Oracle Endeca Deployment Template

Usage Guide

Version 3.2.2 • March 2012

ORACLE

ENDECA

Contents

Copyright and disclaimer

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 Guided Search is the most effective way for your customers to dynamically explore your storefront and find relevant and desired items quickly. An industry-leading faceted search and Guided Navigation solution, Oracle Endeca Guided Search enables businesses to help guide and influence customers in each step of their search experience. At the core of Oracle Endeca Guided Search 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

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

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

Who should use this guide

This guide is for developers starting to deploy new Oracle applications using the Deployment Template.

This template includes functionality required for a Dgraph deployment powered by the EAC and the Java EAC Development Toolkit, including support for baseline and partial index updates, Oracle Endeca Workbench integration, and support for Oracle Endeca CAS Server file system and CMS crawls.

This template also includes functionality required for an Agraph deployment powered by the EAC and the Java EAC Development Toolkit with support for baseline index updates with or without Parallel Forge enabled, and optional integration with Oracle Endeca Workbench. This document describes directories and script functionality and identifies touch-points where developers may need to configure or extend the template for their projects.

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 Endeca Customer Support

Oracle Endeca Customer 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 Endeca Customer Support through Oracle's Support portal, My Oracle Support at *https://support.oracle.com*.

Chapter 1

System Requirements and Installation

This section describes the Deployment Template requirements and the installation procedure.

Oracle Endeca software compatibility

To determine the compatibility of the Deployment Template with other Oracle Endeca installation packages, see the *Oracle Endeca Guided Search Compatibility Matrix* available on the Oracle Technology Network.

Java requirements

The EAC Development Toolkit requires Sun's Java 5 JRE or newer. By default, the Deployment Template will use the Java 6 JRE included in the Platform Services installation.

Perl requirements

The Deployment Template requires Perl 5.8.3, which is included in the Platform Services installation.

Platform requirements

The Deployment Template supports all system architectures and operating systems that are supported by Oracle Endeca Guided Search.

When installed on a UNIX system, the Deployment Template requires an English locale.

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

Installation

The Deployment Template is distributed as a zip file named `deploymentTemplate-[VERSION].zip`.

Extract the zip file, using WinZip or an alternate decompression utility, into any location. The package will unpack into a self-contained directory structure tree:

```
Endeca\Solutions\deploymentTemplate-[VERSION]\
```

Oracle recommends that you extract the ZIP file into the same directory as the Oracle Endeca software. For example, if you have installed Platform Services on Windows in to the following location:

```
C:\Endeca\PlatformServices\6.1.2\
```

Extract the ZIP file in to C:\ so that the Deployment Template installs into:

```
C:\Endeca\Solutions\deploymentTemplate-[VERSION]\
```

After installation, there are five directories - `bin`, `conf`, `data`, `doc`, and `lib` under the `deployment¬ Template-[VERSION]` directory.

Migrating from a previous version

This topic provides high-level instructions on how to migrate your deployment application to the current version of the Deployment Template.

Due to the flexible nature of the Deployment Template and the opportunities for customization, specifying a comprehensive migration path to Deployment Template 3.2 from a previous version is not possible. It is the job of Deployment Template users to know how they have customized their deployment so that the appropriate modifications can be retained.

However, it is possible to follow high-level steps that guide your migration path. Above all, you must be aware of the customizations that you made to the previous Deployment Template files so that you can port them to the newer version.

To upgrade to Deployment Template 3.2 from a previous release:

1. Upgrade to Platform Services 6.1 and Oracle Endeca Workbench 2.1.
2. Optionally, upgrade to Content Acquisition System 2.2.
3. Install Deployment Template 3.2, as documented in the "Installation" topic in this chapter.

   Make sure you read the new release notes to see if any changes will affect your application.
4. Run the Deployment Template 3.2 deploy script to generate a new application folder.
5. Port the changes that you made to the old `AppConfig.xml` over to the Deployment Template 3.2 `AppConfig.xml`.
6. Replace the Deployment Template 3.2 pipeline with your custom pipeline.
7. Place your input source data in the appropriate directory.
8. Drop any custom Java packages into the Deployment Template `lib/java` directory.
9. Run the `initialize_services` script, load the source data, and run a baseline update.

If everything runs correctly, you can then migrate other components of your application, such as your CAS crawl configurations.

Chapter 2

Application Deployment

This section describes the application deployment tasks.

Deployment prerequisites

Before beginning application deployment, ensure that the Oracle Endeca components have been installed on all servers that will make up your deployment environment and that environment variables used by the Oracle Endeca software (including `ENDECA_ROOT`) are set.

For more details about setting up your deployment environment, see the *Oracle Endeca Guided Search Getting Started Guide*.

EAC applications

In order to begin development, the template must be deployed onto the primary controller server in the deployment environment. Installation scripts have been provided for Windows and UNIX, relying on a common Perl installer to perform the deployment steps.

In every deployment environment, one server serves as the primary control machine and hosts the EAC Central Server, while all other servers act as agents to the primary server and host EAC Agent processes that receive instructions from the Central Server. Both the EAC Central Server and the EAC Agent run as applications inside the Endeca HTTP Service. The Deployment Template only needs to be installed on the selected primary server, which typically is the machine that hosts the Central Server

Deploying an EAC application on Windows

This section describes the steps for deploying an EAC application in a Windows environment using the provided `deploy.bat` file.

The `deploy.bat` script in the `bin` directory configures and distributes the template files into the deployment directory structure.

To deploy an application:

1. On the primary server, start a command prompt and navigate to `deploymentTemplate-[VER¬SION]\bin`.
2. From the `bin` directory, run the `deploy` script.

For example:

```
C:\Endeca\Solutions\deploymentTemplate-3.2\bin>deploy
```

The template identifies the location and version of your Platform Services installation based on the ENDECA_ROOT environment variable. If the information presented by the installer does not match the version or location of the software you plan to use for the deployment, stop the installation, reset your ENDECA_ROOT environment variable, and start again. Note that the installer may not be able to parse the Platform Services version from the ENDECA_ROOT path if it is installed in a non-standard directory structure. It is not necessary for the installer to parse the version number, so if you are certain that the ENDECA_ROOT path points to the correct location, proceed with the installation.

3. Specify whether the application is a Dgraph deployment or an Agraph deployment.
   a) If an Agraph deployment was selected, specify whether your application is going to use Parallel Forge.
      The template identifies the location and version of your Platform Services install based on the ENDECA_ROOT environment variable. If the information presented by the installer does not match the version or location of the software you plan to use for the deployment, stop the installation, reset your ENDECA_ROOT environment variable, and start again. Note that the installer may not be able to parse the Platform Services version from the ENDECA_ROOT path if it is installed in a non-standard directory structure. It is not necessary for the installer to parse the version number, so if you are certain that the ENDECA_ROOT path points to the correct location, proceed with the installation.
4. Specify a short name for your application.

   The name should consist of lower- or uppercase letters, or digits between zero and nine.
5. Specify the full path into which your application should be deployed.

   This directory must already exist. The `deploy` script creates a folder inside of the deployment directory with the name of your application and the application directory structure.

   For example, if your application name is `MyApp`, and you specify the deployment directory as `C:\Endeca\apps`, the `deploy` script installs the template for your application into `C:\Endeca\apps\MyApp`.
6. Specify the port number of the EAC Central Server.

   By default, the Central Server host is the machine on which you are running `deploy` script and that all EAC Agents are running on the same port. All of these settings can be configured to reflect a different environment configuration once the application is deployed.
7. Specify whether your application will use Oracle Endeca Workbench for configuration management.

   Integration can be enabled or disabled after deployment, but enabling it deploys a default set of scripts that manage configuration integration from Oracle Endeca Workbench.
8. Specify the port number of Oracle Endeca Workbench.

   By default, the installer assumes that the Workbench host is the machine on which it is being run, but this can be re-configured once the application is deployed.
9. The installer can be configured to prompt the user for custom information specific to the deployment.

   By default, Dgraph deployments use this functionality to prompt the user for Dgraph and Log Server port numbers, while the Agraph deployment also ask for Agraph (and Forge server, if using Parallel Forge) port numbers.

If the application directory already exists, the installation script archives the existing directory, to avoid accidental loss of data. For example, if the installer finds that `C:\Endeca\apps\MyApp` already exists, it renames the existing directory to `C:\Endeca\apps\MyApp.[timestamp]`.bak and installs

the new deployment into a new MyApp directory. Note that in the case of deployments that use a large amount of disk space, it is important to remove archived deployment directories to clear up disk space.

Related Links

Deploying an EAC application on UNIX on page 15
This section describes the steps for deploying an EAC application in a UNIX environment using the provided `deploy.sh` file.

Integrating and running CAS crawls that write to Record Store instances on page 79
This section describes the high-level steps for integrating and running a CAS crawl that writes output to a Record Store instance.

