatelog</code>	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 <code>dgraph.updatelog</code> . Running the Dgraph in an Endeca Application Manager environment creates a default named <code>dgraph-S0-R0-update.log</code> .
<code>--updateverbose</code>	Show verbose messages while processing updates.
<code>--validate_data</code>	Validate that all indexed data loads and then exit.
<code>--version</code>	Print version information and exit.
<code>--wb_maxbrks</code>	In word-break analysis, specify the maximum number of breaks to insert or remove per query. The default is 1.
<code>--wb_minbrklen</code>	In word-break analysis, specify the minimum length of a new word-break term. The default is 2.
<code>--wb_noibrk</code>	In word-break analysis, disable word-break insertion analysis.
<code>--wb_norbrk</code>	In word-break analysis, disable word-break removal analysis.
<code>--wildcard_approx <mode></code>	<i>Deprecated.</i> The MDEX Engine ignores this flag.
<code>--wildcard_max <count></code>	Specify the maximum number of terms that can match a wildcard term in a wildcard query that contains punctuation, such as <code>ab*c.def*</code> . The default is 100.
<code>--whymatch</code>	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).
<code>--whymatchConcise</code>	Similar to <code>--whymatch</code> , 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.

Flag	Description
--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.
--ws	<i>Deprecated.</i> The MDEX Engine ignores this flag and issues a warning if it is specified.
--xquery_fndoc <mode>	<p>Specifies the handling of the <code>fn:doc()</code> function within XQuery. The following three values are supported:</p> <ul style="list-style-type: none"> • <code>none</code> causes all calls to <code>fn:doc()</code> to fail. • <code>sandbox</code> allows <code>fn:doc()</code>, but interprets its argument as a relative path within the XML subdirectory of the XQuery service directory. • <code>open</code> allows <code>fn:doc()</code> and interprets its argument as a URL. <p>If not specified, defaults to <code>none</code>. Note that <code>open</code> is not supported for use in deployed applications.</p>
--xquery_path <path>	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 <code>lib</code> subdirectory. If not specified, a user XQuery path is not used.



Appendix C

XML Configuration Files

This section describes the XML configuration files used by the Endeca 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

The XML configuration files contain settings used to control aspects of the behavior of the Forge, Dgidx, Agidx, Dgraph, and Agraph processes.

The XML configuration files, which are documented in detail in the *Endeca XML Reference*, define dimensions, dimension search, dimension search index, precedence rules, properties, refinements, and many other aspects of your incoming records. Dgidx reads and consumes some of the XML configuration files, while others are passed further into the Dgidx output directory and from there to the Dgraph input directory. In particular, the Dgraph rereads configuration files which are present in XML format in the Dgraph input directory during configuration updates. These files include business rule, keyword redirect, and landing page definitions, thesaurus entries, search interface definitions, derived property definitions, rendering settings, and other files.

The XML configuration files are copied by Forge to its output directory for use by the Dgidx indexer process. If the Forge output and Dgidx input directories are not the same, the Deployment Template scripts copy the configuration files from the Forge output directory to the Dgidx input directory.

The `Project.xml` file informs the Dgidx where to find the Forge-generated files (records, dimensions, and property/dimension configuration), as well as the configuration files copied over by Forge. You do not typically need to edit this file.



Note: In some cases, you need to correct a problem in the XML configuration files. If you are using Endeca Workbench, to learn how to obtain configuration files out of Workbench (or push existing files into Workbench), see the appendix in this guide.

Creating the XML configuration files

The XML configuration files are created as part of your application in one of two ways:

- When you deploy a new application using the Deployment Template, a set of default XML configuration files are created for you in the application's `/config/pipeline` directory. These

files are all initially created by the Deployment Template with a prefix containing the application name, such as `MyApp.Dimensions.xml`. Members of your team can edit these files in Developer Studio or Endeca Workbench to change the default settings.

- When application developers create an Endeca project using Developer Studio, Developer Studio creates the XML configuration files. As your application developers modify the project, Developer Studio and Workbench write changes to the XML files.

The `/config/pipeline` directory also contains the Developer Studio pipeline file, `pipeline.epx`.