Integrating and running CAS crawls that write to record output files on page 89
This section describes the high-level steps for integrating and running a CAS crawl that writes output to a record output file.

Deploying an EAC application on UNIX

This section describes the steps for deploying an EAC application in a UNIX environment using the provided `deploy.sh` file.

Before deployment, unpack the Deployment Template package on the primary server. Five directories - `bin`, `conf`, `data`, `doc`, and `lib` - are installed into the `deploymentTemplate-[VERSION]` directory.

The shell script in the `bin` directory is used in the following deployment steps to configure and distribute the template files into the deployment directory structure.

To deploy the application:

1. On the primary server, execute the batch script to deploy the application:
   `deploymentTemplate-[VERSION]/bin/deploy.sh`
   The template identifies the location and version of your Platform Services install based on the `EN¬DECA_ROOT` environment variable. If the information presented by the installer does not match the version or location of the software you plan to use for the deployment, stop the installation, reset your `ENDECA_ROOT` environment variable, and start again. Note that the installer may not be able to parse the Platform Services version from the `ENDECA_ROOT` path if it is installed in a non-standard directory structure. It is not necessary for the installer to parse the version number, so if you are certain that the `ENDECA_ROOT` path points to the correct location, proceed with the installation.
2. Specify whether your application is going to be a Dgraph deployment or an Agraph deployment.
   a) If an Agraph deployment was selected, specify whether your application is going to use Parallel Forge.
3. Specify a short name for your application. The name should consist of lower- or uppercase letters, or digits between zero and nine.
4. Specify the full path into which your application should be deployed.
   This directory must already exist. The installation creates a folder inside of the deployment directory with the name of your application and the application directory structure and files will be deployed there. For example, if your application name is MyApp, specifying the deployment directory as `/localdisk/apps` installs the template for your application into `/localdisk/apps/myapp`.
5. Specify the port number of the EAC Central Server.
   By default, the installer assumes that the Central Server host is the machine on which it is being run and that all EAC Agents are running on the same port. All of these settings can be configured to reflect a different environment configuration once the application is deployed.

6. Specify whether your application will use Oracle Endeca Workbench for configuration management.

   Integration can be enabled or disabled after deployment, but enabling it deploys a default set of scripts that manage configuration integration from Oracle Endeca Workbench.

7. Specify the port number of Oracle Endeca Workbench.

   By default, the installer assumes that the Workbench host is the machine on which it is being run, but this can be re-configured once the application is deployed.

8. The installer can be configured to prompt the user for custom information specific to the deployment.

   By default, Dgraph deployments use this functionality to prompt the user for Dgraph and Log Server port numbers, while the Agraph deployment also ask for Agraph (and Forge server, if using Parallel Forge) port numbers.

If the application directory already exists, the installation script archives the existing directory, to avoid accidental loss of data. For example, if the installer finds that `/localdisk/apps/myapp` already exists, it renames the existing directory to `/localdisk/apps/myapp.[timestamp].bak` and installs the new deployment into a new myapp directory. Note that in the case of deployments that use a large amount of disk space, it is important to remove archived deployment directories to clear up disk space.

Related Links

Deploying an EAC application on Windows on page 13
This section describes the steps for deploying an EAC application in a Windows environment using the provided `deploy.bat` file.

Integrating and running CAS crawls that write to Record Store instances on page 79
This section describes the high-level steps for integrating and running a CAS crawl that writes output to a Record Store instance.

Integrating and running CAS crawls that write to record output files on page 89
This section describes the high-level steps for integrating and running a CAS crawl that writes output to a record output file.

Configuring an automated/file-based deployment

The installer script provides a file-based configuration option to simplify the deployment and installation of the Deployment Template. This may be especially useful during development, when the same deployment process must be repeated many times.

A sample configuration file is provided in the `conf` subdirectory of the `deploymentTemplate-[VERSION]` install directory. The install configuration file should specify the deployment type, application name and deployment path. The following example specifies the installation of a Dgraph deployment with Workbench integration enabled:

```
<install app-name="MyApp" >
  <deployment-path>C:\Endeca</deployment-path>
  <base-module type="dgraph" />
  <options>
    <option name="eac-port">8888</option>
    <option name="workbench-enabled">true</option>
    <option name="workbench-port">8006</option>
    <option name="dgraph1Port">15000</option>
    <option name="dgraph2Port">15001</option>
    <option name="logserverPort">15010</option>
```

```
  </options>
</install>
```

The following example specifies the installation of an Agraph deployment with Parallel Forge enabled and Workbench integration disabled:

```
<install app-name="MyApp" >
  <deployment-path>C:\Endeca</deployment-path>
  <base-module type="agraph" />
  <options>
    <option name="eac-port">8888</option>
    <option name="workbench-enabled">false</option>
    <option name="parallel-forge">true</option>
    <option name="dgraph1Port">15000</option>
    <option name="dgraph2Port">15001</option>
    <option name="agraph1Port">14000</option>
    <option name="agraph2Port">14001</option>
    <option name="forgeServerPort">14099</option>
    <option name="logserverPort">15010</option>
  </options>
</install>
```

To specify an install configuration file:

Specify the `--install-config` flag to the deploy batch or shell script to specify the location of an install configuration file.
The following example specifies that argument to the Windows installation script:

```
deploymentTemplate-[VERSION]\bin\deploy.bat --install-config ..\conf\in¬
stall_config.xml
```

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

Custom applications

This section provides information about deploying custom applications.

Custom application descriptors

The Deployment Template installer is driven off of application descriptor XML documents that describe the directory structure associated with the deployment as well as the files to distribute during the installation process.

By default, the installer is shipped with application descriptor files for Agraph and Dgraph deployments powered by the EAC Development Toolkit (located in the conf subdirectory of the `deploymentTem¬plate-[VERSION]` install directory).

This document describes the directory structure of the deployment as well as the copying that is done during the installation to distribute files into the new directories. Additionally, this document describes whether files are associated with a Windows or UNIX deployment, and whether copied files should be

updated to replace tokens in the format @@TOKEN_NAME@@ with text strings specified to the installer.

The following tokens are handled by the installer by default:

- @@EAC_PORT@@ - EAC Central Server port.
- @@HOST@@ - Hostname of the server on which the deploy script is invoked.
- @@PROJECT_DIR@@ - Absolute path of the target deployment directory.
- @@ESCAPED_PROJECT_DIR@@ - Absolute path of the target deployment directory using only forward slashes.
- @@PROJECT_NAME@@ - Name of the application to deploy.
- @@ENDECA_ROOT@@ - Absolute path of the ENDECA_ROOT environment variable.
- @@SCRIPT_SUFFIX@@ - ".bat" for Windows, ".sh" for Linux installs.
- @@SLASH@@ - "\" for Windows, "/" for Linux installs.
- @@PARALLEL_FORGE_ENABLED@@ - "true" or "false," indicating whether parallel forge is enabled for the Agraph deployment.
- @@WORKBENCH_ENABLED@@ - "true" or "false," indicating whether Workbench integration is enabled for the application.
- @@WORKBENCH_PORT@@ - Oracle Endeca Workbench port.

In addition to these default tokens, users can specify custom tokens to substitute in their files. Tokens are specified in the application descriptor file, including the name of the token to substitute as well as the question with which to prompt the user or the installer configuration option to parse to retrieve the value to substitute for the token. The default application descriptors use this functionality to request the port number for Dgraphs, Agraphs, Log Servers and Forge servers.

Projects that deviate from the Deployment Template directory structure may find it useful to create a custom application descriptor document, so that the Deployment Template installer can continue to be used for application deployment.

Custom deployment descriptors may also be used to define add-on modules on top of a base install. For example, sample applications (such as the Sample Term Discovery and Clustering application) are shipped with a custom deployment descriptor file, which describes the additional files and directories to install on top of a base Dgraph deployment. Modules may be installed using the deploy batch or shell script, specifying the `--app` argument with the location of the application descriptor document. For example:

```
deploy.bat --app \
C:\Endeca\Solutions\sampleTermDiscovery-[VERSION]\data\deploy.xml
```

The installer prompts you to specify whether it should install the module as a standalone installation or if it should be installed on top of the base Dgraph deployment. Multiple add-on modules may be specified to the installer script, though only one of them may be a base install (that is, all but one of them should specify an attribute of update= "true").

The following excerpt from the Dgraph deployment application descriptor identifies the document's elements and attributes:

```
<!--
  Deployment Template installer configuration file. This file defines the
directory structure to create and the copies to perform to distribute files
 into the new directory structure.

  The update attribute of the root install element indicates whether this
is a core installation or an add-on module. When set to false or unspecified,
 the installation requires the removal of an existing target install direc¬
tory (if present). When update is set to true, the installer preserves any
 existing directories, adding directories as required and distributing files
 based on the specified copy pattern.
```

```
-->
<app-descriptor update="false" id="Dgraph">

  <custom-tokens>
    <!-- Template custom token:
      <token name="MYTOKEN">
        <prompt-question>What is the value to substitute for token MYTO¬
KEN?</prompt-question>
        <install-config-option>myToken</install-config-option>
        <default-value>My Value</default-value>
      </token>

      This will instruct the installer to look for the "myToken" option
      in a specified install config file (if one is specified) or to
      prompt the user with the specified question to submit a value. If a
      value is entered/retrieved, the installer will substitute instances
      of @@MYTOKEN@@ with the value.
    -->
  </custom-tokens>

  <dir-structure>
    <!-- Template directory:
      <dir platform="unix" primary="true"></dir>

      primary           builds directory only on primary server installs

      platform          builds directory only on specified platform.
                          Valid values: "win" and "unix"
    -->
  </dir-structure>

  <!—
  Copy source directory is specified relative to this file's directory
  -->
  <copy-pattern src-root="../data ">
    <!-- Template copy pattern:
      <copy clear-dest-dir="true" recursive="true"
            preserve-subdirs="true" filter-files="true"
            primary="true" platform="win" Endeca-version="480">
        <src-dir></src-dir>
        <src-file></src-file>
        <dest-dir></dest-dir>
      </copy>

      src-dir           source directory, relative to root of deployment
                          template package.

      src-file          source filename or pattern (using '*' wildcard
                          character) to copy from source dir

      dest-dir          destination directory, relative to root of target
                          deployment directory.

      clear-dest-dir    removes all files in target dir before copying

      recursive         copies files matching pattern in subdirectories
                          of the specified source dir

      preserve-subdirs  copies files, preserving dir structure. Only
                          applicable to recursive copies
```

```
      filter-files      filters file contents and file names by replacing
                          tokens (format @@TOKEN@@) with specified
                          strings.

      mode              applies the specified permissions to the files
                          after the copy. Mode string should be 3 octal
                          digits with an optional leading zero to
                          indicate octal, e.g. 755, 0644. Not relevant
                          for Windows deployments.

      platform          applies copy to specified platform. Valid
                          values: "win" "unix"

      Endeca-version    applies copy to specified Oracle Endeca version
Valid
                          values: "460" "470" "480" "500"
    -->
  </copy-pattern>
</app-descriptor>
```

Configuring an automated/file-based deployment for a custom application

The install configuration file discussed in previous sections may be used to specify the location of custom application descriptor documents in lieu of the `--app` command line argument to the installer.

The following example shows how to install the Sample Term Discovery and Clustering application on top of the base Dgraph deployment.

```
<install app-name="MyApp" >
  <deployment-path>C:\Endeca</deployment-path>
  <base-module type="dgraph" />
  <additional-module type="custom">
    C:\Endeca\Solutions\sampleTermDiscovery-[VERSION]\data\deploy.xml
  </additional-module>
  <options>
    <option name="eac-port">8888</option>
    <option name="workbench-enabled">false</option>
    <option name="dgraph1Port">15000</option>
    <option name="dgraph2Port">15001</option>
    <option name="logserverPort">15010</option>
  </options>
</install>
```

Usage and development

Once installed, the Deployment Template includes all of the scripts and configuration files required to create an index and start a cluster of MDEX Engines.

The template includes the pipeline configuration files and data extract files from the `sample_wine_data` reference project that ships with Platform Services. It also includes a pipeline configuration intended for file system and CMS crawls. The following sections document the process of configuring the installed deployment to run your application.

Displaying the Deployment Template version

You can print out the version number of the Deployment Template from the command line.

The `runcommand` script has a `--version` flag that prints the version number of the Deployment Template and exits. The command actually prints the version number of the EAC Development Toolkit.

Displaying the version is important for troubleshooting purposes.

To display the version of the Deployment Template:

1. From a command prompt, navigate to the `[appdir]\control` directory on Windows (`[appdir]/control` on UNIX).
2. Run the `runcommand` script with the `--version` flag, as in this Windows example:

```
C:\Endeca\Apps\control>runcommand --version
```

The command prints the version, as in this sample output:

```
EAC Toolkit: 3.1.734 - 2009-05-18T08:11:11-0400
```

The third number ("734" in the example) is the build number, followed by the build date/time.

Configuring an application

This section guides you through the process of configuring the deployment to run your application.

1. Start the EAC on each server in the deployment environment.

   If this is a UNIX deployment, use the `$ENDECA_ROOT/tools/server/bin/startup.sh` shell script to start the EAC with the configuration files specified in your `$Endeca_CONF` directory. If this is a Windows deployment, ensure the Endeca HTTP Service is running, as it is the parent service that contains the EAC. The files in the workspace directory specify the settings for the EAC, including the port of the EAC Central Server and the EAC Agent server ports (on all servers, including the primary server) and SSL settings for the EAC.

   If the application is intended to integrate with Oracle Endeca Workbench, ensure that the Workbench is running.

2. Edit the `AppConfig.xml` file in `[appdir]/config/script` to reflect the details of your environment. Specifically:
   - Ensure that the `eacHost` and `eacPort` attributes of the app element specify the correct host and port of the EAC Central Server.
   - Ensure that the `host` elements specify the correct host name or names and EAC ports of all EAC Agents in your environment.
   - Ensure that the `ConfigManager` component specifies the correct host and port for Oracle Endeca Workbench, or that Workbench integration is disabled.

   In addition to checking the host and port settings, you should configure components (for example, add or remove Dgraphs to specify an appropriate Dgraph cluster for your application), adjust process flags if necessary, and select appropriate ports for each Dgraph and Logserver.

3. Run the `initialize_services` script to initialize each server in the deployment environment with the directories and configuration required to host your application.

This script removes any existing provisioning associated with this application in the EAC and then adds the hosts and components in your provisioning document to the EAC, creating the directory structure used by these components on all servers. In addition, if Workbench integration is enabled, this script initializes Oracle Endeca Workbench by uploading the application's configuration files.

- On Windows:
  ```
  [appdir]\control\initialize_services.bat
  ```
- On UNIX:
  ```
  [appdir]/control/initialize_services.sh
  ```

Use caution when running this script. The script forces any components that are defined for this application to stop, which may lead to service interruption if executed on a live environment. The script also removes any current Workbench configuration and removes any rules not maintained in `[appdir]/config/pipeline`.

4. Upload `forge_input` configuration to the primary server.

   Replace the sample wine configuration files in `[appdir]/config/pipeline` with the configuration files created in Developer Studio for your application. For details about inserting your custom project pipeline into the Deployment Template, refer to the section *Inserting a Custom Pipeline* on page 109.

5. If necessary, upload new source data to the primary server.

   Replace the sample wine data (`wine_data.txt.gz`) in `[appdir]/test_data/baseline` with the data used by your application. Similarly, replace the partial update data in `[appdir]/test_data/partial` with your application's partial data.

   This step is not necessary if your application does not use data extracts (for example, if your application retrieves data directly from a database via ODBC or JDBC or from a CAS crawl). If this is the case, remove the `wine_data.txt.gz` file from `[appdir]/test_data/baseline`.

If no script customization is required, the application is now ready for use. During development, use the `load_baseline_test_data` script to simulate the data extraction process (or data readiness signal, in the case of an application that uses a non-extract data source). This script delivers the data extract in `[appdir]/test_data/baseline` and runs the `set_baseline_data_ready_flag` script, which sets a flag in the EAC indicating that data has been extracted and is ready for baseline update processing. In production, this step should be replaced with a data extraction process that delivers extracts into the incoming directory and sets the "`baseline_data_ready`" flag in the EAC. This flag can be set by making a Web service call to the EAC or by running the provided `set_baseline_data_ready_flag` script.

Related Links

Integrating and running CAS crawls that write to Record Store instances on page 79
This section describes the high-level steps for integrating and running a CAS crawl that writes output to a Record Store instance.

Integrating and running CAS crawls that write to record output files on page 89
This section describes the high-level steps for integrating and running a CAS crawl that writes output to a record output file.

Running the sample baseline update scripts

Run the sample test scripts to create your first index and start the MDEX Engines in your deployment environment.

To run a baseline update:

1. Run the `load_baseline_test_data` script.
   - On Windows:
     ```
     [appdir]\control\load_baseline_test_data.bat
     ```
   - On UNIX:
     ```
     [appdir]/control/load_baseline_test_data.sh
     ```
2. Run the `baseline_update` script.
   - On Windows:
     ```
     [appdir]\control\baseline_update.bat
     ```
   - On UNIX:
     ```
     [appdir]/control/baseline_update.sh
     ```

Related Links

Running the sample partial update scripts on page 23
In addition to the baseline update scripts, the deployment template provides a set of sample partial update scripts.