You can merge the configuration files from Workbench with the `MyApp/config/pipeline` file set during a baseline update, depending on your ConfigManager settings. See the *Deployment Template Usage Guide* for more details.

Changing the Deployment Template output prefix

If you are using the Deployment Template, be aware that if you change the `output-prefix` value in the Indexer Adapter pipeline component in Developer Studio and save this setting, a new set of XML files is saved with that modified prefix.

The Deployment Template project, however, retains the default prefix for the names of its components, and can copy older files into the output directory, when its scripts run Forge, for instance. This may overwrite the most recent configuration files. Therefore, Endeca recommends that you do not change the default prefix for the XML configuration files.

Creating and modifying the XML configuration files

There are several ways to create and update the XML configuration files.

Endeca recommends that you use Developer Studio or Endeca Workbench to configure your project, and only modify the XML files directly when required in unusual situations. If you choose to modify the XML configuration files directly, avoid writing Byte Order Marks (BOMs) into them.

For more information about BOMs, see [this site](#).



Appendix D

Transferring Endeca Implementations Between Environments

Read this section only if you use Endeca Workbench (and not the Deployment Template) for your project. This section describes how to transfer your Endeca implementation from a staging environment that uses Endeca Workbench to a production environment that uses Endeca Workbench.

For implementations using the Deployment Template

While this section is limited to implementations that only use the Endeca Workbench, the preferred approach to establishing and maintaining an implementation is by using the Deployment Template.

For information about creating and replicating your implementation using the Deployment Template, see the sections, *Creating Multiple Server Environments* and *Replicating application definitions* in this guide.

About transferring your implementation

This section focuses on transferring your instance configuration and MDEX Engine between environments.

Two methods are described: one uses the Endeca tools to manually transfer the implementation, and the other uses the `emgr_update` utility that allows you to script and automate transfers. To improve the readability of the section, we assume you are transferring your Endeca implementation from a staging environment to a production environment. This need not be the case, however. You can use these procedures to transfer between any environments you choose.

Depending on your environment and requirements, you may also have to transfer your front-end Web application to complete the move from one environment to another. From an Endeca perspective, all you have to do to transfer the front-end Web application is make sure the MDEX Engine hostname and port you are using in your `ENEConnection` object is correct for the new environment.




Note: See the *Endeca Basic Development Guide* for details on using the `ENEConnection` interface.

Retrieving the Endeca Workbench instance configuration with Developer Studio

You can use the Endeca tools to manually transfer from a staging environment that uses Endeca Workbench to a production environment that uses Endeca Workbench.

To retrieve the Endeca Workbench instance configuration with Developer Studio:

1. In your staging environment, start Developer Studio and create a new project.
2. From the **Tools** menu, choose **Workbench Settings**. The Workbench Settings dialog box appears.
3. Specify the hostname and port for Endeca Workbench for this application. Make sure the hostname and port that are specified correspond with Endeca Workbench whose information you want to retrieve.
4. In the same dialog box, select the application from the drop-down list, or make sure that the application name that is specified corresponds with the name of the application whose configuration you want to retrieve from Endeca Workbench. Note that you can have instance configurations for more than one application created in Endeca Workbench.
5. In the Endeca Workbench toolbar, click **Get Instance Configuration:** 
6. From the **File** menu, choose **Save** to save the project with the latest instance configuration.
7. Optionally, remove inactive dynamic business rules from the instance configuration.
8. Copy the instance configuration files from the saved project to the location in your production environment where the Endeca Application Controller expects them to be.
9. If you have used the load function in Developer Studio to load auto-generated or external dimensions, you should manually synchronize these Forge state files.
10. Use the Application Controller to run a baseline update on the production system.



Note: If you have loaded any auto-generated or external dimension values into Developer Studio using the load function, the ID assignments are not stored in the instance configuration. You must manually synchronize these files between implementation environments.

Related Links

[Transferring auto-generated and external dimension value ID assignments](#) on page 118

If you have loaded any auto-generated or external dimension values into Developer Studio using the load function, the ID assignments are not stored in the instance configuration. Therefore, you should manually synchronize these files between implementation environments.