Running the sample partial update scripts

In addition to the baseline update scripts, the deployment template provides a set of sample partial update scripts.

To run a partial update:

1. Run the `load_partial_test_data` script.
   - On Windows:
     ```
     [appdir]\control\load_partial_test_data.bat
     ```
   - On UNIX:
     ```
     [appdir]/control/load_partial_test_data.sh
     ```
2. Run the `partial_update` script.
   - On Windows:
     ```
     [appdir]\control\partial_update.bat
     ```
   - On UNIX:
     ```
     [appdir]/control/partial_update.sh
     ```

Related Links

Running the sample baseline update scripts on page 23
Run the sample test scripts to create your first index and start the MDEX Engines in your deployment environment.

Communicating with SSL-enabled Oracle Endeca components

The Deployment Template supports enabling SSL to communicate securely with the EAC Central Server and with the CAS Server for version 3.0.x and later. (Secure communication between the Deployment Template and the CAS Server is not supported in CAS 2.2.x.)

For details about enabling SSL in the EAC Central Server or Agent, refer to the *Oracle Endeca Security Guide*. For details about enabling SSL in CAS, refer to the *CAS Developer's Guide*.

To use the template with an SSL-enabled Central Server:

1. Update `runcommand.bat/.sh` to load your SSL keystore and truststore.

Note: To enable secure communication, you must have already followed the documentation to create a Java keystore and truststore, containing your generated certificates. Upload a copy of these certificates to the server on which your Deployment Template scripts will run. Edit the `runcommand` file to specify the locations of these files.

- On Windows, edit `runcommand.bat` to add the following lines:

```
...

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

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

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

set JAVA_ARGS=%JAVA_ARGS% "-Djavax.net.ssl.trustStore=%TRUSTSTORE%" "-
Djavax.net.ssl.trustStoreType=JKS" "-Djavax.net.ssl.trustStorePass¬
word=[truststore password]"

set JAVA_ARGS=%JAVA_ARGS% "-Djavax.net.ssl.keyStore=%KEYSTORE%" "-
Djavax.net.ssl.keyStoreType=JKS" "-Djavax.net.ssl.keyStorePassword=[key¬
store password]"

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

```
...
```

Note that the final two new lines (beginning with "`set JAVA_ARGS`" are wrapped to fit the page size of this document, but each of those two lines should have no line breaks. Also note that you need to fill in the locations and passwords of your keystore and truststore files in the locations indicated by the placeholders in italics.

- On UNIX, edit `runcommand.sh` as follows:

```
...

JAVA_ARGS="${JAVA_ARGS} -Djava.util.logging.config.file=${WORK¬
ING_DIR}/../config/script/logging.properties"

if [ -f "[/path/to/truststore]" ] ; then
  if [ -f "[/path/to/keystore]" ] ; then
    TRUSTSTORE=[/path/to/truststore]
    KEYSTORE=[/path/to/keystore]
    JAVA_ARGS="${JAVA_ARGS} -Djavax.net.ssl.trustStore=${TRUSTSTORE}"
    JAVA_ARGS="${JAVA_ARGS} -Djavax.net.ssl.trustStoreType=JKS"
   JAVA_ARGS="${JAVA_ARGS} -Djavax.net.ssl.trustStorePassword=[trust¬
store password]"
    JAVA_ARGS="${JAVA_ARGS} -Djavax.net.ssl.keyStore=${KEYSTORE}"
    JAVA_ARGS="${JAVA_ARGS} -Djavax.net.ssl.keyStoreType=JKS"
   JAVA_ARGS="${JAVA_ARGS} -Djavax.net.ssl.keyStorePassword=[keystore
 password]"
  else
    echo "WARNING: Cannot find keystore at [/path/to/keystore]. Secure
 EAC communication may fail."
  fi
else
  echo "WARNING: Cannot find truststore at [/path/to/truststore]. Secure
 EAC communication may fail."
fi

CONTROLLER_ARGS="--app-config AppConfig.xml"

...
```

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

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

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

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

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

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

5. Specify the non-SSL connector for hosts.

Internally, the EAC Central Server always initiates communication with Agents by communicating with the non-SSL connector. When the Agent is SSL-enabled, the non-secure port redirects communication to the secure port. In both cases, the appropriate configuration is to specify the non-secure port for provisioned hosts.

6. Specify the non-SSL connector for Oracle Endeca Workbench.

   In the `ConfigManager` component, the property `webStudioPort` should specify the non-secure connector for the Endeca Tools Service, as communication with Oracle Endeca Workbench configuration store always uses the unsecured channel.

The following excerpt from the AppConfig.xml document shows a sample configuration for an SSL-enabled application.

```
<!—
#############################################################################
# Global variables
#
-->
<app appName="MySslApp" eacHost="host-lt1" eacPort="8443"
    dataPrefix="MySslApp" sslEnabled="true" lockManager="LockManager">
  <working-dir>C:\Endeca\Apps\MySslApp</working-dir>
  <log-dir>./logs</log-dir>
</app>

<!--
#############################################################################
# Servers/hosts
#
-->
<host id="ITLHost" hostName="myhost1.company.com" port="8888" />
<host id="MDEXHost" hostName="myhost2.company.com" port="8888" />

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

<custom-component id="CAS" host-id="CASHost" class="com.Oracle Ende¬
ca.eac.toolkit.component.cas.ContentAcquisitionServerComponent">
  <properties>
    <property name="casHost" value="localhost" />
    <property name="casPort" value="8505" />
  </properties>
</custom-component>

-->
```

Customizations

The standard processing and script operations of the Deployment Template are sufficient to support the operational requirements of most projects. Some applications require customization to enable custom processing steps, script behavior, or even directory structure changes.

Developers are encouraged to use the template as a starting point for customization. The scripts and modules provided with the template incorporate Oracle's best practice recommendations for synchronization, archiving, and update processing. The Deployment Template is intended to provide a set of standards on which development should be founded, while allowing the flexibility to develop custom scripts to meet specific project needs.

The following list describes a number of customization approaches that can be implemented to extend the existing functionality or add new functionality to the template. For further details about configuration and customization in the `AppConfig.xml` document, refer to Appendix A ("EAC Development Toolkit").

- Configure `AppConfig.xml` - The simplest form of configuration consists of editing the `AppConfig.xml` configuration document to change the behavior of components or to add or remove components. This type of configuration includes the addition of removal of Dgraphs to the main cluster or even the creation of additional clusters. In addition, this category includes adjustment of process arguments (for example, adding a Java classpath for the Forge process in order to enable the use of a Java Manipulator), custom properties and directories (for example, changing the number of index archives that are stored on the indexing server).
- Change behavior of existing BeanShell scripts - Scripts are written in the Java scripting language BeanShell. Scripts are defined in the `AppConfig.xml` document and are interpreted at runtime by the BeanShell interpreter. This allows developers and system administrators to adjust the behavior of the baseline, partial, and configuration update scripts by simply modifying the configuration document. For example, if a deployment uses JDBC to read data into the Forge pipeline instead of using extracted data files, the following changes would be implemented in the BaselineUpdate script:

  1. Remove the line that retrieves data and configuration for Forge: `Forge.getData();`
  2. Insert a new copy command to retrieve configuration for Forge to process:

```
...
// get Web Studio config, merge with Dev Studio config
ConfigManager.downloadWsConfig();
ConfigManager.fetchMergedConfig();

// fetch extracted data files, run ITL
srcDir = PathUtils.getAbsolutePath(Forge.getWorkingDir(),
  Forge.getConfigDir()) + "/\\*";
destDir = PathUtils.getAbsolutePath(Forge.getWorkingDir(),
  Forge.getInputDir());

dimensionCopy = new CopyUtility(Forge.getAppName(),
  Forge.getEacHost(), Forge.getEacPort(), Forge.isSslEnabled());
dimensionCopy.init("copy_dimensions", Forge.getHostId(),
  Forge.getHostId(), srcDir, destDir, true);
dimensionCopy.run();

Forge.getData();
Forge.run();
Dgidx.run();
...
```

  Note that this amended BeanShell script imports two classes from the classpath, references variables that point to elements in the `AppConfig.xml` document (e.g. Forge, Dgidx) and defines new variables without specifying their type (e.g. srcDir, destDir). Details about BeanShell scripting can be found in Appendix A of this guide.
- Write new BeanShell scripts - Some use cases may call for greater flexibility than can easily be achieved by modifying existing BeanShell scripts. In these cases, writing new BeanShell scripts

may accomplish the desired goal. For example, the following BeanShell script extends the previous example by pulling the new functionality into a separate script:

```
<script id="CopyConfig">
  <bean-shell-script>
    <![CDATA[

  // fetch extracted data files, run ITL
  srcDir = PathUtils.getAbsolutePath(Forge.getWorkingDir(),
    Forge.getConfigDir()) + "/\\*";
  destDir = PathUtils.getAbsolutePath(Forge.getWorkingDir(),
    Forge.getInputDir());

  dimensionCopy = new CopyUtility(Forge.getAppName(),
    Forge.getEacHost(), Forge.getEacPort(), Forge.isSslEnabled());
  dimensionCopy.init("copy_dimensions", Forge.getHostId(),
    Forge.getHostId(), srcDir, destDir, true);
  dimensionCopy.run();

    ]]>
  </bean-shell-script>
</script>
```

Once the new script is defined, the BaselineUpdate script simplifies to the following:

```
...
// get Web Studio config, merge with Dev Studio config
ConfigManager.downloadWsConfig();
ConfigManager.fetchMergedConfig();

// fetch extracted data files, run ITL
CopyConfig.run();
Forge.getData();
Forge.run();
Dgidx.run();
...
```

- Define utilities in `AppConfig.xml` - A common use case for customization is to add or adjust the functionality of utility invocation. Our previous example demonstrates the need to invoke a new copy utility when the Forge implementation changes. Other common use cases involve invoking a data pre-processing script from the shell and archiving a directory. In order to enable this, the Deployment Template allows utilities to be configured in the `AppConfig.xml` document. To configure the copy defined above in the document, use the copy element:

  ```
  <copy id="CopyConfig" src-host-id="ITLHost" dest-host-id="ITLHost"
    recursive="true">
    <src>./data/complete_index_config/*</src>
    <dest>./data/processing</dest>
  </copy>
  ```

  Once configured, this copy utility is invoked using the same command that was previously added to the BaselineUpdate to invoke the custom BeanShell script: `CopyConfig.run();`
- Extend the Java EAC Development Toolkit - In rare cases, developers may need to implement complex custom functionality that would be unwieldy and difficult to maintain if implemented in the `AppConfig.xml` document. In these cases, developers can extend objects in the toolkit to create new Java objects that implement the desired custom functionality. Staying with the previous example, the developer might implement a custom Forge object to change the behavior of the `getData()` method to simply copy configuration without looking for extracted data files.

  ```
  package com.Endeca.soleng.eac.toolkit.component;
  ```

```
import java.util.logging.Logger;
import com.Endeca.soleng.eac.toolkit.exception.*;

public class MyForgeComponent extends ForgeComponent
{
  private static Logger log =
    Logger.getLogger(MyForgeComponent.class.getName());

  protected void getData() throws AppConfigurationException,
      EacCommunicationException, EacComponentControlException,
      InterruptedException
  {

    // get dimensions for processing
    getConfig();

  }
}
```

Obviously, this trivial customization is too simple to warrant the development of a new class. However, this approach can be used to override the functionality of most methods in the toolkit or to implement new methods.

In order to use the new functionality, the developer will compile the new class and ensure that it is included on the classpath when invoking scripts. The simplest way to do this is to deploy the compiled `.class` file to the `[appdir]/config/script` directory. Once on the classpath, the new component can be loaded in place of the default Forge component by making the following change to the Forge configuration in `AppConfig.xml`:

```
<forge class="com.Endeca.soleng.eac.toolkit.component.MyForgeComponent"

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

Some types of customization will require more complex configuration. Refer to Appendix A ("EAC Development Toolkit") for information about configuring custom Java classes using the Spring Framework namespace in the `AppConfig.xml` document.

Chapter 3

Application Configuration

This section provides an overview of the elements defined in the configuration document.

About application configuration

The application configuration document `[appdir]/config/script/AppConfig.xml` defines the hosts and components that make up an EAC application and the scripts that orchestrate updates by executing the defined components.

Multiple configuration documents may be used to define distinct parts of an application, to separate scripts from component provisioning, or for other purposes. The Deployment Template provides a single `AppConfig.xml` file for the deployment type you choose. However, any number of `--app-config` arguments may be specified the Controller class in the EAC development toolkit. All of the objects in the files will be read and processed and scripts can refer to components, hosts, or other scripts defined in other files.

The following sections describe each portion of the application configuration document.

Global application settings

This first section of the configuration document defines global application-level configuration, including the host and port of the EAC Central Server, the application name and whether or not SSL is to be used when communicating with the Central Server.

In addition, a default working and log directory are specified and a default `lockManager` is specified for use by other elements defined in the document. All elements inherit these settings or override them.

```
<!—
########################################################################
# Global variables
#
-->
  <app appName="MyApp" eacHost="myhost1.company.com" eacPort="8888"
      dataPrefix="MyApp" sslEnabled="false" lockManager="LockManager">
    <working-dir>C:\Endeca\MyApp</working-dir>
```

```
    <log-dir>./logs/baseline</log-dir>
  </app>
```

Hosts

All servers in a deployment are enumerated in the host definition portion of the document.

Each host must be given a unique ID. The port specified for each host is the port on which the EAC Agent is listening, which is the Endeca HTTP Service port on that server.

```
<!–
########################################################################
# Servers/hosts
#
-->
  <host id="ITLHost" hostName="myhost1.company.com" port="8888" />
  <host id="MDEXHost" hostName="myhost2.company.com" port="8888" />
```

Lock Manager

The LockManager component is used to obtain and release locks and to set and remove flags using the EAC's synchronization Web service.

A LockManager object is associated with the elements in the application to enable a centralized access point to locks, allowing multiple objects to test for the existence of locks and flags. When a script or component invocation fails, the Deployment Template attempts to release all locks acquired during the invocation for a LockManager configured to release locks on failure. Multiple LockManager components may be configured, if it is appropriate for some locks to be released on failure while others remain.

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

Fault tolerance and polling interval properties

Two sets of configurable properties set the behavior of the Deployment Template fault tolerance mechanism and the frequency of status checks for components.

Fault tolerance property

You can now configure fault tolerance (i.e., retries) for any component (such as Forge, Dgidx, and Dgraph) when invoked through the EAC. This functionality also extends to the CAS server when running a crawl with the CAS component. The name of the fault-tolerance property is `maxMissedSta¬tusQueriesAllowed`.

When components are run, the Deployment Template instructs the EAC to start a component, then polls on a regular interval to check if the component is running, stopped, or failed. If one of these status checks fails, the Deployment Template assumes the component has failed and the script ends. The `maxMissedStatusQueriesAllowed` property allows a configurable number of consecutive failures to be tolerated before the script will end.

The following is an example of a Forge component configured to tolerate a maximum of ten consecutive failures:

```
<forge id="Forge" host-id="ITLHost">
  <properties>
    <property name="numStateBackups" value="10"/>
    <property name="numLogBackups" value="10"/>

    <property name="maxMissedStatusQueriesAllowed" value="10"/>
  </properties>
  ...
</forge>
```

The default number of allowed consecutive failures is 5. Note that these status checks are consecutive, so that every time a status query returns successfully, the counter is reset to zero.

Keep in mind that you can use different fault-tolerance settings for your components. For example, you could set a value of 10 for the Forge component, a value of 8 for Dgidx, and a value of 6 for the Dgraph.

Polling interval properties

As described in the previous section, the Deployment Template polls on a regular interval to check if a started component is running, stopped, or failed. A set of four properties is available to configure each component for how frequently the Deployment Template polls for status while the component is running. Because each property has a default value, you can use only those properties that are important to you.

The polling properties are as follows:

- `minWaitSeconds` specifies the threshold (in seconds) when slow polling switches to standard (regular) polling. The default is **-1** (i.e., no threshold, so the standard polling interval is used from the start).
- `slowPollingIntervalMs` specifies the interval (in milliseconds) that status queries are sent as long as the `minWaitSeconds` time has not elapsed. The default slow polling interval is **60** seconds.
- `standardPollingIntervalMs` (specified in milliseconds) is used after the `minWaitSeconds` time has passed. If no `minWaitSeconds` setting is specified, the `standardPollingIntervalMs` setting is always used. The default standard polling interval is **1** second.
- `maxWaitSeconds` specifies the threshold (in seconds) when the Deployment Template gives up asking for status and assumes that it has failed. The default is **-1** (i.e., no threshold, so the Deployment Template will keep trying indefinitely).

Here is an example configuration for a long-running Forge component that typically takes 8 hours to complete:

```
<forge id="Forge" host-id="ITLHost">
  <properties>
    <property name="numStateBackups" value="10"/>
    <property name="numLogBackups" value="10"/>

    <property name="standardPollingIntervalMs" value="60000"/>
    <property name="slowPollingIntervalMs" value="600000"/>
    <property name="minWaitSeconds" value="28800"/>
    <property name="maxMissedStatusQueriesAllowed" value="10"/>
  </properties>
  ...
</forge>
```

The result of this configuration would be that for the first 8 hours (`minWaitSeconds`=28800), Forge's status would be checked every 10 minutes (`slowPollingIntervalMs`=600000), after which time

the status would be checked every minute (`standardPollingIntervalMs`=60000). If a status check fails, a maximum of 10 consecutive retries will be attempted, based on the `standardPolling¬IntervalMs` setting.