About emgr_update

The emgr_update utility assists you in updating the instance configuration of a production system based on the changes made with the Endeca tools in a staging environment.

By using the appropriate --action operations, you can use emgr_update to do the following tasks:

- Transfer the instance configuration files for a particular application of your choice from the staging environment to the production environment. After the transfer, you run a baseline update using your own EAC scripts. You have the option of transferring all instance configuration files, or transferring just the instance configuration files that Endeca Workbench modified.

- Transfer the instance configuration for a particular application from one Endeca Workbench environment to another.
- Remove instance configuration information for a specified application from the Endeca Workbench configuration.
- Send the Forge dimensions to the Endeca Workbench.

emgr_update syntax reference

This section lists all command line parameters that you can use with `emgr_update`.

The `emgr_update` utility assists you in updating the instance configuration of a production system based on the changes made with the Endeca tools in a staging environment.

You run `emgr_update` from a command line. Open a command prompt or UNIX shell to run the program. The syntax for running `emgr_update` is: `emgr_update <parameters>`

The following table describes the command line parameters you can use with `emgr_update`. You can specify only one `--action` operation for each invocation of the utility.

Here is an example of usage:

```
emgr_update --host localhost:8006 --action get_ws_settings
--prefix wine --dir /apps/endeca/data/forge_input --app_name wine
```

emgr_update parameter	Description
<code>--host name:port</code>	Specifies the host name of a machine running Endeca Workbench and the Endeca Tools Service port on that machine. If you are retrieving settings (using the <code>get</code> operation), this is the host name of the environment you are transferring from; if you are updating settings (using the <code>set</code> operation), this is the host name of the environment you are transferring to.
<code>--action <op></code>	Specifies one of the actions, where <code><op></code> is one of the operations listed below.
<code>--action get_all_settings</code>	Retrieves all the instance configuration settings for a project you performed in the Endeca Workbench in the staging environment, for their use in the production environment. Required parameters: <code>--dir</code> , <code>--prefix</code> Optional parameters: <code>--filter</code>
<code>--action get_ws_settings</code>	Retrieves only those instance configuration settings that can be modified in Endeca Workbench (not all settings). These configuration settings include the following Endeca Workbench features: dynamic business rules, keyword redirects, thesaurus entries, automatic phrases, stop words, and dimension ordering.

emgr_update parameter	Description
<code>--action get_mdex_settings</code>	Retrieves the instance configuration settings that were modified in Endeca Workbench, and that do not require a baseline update to update the MDEX Engine. These configuration settings include the following Endeca Workbench features: dynamic business rules, keyword redirects, thesaurus entries, automatic phrases. Required parameters: <code>--dir</code> , <code>--prefix</code> Optional parameters: <code>--filter</code>
<code>--action set_post_forge_dims</code>	Updates the Endeca Workbench configuration with the post-Forge dimensions.
<code>--action get_post_forge_dims</code>	Retrieves the copy of Endeca Workbench settings for the post-Forge dimensions. Typically, this operation can be used for debugging purposes.
<code>--action update_mgr_settings</code>	Updates an Endeca Workbench production environment with instance configuration settings that were extracted from the Endeca Workbench configuration in the staging environment.
<code>--action remove_all_settings</code>	Removes all the instance configuration files from Endeca Workbench for the application that you specify with the <code>--app_name</code> parameter. Removing the instance configuration does not remove the associated provisioning information for an application.
<code>--app_name <string></code>	Specifies the name of the application provisioned to the EAC Central Server.

Additional action parameters	Description
<code>--dir <pathname></code>	Specifies the <code>pathname</code> of the directory where the instance configuration files are written to or read from. Required for all <code>--action</code> operations except for <code>set_post_forge_dims</code> and <code>remove_all_settings</code> .
<code>--prefix <pathname></code>	Specifies the prefix used for the instance configuration files. This option is required for all <code>--action</code> operations except for <code>get_post_forge_dims</code> and <code>set_post_forge_dims</code> .
<code>--filter</code>	Filters out dynamic business rules that have a state of inactive. (A rule has the property <code>endeca.internal.workflow.state</code> set to <code>INACTIVE</code> .) This option can be used in conjunction with

Additional action parameters	Description
	<p><code>get_all_settings</code> or <code>get_ws_settings</code> when retrieving an instance configuration.</p> <p>Removing inactive rules is not required but it is recommended. With the default rule filter in place, the MDEX Engine does not fire any rule whose state is inactive. In other words, you can transfer an instance configuration, including both active and inactive rules, and the MDEX Engine fires only active rules in reply to user queries.</p>
<code>--post_forge_file <pathname></code>	Specifies the pathname to the file that contains post-Forge dimensions. This option is required for the <code>set_post_forge_dims</code> operation.

Optional global parameters	Description
<code>--stop_on_warnings</code>	Stops the utility without asking you if the target directory is not empty before a get operation, or if it finds extra or missing files before an update operation.
<code>--ignore_warnings</code>	Continues running the utility if the target directory is not empty before a get operation, or continues if there are extra or missing files before an update operation.
<code>--help</code>	Displays the usage parameters for the utility.
<code>--version</code>	Displays the version number for the utility.

Related Links

[About transferring implementations using the emgr_update utility](#) on page 113

Similar to the Endeca tools, the `emgr_update` utility lets you transfer your Endeca implementation from a staging environment that uses Endeca Workbench to the production environment that uses Endeca Workbench.

About transferring implementations using the emgr_update utility

Similar to the Endeca tools, the `emgr_update` utility lets you transfer your Endeca implementation from a staging environment that uses Endeca Workbench to the production environment that uses Endeca Workbench.

The primary benefit of the `emgr_update` utility is that it allows you to script and automate transfers between environments. Transferring an implementation from staging to production is a two-step process where you get the instance configuration from the staging environment and then set (update) the configuration in the production environment.

This section describes how to transfer instance configuration files from a staging environment that uses Endeca Workbench to a production environment that also uses Endeca Workbench. Two scenarios are described:

- Transferring all instance configuration files for an Endeca project.

- Transferring only the instance configuration files that can be modified by Endeca Workbench.



Note: If you have loaded any auto-generated or external dimension values into Developer Studio using the load function, the ID assignments are not stored in the instance configuration. You must manually synchronize these files between implementation environments.

Related Links

[emgr_update syntax reference](#) on page 111

This section lists all command line parameters that you can use with emgr_update.

[Transferring auto-generated and external dimension value ID assignments](#) on page 118

If you have loaded any auto-generated or external dimension values into Developer Studio using the load function, the ID assignments are not stored in the instance configuration.

Therefore, you should manually synchronize these files between implementation environments.

Transferring all instance configuration files

To transfer all instance configuration files, you move the files from the staging environment to the Forge input directory in the production environment.

You then use the Endeca Application Controller to run a baseline update.

To transfer all configuration files to the production system:

1. In the staging environment, use Developer Studio and/or Endeca Workbench to make changes to the project.
2. Run `emgr_update` with an `--action` of `get_all_settings`.
 - a) For the `--host` parameter, specify the machine name and port for the staging Endeca Workbench environment.
 - b) For the `--dir` parameter, specify the Forge input directory in the production environment.
 - c) For the `--app_name` parameter, specify the application name whose instance configuration you want to transfer.
 - d) Use the `--filter` parameter to remove inactive business rules.

The following is a UNIX example:

```
emgr_update --host localhost:8006 --app_name My_application --action
get_all_settings --prefix wine --filter --dir /apps/endeca/data/forge_input
```

3. If the destination directory is not empty, you will be prompted to continue. Answer `y`.

When the utility finishes, all project configuration files (including the project and pipeline files) are copied to the production directory specified by the `--dir` parameter.

4. Use the Endeca Application Controller to run a baseline update on the production system.

The utility uses `prefix.esp` as the name of the output Developer Studio project file (where `prefix` is whatever you specified with the `--prefix` parameter). If there is an existing project file in the production directory with another name, it is recommended that you change it to `prefix.esp`.