Keep in mind that these values can be set independently for each component.

Fault tolerance and polling interval for utilities

Fault tolerance and polling interval values can also be set for these utilities:

- copy
- shell
- archive
- rollback

You set the new values by adjusting the BeanShell script code that is used to construct and invoke the utility. You adjust the code by using these setter methods from the EAC Toolkit's `Utility` class:

- `Utility.setMinWaitSeconds()`
- `Utility.setMaxWaitSeconds()`
- `Utility.setMaxMissedStatusQueriesAllowed()`
- `Utility.setPollingIntervalMs()`
- `Utility.setSlowPollingIntervalMs()`
- `Utility.setMaxMissedStatusQueriesAllowed()`

If you do not use any of these methods, then the utility will use the default values listed in the two previous sections.

For example, here is a default utility invocation in the CAS crawl scripts:

```
// create the target dir, if it doesn't already exist
mkDirUtil = new CreateDirUtility(CAS.getAppName(),
   CAS.getEacHost(), CAS.getEacPort(), CAS.isSslEnabled());
mkDirUtil.init(Forge.getHostId(), destDir, CAS.getWorkingDir());
mkDirUtil.run();
```

You would then add these methods before calling the `run()` method, so that the code would now look like this:

```
// create the target dir, if it doesn't already exist
mkDirUtil = new CreateDirUtility(CAS.getAppName(),
   CAS.getEacHost(), CAS.getEacPort(), CAS.isSslEnabled());
mkDirUtil.init(Forge.getHostId(), destDir, CAS.getWorkingDir());
mkDirUtil.setMinWaitSeconds(30);
mkDirUtil.setMaxWaitSeconds(120);
mkDirUtil.setMaxMissedStatusQueriesAllowed(10);
mkDirUtil.setPollingIntervalMs(5000);
mkDirUtil.setSlowPollingIntervalMs(30000);
mkDirUtil.run();
```

Alternatively, if your utility was defined in your `AppConfig.xml` like this:

```
<copy id=”MyCopy” src-host-id=”ITLHost” dest-host-id=”MDEXHost” recur¬
sive=”true”>
   <src>./path/to/files</src>
   <dest>./path/to/target</dest>
</copy>
```

You would add the same type of lines as above, before calling the `run()` method; for example:

```
MyCopy.setMaxMissedStatusQueriesAllowed(10);
 MyCopy.run();
```

For more information on the `Utility` methods, see the Javadocs for the EAC Toolkit package.

Forges

One or many Forge components are defined for baseline update processing and partial update processing depending on the deployment type you choose.

If an Agraph deployment type is chosen, a Forge cluster component is defined. This object is used to apply actions to an entire cluster of Forges, rather than manually iterating over a number of Forges. In addition, the object contains logic associated with executing Forges in parallel based on Forge groups, which are described below. Multiple Forge clusters can be defined, with no restriction around which Forges belong to each cluster or how many clusters a Forge belongs to.

A Forge cluster is configured with references to all Forges that belong to that cluster. In addition, the cluster can be configured to copy data in parallel or serially. This setting applies to copies that are performed to retrieve source data and configuration to each server that hosts a Forge component. By default, the template sets this value to true.

```
<!--
########################################################################
# Forge Cluster
#
-->
<forge-cluster id="ForgeCluster" getDataInParallel="true">
  <forge ref="ForgeServer" />
  <forge ref="ForgeClient1" />
  <forge ref="ForgeClient2" />
</forge-cluster>
```

In addition to standard Forge configuration settings and process arguments, the Deployment Template uses several configurable properties and custom directories during processing:

- `numLogBackups` - Number of log directory backups to store.
- `numStateBackups` - Number of autogen state directory backups to store.
- `numPartialsBackups` - Number of cumulative partials directory backups to store. It is recommended that you increase the default value of 5. The reason is that the files in the updates directory for the Dgraph are automatically deleted after partials are applied to the Dgraph. The number you choose depends on how often you run partial updates and how many copies you want to keep.
- `incomingDataHost` - Host to which source data files are extracted.
- `incomingDataDir` - Directory to which source data files are extracted.
- `incomingDataFileName` - Filename of the source data files that are extracted.
- `configHost` - Host from which configuration files and dimensions are retrieved for Forge to process.
- `configDir` - Directory from which configuration files and dimensions are retrieved for Forge to process.
- `cumulativePartialsDir` - Directory where partial updates are accumulated between baseline updates.
- `wsTempDir` - Temp Oracle Endeca Workbench directory to which post-Forge dimensions are copied to be uploaded to the Workbench.
- `skipTestingForFilesDuringCleanup` - Used for directory-cleaning operations. If set to "true", will skip the directory-contents test and instead proceed directly to cleaning the directory. The default behavior is to test the directory contents and skip cleanup if the directory is not empty.
- The properties documented in the "Fault tolerance and polling interval properties" topic.

This excerpt combines properties from both the baseline and partial update Forge to demonstrate the use of all of these configuration settings.

```
<properties>
  <property name="forgeGroup" value="A" />
  <property name="incomingDataHost">ITLHost</property>
  <property name="incomingDataFileName">project_name-part0-*</property>
  <property name="configHost">ITLHost</property>
  <property name="numStateBackups" value="10" />
  <property name="numLogBackups" value="10" />
  <property name="numPartialsBackups" value="5" />
  <property name="skipTestingForFilesDuringCleanup" value="true" />
</properties>
<directories>
  <directory name="incomingDataDir">./data/partials/incoming</directory>
  <directory name="configDir">./config/pipeline</directory>
  <directory name="cumulativePartialsDir">
    ./data/partials/cumulative_partials
  </directory>
  <directory name="wsTempDir">./data/web_studio/temp</directory>
</directories>
```

In addition to standard Forge configuration and process arguments, Forge processes add a custom property used to define which Forge processes run in parallel with each other when they belong to a Forge cluster.

`forgeGroup` - Indicates the Forge's membership in a Forge group. When the run method on a Forge cluster is executed, Forge processes within the same Forge group are run in parallel. Forge group values are arbitrary strings. The Forge cluster iterates through the groups in alphabetical order, though non-standard characters may result in groups being updated in an unexpected order.

Dgidxs

One or many Dgidx components are defined depending on the deployment type you choose.

If an Agraph deployment type is chosen, an indexing cluster component is defined. This object is used to apply actions to an entire cluster of Dgidxs, rather than manually iterating over a number of Dgidxs. In addition, the object contains logic associated with executing Dgidxs in parallel based on Dgidx groups, which are described below. Multiple indexing clusters can be defined, with no restriction around which Dgidx belongs to each cluster or how many clusters a Dgidx belongs to.

An indexing cluster is configured with references to all Dgidxs that belong to that cluster. In addition, the cluster can be configured to copy data in parallel or serially. This setting applies to copies that are performed to retrieve source data and configuration to each server that hosts a Dgidx component. By default, the template sets this value to true.

```
<!--
########################################################################
# Indexing Cluster
#
-->
<indexing-cluster id="IndexingCluster" getDataInParallel="true">
  <agidx ref="Agidx1" />
  <dgidx ref="Dgidx1" />
  <dgidx ref="Dgidx2" />
</indexing-cluster>
```

In addition to standard Dgidx configuration settings and process arguments, the Deployment Template uses several configurable properties and custom directories during processing:

- `numLogBackups` - Number of log directory backups to store.
- `numIndexbackups` - Number of index backups to store.
- `incomingDataHost` - Host to which source data files are extracted.
- `incomingDataDir` - Directory to which source data files are extracted.
- `incomingDataFileName` - Filename of the source data files that are extracted.
- `configHost` - Host from which configuration files and dimensions are retrieved for Dgidx to process.
- `configDir` - Directory from which configuration files and dimensions are retrieved for Dgidx to process.
- `configFileName` - Filename of the configuration files and dimensions that are retrieved for Dgidx to process.
- `skipTestingForFilesDuringCleanup` - Used for directory-cleaning operations. If set to "true", will skip the directory-contents test and instead proceed directly to cleaning the directory. The default behavior is to test the directory contents and skip cleanup if the directory is not empty.
- The properties documented in the "Fault tolerance and polling interval properties" topic.

In addition to standard Dgidx configuration and process arguments, Dgidx processes add a custom property used to define which Dgidx processes run in parallel with each other when they belong to an indexing cluster.

`dgidxGroup` - Indicates the Dgidx's membership in a Dgidx group. When the run method on an indexing cluster is executed, Dgidx processes within the same Dgidx group are run in parallel. Dgidx group values are arbitrary strings. The indexing cluster iterates through the groups in alphabetical order, though non-standard characters may result in groups being updated in an unexpected order.