Note: If you have loaded any auto-generated or external dimension values into Developer Studio using the load function, the ID assignments are not stored in the instance configuration. You must manually synchronize these files between implementation environments.

Transferring only instance configuration files modified by Endeca Workbench

You can transfer only those instance configuration files that can be modified in Endeca Workbench from a staging environment to a production environment.

In this task, you transfer files from the staging environment to the Forge input directory in the production environment. However, these files are the instance configuration files that can be modified by Endeca Workbench. They are not the full set of instance configuration files. Endeca Workbench can modify instance configuration files for any of the following features:

- Dynamic business rules
- Thesaurus entries
- Automatic phrases
- Stop words
- Dimension ordering

A subsequent baseline update uses the updated files for these features.

To transfer instance configuration files modified by Endeca Workbench to a production system:

1. In the staging environment, use Endeca Workbench to make any necessary instance configuration changes.
2. Run `emgr_update` with an `--action` of `get_ws_settings`.
 - a) For the `--host` parameter, specify the machine name and port for the staging Workbench environment.
 - b) For the `--dir` parameter, specify the Forge input directory in the production environment.
 - c) For the `--app_name` parameter, specify the application name whose instance configuration you want to transfer.
 - d) Use the `--filter` parameter to remove inactive business rules.

The following is a Windows example:

```
emgr_update.bat --host localhost:8888 --app_name My_application --action
get_ws_settings--prefix wine --filter --dir c:\endecaproduction\data\
forge_input
```

3. If the destination directory is not empty, you will be prompted to continue. Answer `y`. When the utility finishes, the project files that Endeca Workbench modified are copied to the production directory specified by the `--dir` parameter.
4. Use the Endeca Application Controller (EAC) to run a baseline update on the production system.

Transferring from one Workbench environment to another

The process for transferring and deploying instance configuration files from one Endeca Workbench environment to another, using the `emgr_update` utility is accomplished using one of the `emgr_update` utility's `--action` operations. The entire process can be scripted.

To transfer and deploy all instance configuration files to the production system:

1. In the staging environment, use Developer Studio, Endeca Workbench, or a combination of both to make changes to the project.
2. Run `emgr_update` with an `--action` of `get_all_settings`.
 - a) For the `--host` parameter, specify the machine name and port for the staging Endeca Workbench environment.
 - b) For the `--dir` parameter, specify the Forge input directory in the production environment.

- c) For the `--app_name` parameter, specify the application name whose instance configuration you want to transfer.
- d) Use the `--filter` parameter to remove inactive business rules.

The following is a Windows example:

```
emgr_update.bat --host localhost:8006 --app_name My_app --action
get_all_settings --prefix wine --filter --dir c:\endecaproduction\data\forge_input
```

3. If the destination directory is not empty, you will be prompted to continue. Answer `y`.

When the utility finishes, all project configuration files are copied to the production directory specified by the `--dir` parameter.

4. Run `emgr_update` with an `--action` of `update_mgr_settings`.
 - a) For the `--host` parameter, specify the machine name and port for the production environment in Endeca Workbench.
 - b) For the `--dir` parameter, specify the directory that contains the project configuration files that will be used to update the production environment in Endeca Workbench (typically, this will be the same directory that was used in step 2).
 - c) For the `--app_name` parameter, specify the application name whose instance configuration you want to transfer.

The following is a Windows example:

```
emgr_update.bat --host localhost:8006 --app_name My_app --action up-
date_mgr_settings --prefix wine --dir c:\endecaproduction\data\forge_input
```

5. Use the Endeca Application Controller (EAC) to run a baseline update on the production system.

Removing instance configuration files from Endeca Workbench

Deleting an application in the EAC Admin Console in Endeca Workbench does not remove its instance configuration files. If you want to delete the instance configuration files for the application, you can use `emgr_update`.

To remove instance configuration files:

- Run `emgr_update` with an `--action` of `remove_all_settings`.
 - a) For the `--host` parameter, specify the machine name and port for the staging environment in Endeca Workbench.
 - b) For the `--app_name` parameter, specify the application name whose instance configuration you want to remove.

The following is a Windows example:

```
emgr_update.bat --host localhost:8006 --app_name My_app --action re-
move_all_settings --prefix My_prefix
```

Sending the dimensions file produced by Forge to Endeca Workbench

If you are using your own scripts for running the baseline update, then after you run Forge, you need to send the dimensions file produced by Forge to the Endeca Workbench instance configuration for your application.



Note: Read this section only if you are not using a Deployment Template default script for running the baseline update, and are using your own scripts for this purpose.

To send the dimensions file produced by Forge to the Endeca Workbench, run `emgr_update` as follows:

- On the machine that has access to the output files of Forge (this is typically the machine on which you ran Forge), run `emgr_update` with an action of `set_post_forge_dims`:
 - a) For the `--host` parameter, specify the machine name and port for the environment in Endeca Workbench.
 - b) For the `--app_name` parameter, specify the application name whose instance configuration you want to update with this information.
 - c) For the `--post_forge_file` parameter, specify the full pathname to the output file where Forge stores its dimensions.

The following is a Windows example:

```
emgr_update.bat --host localhost:8006 --action set_post_forge_dims --
app_name wine --post_forge_file C:\sample_wine_data\data\parti-
tion0\forge_output\wine.dimensions.xml
```

Removing inactive rules from an instance configuration

You should remove dynamic business rules that have a state of inactive before transferring an Endeca implementation from staging to production.



Note: The `--filter` option of `emgr_update` removes inactive rules. This topic describes another option for manually removing inactive rules.

The **State** column in the rule list on the **Rule Manager** page of Web Studio indicates whether a rule is active or inactive. Additionally, you can examine the `merch_rule_group.xml` file for a rule and check whether a rule has the property `endeca.internal.workflow.state` set to `ACTIVE` or `INACTIVE`.

Removing inactive rules is not required but it is recommended. With the default rule filter in place, the MDEX Engine does not fire any rule whose state is inactive. In other words, you can transfer an instance configuration, including both active and inactive rules, and the MDEX Engine fires only active rules in reply to user queries.

To remove inactive dynamic business rules:

1. Create an XSLT file that strips out the inactive rules from each rule group's XML file. Use the following example by copying the XSLT below and pasting it into a text file:

```
<?xml version="1.0" encoding="utf-8"?>
  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">

  <!-- explicitly output doctype -->
  <xsl:output method="xml" doctype-system="merch_rule_group.dtd" />

  <!-- copy all elements and their attributes -->
  <xsl:template match="* | @"*>
```

```

<xsl:copy><xsl:copy-of select="@*" /><xsl:apply-templates/></xsl:copy>

</xsl:template>

<!-- strip out the MERCH_RULE elements who have a PROP specifying
      the workflow state as INACTIVE -->
<xsl:template
match="MERCH_RULE[PROP[@NAME='endeca.internal.workflow.state']][PVAL='IN-
ACTIVE']]"
></xsl:template>

</xsl:stylesheet>

```

2. Save the file with an `.xslt` suffix to a location of your choice.
3. Obtain and install an XSLT processor. Xalan is a popular XSLT processor if you do not have one already. You can download Xalan from <http://xml.apache.org/xalan-j/>.
4. Copy the rule group DTD file to the working directory for Xalan. In a default installation, the DTD is located in `C:\Endeca\MDEX\version\conf\dtd\merch_rule_group.dtd`.
5. Run the XSLT processor once per rule group passing in the name of the rule group's XML file. For example, if a project has five rule groups, run Xalan five times.

The processor applies the XSLT instructions to each rule group's XML file and outputs only active rules. There are several parameters the XSLT processor needs, including:

- The name and path to each rule group XML file. Files are named `merch_rule_group_name.xml` where the value of *name* is the literal rule group name.
 - The name and path to the XSLT file you created in step 1.
6. Continue transferring the modified instance configuration according the instructions for your environment.

Transferring auto-generated and external dimension value ID assignments

If you have loaded any auto-generated or external dimension values into Developer Studio using the load function, the ID assignments are not stored in the instance configuration. Therefore, you should manually synchronize these files between implementation environments.