Agidxs

An Agidx component is defined. An indexing cluster is used to simplify common operations across multiple Agidxs. The indexing cluster is described in the previous Dgidx section.

In addition to standard Agidx configuration settings and process arguments, the Deployment Template uses several configurable properties and custom directories during processing.

- `numLogBackups` - Number of log directory backups to store.
- `numIndexbackups` - Number of index backups to store.
- `incomingPreviousOutputHost` - Host to which source data files are extracted.
- `incomingPreviousOutputDir` - Directory to which source data files are extracted.
- `skipTestingForFilesDuringCleanup` - Used for directory-cleaning operations. If set to "true", will skip the directory-contents test and instead proceed directly to cleaning the directory. The default behavior is to test the directory contents and skip cleanup if the directory is not empty.
- The properties documented in the "Fault tolerance and polling interval properties" topic.

In addition to standard Agidx configuration and process arguments, Agidx processes add a custom property used to define which Agidx processes run in parallel with each other when they belong to an indexing cluster.

`agidxGroup` - Indicates the Agidx's membership in an Agidx group. When the run method on an indexing cluster is executed, Agidx processes within the same Agidx group are run in parallel. Agidx group values are arbitrary strings. The indexing cluster iterates through the groups in alphabetical order, though non-standard characters may result in groups being updated in an unexpected order.

Dgraphs

If a Dgraph deployment type is chosen, a Dgraph cluster component is defined.

This object is used to apply actions to an entire cluster of Dgraphs, rather than manually iterating over a number of Dgraphs. In addition, the object contains logic associated with Dgraph restart strategies, which are described below. Multiple Dgraph clusters can be defined, with no restriction around which Dgraphs belong to each cluster or how many clusters a Dgraph belongs to.

A Dgraph cluster is configured (via the `dgraph-cluster` element) with references to all Dgraphs that belong to that cluster. In addition, the cluster can be configured to copy data in parallel or serially. This setting applies to copies that are performed to distribute a new index, partial updates or configuration updates to each server that hosts a Dgraph. By default, the template sets this value to true.

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

Two Dgraphs are defined by the template by default.

Global Dgraph settings

In order to avoid defining shared configuration for multiple Dgraphs in each Dgraph's XML configuration, the document provides the `dgraph-defaults` element, where shared settings can be configured and inherited (or overridden) by each Dgraph defined in the document. This defaults object specifies a number of custom configuration properties that are used by the update scripts to define operational functionality.

- `numLogBackups` - Number of log directory backups to store.
- `shutdownTimeout` - Number of seconds to wait for a component to stop (after receiving a stop command).
- `numIdleSecondsAfterStop` - Number of seconds to pause/sleep after a component is stopped. Typically, this will be used to ensure that log file locks are release by the component before proceeding.
- `srcIndexDir` - Location from which a new index will be copied to a local directory on the Dgraph's host.
- `srcIndexHostId` - Host from which a new index will be copied to a local directory on the Dgraph's host.
- `localIndexDir` - Local directory to which a single copy of a new index is copied from the source index directory on the source index host.
- `srcPartialsDir` - Location from which a new partial update will be copied to a local directory on the Dgraph's host.
- `srcCumulativePartialsDir` - Location from which all partial updates accumulated since the last baseline update will be copied to a local directory on the Dgraph's host.
- `srcPartialsHostId` - Host from which partial updates will be copied to a local directory on the Dgraph's host.
- `localCumulativePartialsDir` - Local directory to which partial updates are copied from the source (cumulative) partials directory on the source partials host.

- `srcDgraphConfigDir` - Location from which Dgraph configuration files will be copied to a local directory on the Dgraph's host.
- `srcDgraphConfigHostId` - Host from which Dgraph configuration files will be copied to a local directory on the Dgraph's host.
- `localDgraphConfigDir` - Local directory to which Dgraph configuration files are copied from the source Dgraph config directory on the source Dgraph config host.
- `srcXQueryHostId` - Host from which XQuery modules will be copied to a local directory on the Dgraph's host.
- `srcXQueryDir` - Location from which XQuery modules will be copied to a local directory on the Dgraph's host.
- `localXQueryDir` - Local directory to which XQuery modules are copied from the source Dgraph XQuery directory on the source Dgraph XQuery modules host.
- `skipTestingForFilesDuringCleanup` - Used for directory-cleaning operations. If set to "true", will skip the directory-contents test and instead proceed directly to cleaning the directory. The default behavior is to test the directory contents and skip cleanup if the directory is not empty.
- The properties documented in the "Fault tolerance and polling interval properties" topic.

```
<!--
########################################################################
# Global Dgraph settings, inherited by all dgraphs
#
-->
<dgraph-defaults>
  <properties>
    <property name="srcIndexDir" value="./data/dgidx_output" />
    <property name="srcIndexHostId" value="ITLHost" />
    <property name="srcPartialsDir" value="./data/partials/forge_output"
/>
    <property name="srcPartialsHostId" value="ITLHost" />
    <property name="srcCumulativePartialsDir" value="./data/partials/cumu¬
lative_partials" />
    <property name="srcCumulativePartialsHostId" value="ITLHost" />
   <property name="srcDgraphConfigDir" value="./data/web_studio/dgraph_con¬
fig" />
    <property name="srcDgraphConfigHostId" value="ITLHost" />
    <property name="srcXQueryHostId" value="ITLHost" />
    <property name="srcXQueryDir" value="./config/lib/xquery" />
    <property name="numLogBackups" value="10" />
    <property name="shutdownTimeout" value="30" />
    <property name="numIdleSecondsAfterStop" value="0" />
  </properties>
  <directories>
    <directory name="localIndexDir">./data/dgraphs/local_dgraph_input</di¬
rectory>
   <directory name="localCumulativePartialsDir">./data/dgraphs/local_cumu¬
lative_partials</directory>
   <directory name="localDgraphConfigDir">./data/dgraphs/local_dgraph_con¬
fig</directory>
   <directory name="localXQueryDir">./data/dgraphs/local_xquery</directory>

  </directories>
  <args>
    <arg>--threads</arg>
    <arg>2</arg>
    <arg>--spl</arg>
    <arg>--dym</arg>
    <arg>--xquery_path</arg>
    <arg>./data/dgraphs/local_xquery</arg>
```

```
  </args>
  <startup-timeout>120</startup-timeout>
</dgraph-defaults>
```

Each Dgraph defined in the document (via the `dgraph` element) inherits from the settings defined in the `dgraph-defaults` element, and also specifies settings that are unique to the Dgraph.

Note: As of version 3.1 of the Deployment Template, the `numCacheWarmupSeconds` and `offlineUpdate` properties are ignored (and warning messages generated) because they are not supported in the 6.1.x MDEX Engine.

Restart and update custom properties

In addition to standard Dgraph configuration and process arguments, the `dgraph` element adds two custom properties that define restart and update strategies:

- `restartGroup`
- `updateGroup`

The `restartGroup` property indicates the Dgraph's membership in a restart group. When applying a new index or configuration updates to a cluster of Dgraphs (or when updating a cluster of Dgraphs with a provisioning change such as a new or modified process argument), the Dgraph cluster object applies changes simultaneously to all Dgraphs in a restart group.

Similarly, the `updateGroup` property indicates the Dgraph's membership in an update group. When applying partial updates, the Dgraph cluster object applies changes simultaneously to all Dgraphs in an update group.

This means that a few common restart strategies can be applied as follows:

- To restart/update all Dgraphs at once: specify the same restartGroup/updateGroup value for each Dgraph.
- To restart/update Dgraphs one at a time: specify a unique restartGroup/updateGroup value for each Dgraph, or omit one or both of the custom properties on all Dgraphs (causing the template to assign a unique group to each Dgraph).
- To restart/update Dgraphs on each server simultaneously: specify the same restartGroup/updateGroup value for each Dgraph on a physical server.
- To restart Dgraphs one at a time but apply partial updates to all Dgraphs at once: specify a unique restartGroup value for each Dgraph and specify the same updateGroup value for each Dgraph.

```
<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>
```

Restart and update group values are arbitrary strings. The DgraphCluster will iterate through the groups in alphabetical order, though non-standard characters may result in groups being updated in an unexpected order.

Running scripts

Dgraph components can specify the name of a script to invoke prior to shutdown and the name of a script to invoke after the component is started. These optional attributes must specify the ID of a Script defined in the XML file(s). These BeanShell scripts are executed just before the Dgraph is stopped or just after it is started. The scripts behave identically to other BeanShell scripts, except that they have an additional variable, invokingObject, which holds a reference to the Dgraph that invoked the script. This functionality is typically used to implement calls to a load balancer, adding or removing a Dgraph from the cluster as it is updated.