Otherwise — unless your data is identical in each environment — IDs assigned by Forge may not match those assigned by Developer Studio. Mismatched ID assignments can affect project settings that depend on IDs stored in Developer Studio such as dimension value ordering, precedence rules, and business rules.

To transfer auto-generated and external dimension value ID assignments:

1. Navigate to the Forge state directory of the Endeca application instance that you use to modify dimension value ordering, precedence rules, and business rules.

For example:

```
C:\Apps\staging\myapp\data\state
```

2. Locate and back up the Forge state files containing the auto-generated and external dimension value ID assignments.

For example:

```
C:\Apps\staging\myapp\data\state\autogen_dimensions.xml.external.gz  
C:\Apps\staging\myapp\data\state\autogen_dimensions.xml.gz
```

3. Copy the Forge state files.
4. Paste these files into the Forge state directory of the application instance to which you are transferring your implementation.

For example:

```
C:\Apps\production\myapp\data\state
```

5. Repeat this process for any additional implementation environments.

Your auto-generated and external dimension value IDs are now identical in both environments. You should repeat this process whenever you modify dimension value ordering, precedence rules, or business rules.

Removing an application from Endeca IAP

To completely remove an application from Endeca IAP, you must remove the provisioning information with the EAC Admin Console and the instance configuration files with `emgr_update`.

Removing an application in the EAC Admin Console removes provisioning information for an application but does not remove instance configuration files. Running `remove_all_settings` in `emgr_update` removes instance configuration files but not provisioning information. To completely remove all information about an application, you remove the provisioning information with the EAC Admin Console and then remove the instance configuration files with `emgr_update`. If you do not perform both steps, you may store unnecessary or duplicate sets of files for an application.

To completely remove an application from the Endeca IAP:

1. In Endeca Workbench, log in to the application you want to remove.
2. On the EAC Admin Console page, click **Delete**. Alternatively, you can also perform steps 1 and 2 using an EAC Web services client. This removes the provisioning information.
3. Run `emgr_update` with an `--action` of `remove_all_settings` to delete the instance configuration files.

Related Links

[Removing instance configuration files from Endeca Workbench](#) on page 116

Deleting an application in the EAC Admin Console in Endeca Workbench does not remove its instance configuration files. If you want to delete the instance configuration files for the application, you can use `emgr_update`.

Index

A

- adding
 - Dgraphs 33
 - MDEX Engine servers 31
- admin operations
 - about 79
 - audit 82
 - auditreset 83
 - exit 82
 - flush 81
 - help 81
 - list of 79
 - logroll 84
 - ping 81
 - reload-services 86
 - restart 82
 - stats 83
 - statsreset 84
 - update 84
 - updateaspell 85
 - updatehistory 86
 - back up
- Agidx flags 91
- Agraph
 - checking aliveness of 67
 - flags 92
 - logs 70
- AppConfig.xml
 - about 16
 - schema for 16
- AppConfig.xml file
 - adding Dgraphs 33
 - adding MDEX Engine servers 31
 - multiple servers 29
- application
 - provisioning on multiple development servers 27
- application removal in the Endeca IAP 119
- Application server, defined 23
- applications
 - provisioning 47
- architecture of the EAC 12
- archiving Dgraph logs 53
- audit admin operation 82
- auditreset admin operation 83
- auto-generated dimension value ID assignments 118
- automating file collection 44

B

- backing up
 - CAS 76
 - different environments 78

- backing up (*continued*)
 - Discovery Framework 77
 - provisioning 48
 - required files 75
 - what not to back up 78

C

- Catalina
 - logs from EAC 56
- collect-app.bat 44, 78
- command-line scripts 40
- config operations
 - about 86
 - for logging verbosity 88
 - help 87
 - log-disable 89
 - log-enable 89
 - log-status 90
 - logging variables 88
 - update 87
- configuration operations
 - about 79
- configuring
 - additional servers and components 26
- connection errors
 - Dgraph and client 71
- control framework
 - choosing 17
- core dump files
 - in the Dgraph 73
 - managing 73
- custom.xml 41

D

- Data Processing (ITL) server, defined 23
- Deployment Template
 - AppConfig.xml 16
 - archiving Dgraph logs 53
 - changing the output prefix 108
 - directories 15
 - removing components 54
 - running 27
 - running Dgidx 60
 - specifying Dgraph arguments 67
- development server
 - multiple 23
- Dgidx
 - log details for text search indexing 64
 - log examples 63
 - memory usage 59
 - processing 59

Dgidx (*continued*)

- records with missing or duplicate record spec values 65
- running at the command prompt 60
- running with Deployment Template runcommand 60
- setting number of threads 96
- speeding up indexing 61
- troubleshooting 61
- variations in indexing time 66

Dgraph

- archiving logs 53
- checking aliveness of 67
- crash dump files on Windows 73
- identifying connection errors 71
- logs 68
- managing core dump files on Linux and Solaris 73
- specifying arguments in the Deployment Template 67
- what to collect for debugging 68

dimension value IDs

- transferring 118

E**EAC**

- changing the IP address for the Central Server 57
- checking the status of its components 49
- determining state, with service URLs 56
- logs for Central Server 56
- removing defunct processes in 50

EAC memory 51

EAC process logs

- increasing verbosity 70

emgr_update

- about 110
- syntax reference table 111

Endeca Application Controller

- about 12
- architecture 12
- architecture example 14

Endeca implementation

- administering 11

exit admin operation 82

external dimension value ID assignments 118

F

flush admin operation 81

Forge state files 40

forward slashes 43

H

help admin operation 81

help config operation 87

I

implementation

- transferring 109

installation packages

- where to install each package on multiple servers 23

instance configuration 40

- removing from Workbench 116
- retrieving for Workbench 110
- transferring files 114

instance configuration files

- transferring for an Endeca project 113

introduction 11

L

library files 40

locks in EAC

- releasing manually 53

log-disable config operation 89

log-enable config operation 89

log-status config operation 90

logging and reporting

- diagram of workflow 21

logging variables

- MDEX Engine 88
- operation syntax 88, 89
- supported variables for 88

logroll admin operation 84

M

manual resolution 45

MDEX Engine

- logging variables for 88
- ways to control 48

MDEX Engine server, defined 23

multiple development servers, about 23

O

operation syntax

- for MDEX Engine logging variables 88, 89

overview

- Deployment Template 14

P

Page Builder templates 40

partial updates

- troubleshooting 71

ping admin operation 81

pinging components 67

provisioning

- application, on multiple servers 27
- provisioning an application 47

R

- records
 - with missing or duplicate record spec values 65
- relatives paths 43
- releasing
 - EAC locks 53
- reload-services admin operation 86
- removing
 - an application 119
- removing components 54
 - and the Deployment Template 56
- replicating application definitions 45
- replicating_application_definitions 37
- restart admin operation 82
- running system operations 48

S

- separation of concerns 45
- staging vs. production environment
 - adding servers 26
- staging vs. production environment, defined 24
- stats admin operation 83
- statsreset admin operation 84

T

- this guide
 - about 9
- threads in Dgidx, setting 96
- token system 45
- Tomcat
 - logs from EAC 56
- Tools server, defined 23
- transferring
 - files between Workbench environments 115
- transferring implementations between environments 109
- transferring instance configuration files 114
- transferring Workbench instance configuration files 115

- troubleshooting
 - baseline updates 70
- troubleshooting:
 - Dgraph port and socket errors 72

U

- update admin operation 84
- update config operation 87
- updateaspell admin operation 85
- updatehistory admin operation 86
- upload.bat 44, 45
- URL operations
 - about 79

V

- variables
 - supported in MDEX Engine logging 88
- variables in scripts 43

W

- ways to control the MDEX Engine 48
- Who should use this guide 10
- Windows
 - Dgraph crash dump files 73
- Workbench
 - removing instance configuration from 116
 - retrieving instance configuration 110
 - transferring files between environments 115
- workflow diagram
 - for data and configuration 18
 - for logging and reporting 19

X

- XML configuration files
 - about 107
 - creating 108