The following example shows two dummy scripts (which just log a message, but could be extended to call out to a load balancer) provisioned to run pre-shutdown and post-startup for Dgraph1.

```
<dgraph id="Dgraph1" host-id="MDEXHost" port="15000"
    pre-shutdown-script="DgraphPreShutdownScript"
    post-startup-script="DgraphPostStartupScript">
  <properties>
    <property name="restartGroup" value="A" />
  </properties>
  <log-dir>./logs/dgraphs/Dgraph1</log-dir>
  <input-dir>./data/dgraphs/Dgraph1/dgraph_input</input-dir>
  <update-dir>./data/dgraphs/Dgraph1/dgraph_input/updates</update-dir>
</dgraph>

<script id="DgraphPreShutdownScript">
  <bean-shell-script>
    <![CDATA[
    id = invokingObject.getElementId();
    hostname = invokingObject.getHost().getHostName();
    port = invokingObject.getPort();
    log.info("Removing dgraph with id " + id + " (host: " + hostname +
      ", port: " + port + ") from load balancer cluster.");
    ]]>
  </bean-shell-script>
</script>

<script id="DgraphPostStartupScript">
  <bean-shell-script>
    <![CDATA[
    id = invokingObject.getElementId();
    hostname = invokingObject.getHost().getHostName();
    port = invokingObject.getPort();
    log.info("Adding dgraph with id " + id + " (host: " + hostname +
      ", port: " + port + ") to load balancer cluster.");
    ]]>
  </bean-shell-script>
</script>
```

The following log excerpt shows these scripts running when a new index is being applied to the dgraph:

```
[03.10.08 10:03:28] INFO: Applying index to dgraphs in restart group 'A'.
[03.10.08 10:03:28] INFO: [MDEXHost] Starting shell utility 'mkpath_dgraph-
input-new'.
[03.10.08 10:03:30] INFO: [MDEXHost] Starting copy utility 'copy_in¬
dex_to_temp_new_dgraph_input_dir_for_Dgraph1'.
[03.10.08 10:03:35] INFO: Removing dgraph with id Dgraph1 (host: mdex1.my¬
company.com, port: 15000) from load balancer cluster.
[03.10.08 10:03:35] INFO: Stopping component 'Dgraph1'.
[03.10.08 10:03:37] INFO: [MDEXHost] Starting shell utility 'move_dgraph-
input_to_dgraph-input-old'.
[03.10.08 10:03:39] INFO: [MDEXHost] Starting shell utility 'move_dgraph-
```

```
input-new_to_dgraph-input'.
[03.10.08 10:03:40] INFO: [MDEXHost] Starting backup utility 'back¬
up_log_dir_for_component_Dgraph1'.
[03.10.08 10:03:42] INFO: [MDEXHost] Starting component 'Dgraph1'.
[03.10.08 10:03:45] INFO: Adding dgraph with id Dgraph1 (host: mdex1.mycompa¬
ny.com, port: 15000) to load balancer cluster.
[03.10.08 10:03:45] INFO: [MDEXHost] Starting shell utility 'rmdir_dgraph-
input-old'.
```

Note that the `dgraph-default` element can also specify the use of pre-shutdown and post-startup scripts as attributes, allowing all Dgraphs in an application to execute the same scripts. For example:

```
<dgraph-defaults pre-shutdown-script="DgraphPreShutdownScript"
    post-startup-script="DgraphPostStartupScript">

  ...

</dgraph-defaults>
```

Deploying XQuery modules

The Deployment Template supports the distribution of XQuery modules to each Dgraph in the group. The `[appdir]config/lib/xquery` directory is provided for users to store their XQuery modules. In addition, a LoadXQueryModules script (in the `AppConfig.xml` file) distributes the XQuery modules to Dgraph servers and instructs the Dgraphs to load the modules.

The procedure to deploy the XQuery modules is:

1. Make certain that the `dgraph-defaults` section of the `AppConfig.xml` file has the XQuery properties set. These global Dgraph setting properties are `srcXQueryHostId`, `srcXQueryDir`, and `localXQueryDir`.
2. Make certain that the Dgraph `--xquery_path` flag is specified as an argument in the `dgraph-defaults` section.
3. Place all the XQuery code in the `[appdir]/config/lib/xquery` and `[appdir]/config/lib/xquery/lib` directories.
4. Execute the `runcommand` script with the `LoadXQueryModules` argument, as in this Windows example:

   ```
   C:\Endeca\Apps\control>runcommand LoadXQueryModules
   ```

The XQuery modules are distributed to the Dgraphs in the deployment and they are instructed to reload/compile the modules.

Specifying arguments for the Dgraphs

Both the `dgraph` and `dgraph-defaults` elements allow you to use the `args` sub-element to pass command-line flags to the Dgraphs. However, if you use an `args` section in both the `dgraph` and `dgraph-defaults` configurations, the results are not cumulative.

Instead, the `args` section for an individual Dgraph completely overrides the `dgraph-defaults` definition (i.e., it does not inherit the parameters that are specified in the `dgraph-defaults` section and then add the ones that are unique for that Dgraph).

Enabling SSL for the Dgraph

You can configure the Dgraph for SSL by using the following elements to define the certificates to use for SSL:

- `cert-file` specifies the path of the `eneCert.pem` certificate file that is used by the Dgraph to present to any client. This is also the certificate that the Application Controller Agent should present to the Dgraph when trying to talk to the Dgraph.
- `ca-file` specifies the path of the `eneCA.pem` Certificate Authority file that the Dgraph uses to authenticate communications with other Oracle Endeca components.
- `cipher` specifies an optional cipher string (such as RC4-SHA) that specifies the minimum cryptographic algorithm that the Dgraph uses during the SSL negotiation. If you omit this setting, the SSL software tries an internal list of ciphers, beginning with AES256-SHA. See the *Oracle Endeca Platform Services Security Guide* for more information.

All three elements are first-level children of the `<dgraph-defaults>` element.

The following example shows the three SSL elements being used within the `dgraph-default` element:

```
<dgraph-defaults>
...
   <cert-file>
      C:\Endeca\PlatformServices\workspace\etc\eneCert.pem
   </cert-file>
   <ca-file>
      C:\Endeca\PlatformServices\workspace\etc\eneCA.pem
   </ca-file>
   <cipher>AES128-SHA</cipher>
</dgraph-defaults>
```

Agraphs

If an Agraph deployment type is chosen, an Agraph cluster component is defined.

This object is used to apply actions to an entire cluster of Agraphs and their associated Dgraphs, rather than manually iterating over a number of graphs. In addition, the object contains logic associated with Agraph restart strategies, which are described below. Multiple Agraph clusters can be defined, with no restriction around which graphs belong to each cluster or how many clusters a graph belongs to.

An Agraph cluster is configured with references to all Agraphs and Dgraphs that belong to that cluster. In addition, the cluster can be configured to copy data in parallel or serially. This setting applies to copies that are performed to distribute a new index, partial updates or configuration updates to each server that hosts a graph. By default, the template sets this value to true.

```
<!--
########################################################################
# Agraph Cluster
#
-->
<agraph-cluster id="AgraphCluster" getDataInParallel="true">
  <agraph ref="Agraph1" />
  <agraph ref="Agraph2" />
  <dgraph ref="Dgraph1" />
  <dgraph ref="Dgraph2" />
</agraph-cluster>
```

In an Agraph deployment, two Agraphs and two Dgraphs are defined by the template by default. In order to avoid defining shared configuration for multiple Agraphs in each Agraph's XML configuration, the document provides an agraph-defaults element, where shared settings can be configured and inherited (or overridden) by each Agraph defined in the document. This defaults object specifies a number of custom configuration properties that are used by the update scripts to define operational functionality.

- `numLogBackups` - Number of log directory backups to store.
- `shutdownTimeout` - Number of seconds to wait for a component to stop (after receiving a stop command).
- `numIdleSecondsAfterStop` - Number of seconds to pause/sleep after a component is stopped. Typically, this will be used to ensure that log file locks are release by the component before proceeding.
- `srcIndexDir` - Location from which a new index will be copied to a local directory on the Dgraph's host.
- `srcIndexHostId` - Host from which a new index will be copied to a local directory on the Dgraph's host.
- `localIndexDir` - Local directory to which a single copy of a new index is copied from the source index directory on the source index host.
- `skipTestingForFilesDuringCleanup` - Used for directory-cleaning operations. If set to "true", will skip the directory-contents test and instead proceed directly to cleaning the directory. The default behavior is to test the directory contents and skip cleanup if the directory is not empty.
- The properties documented in the "Fault tolerance and polling interval properties" topic.

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

    # Global Agraph settings, inherited by all agraphs
    #
  -->
  <agraph-defaults>
    <properties>
      <property name="srcIndexDir" value="./data/agidx_output" />
      <property name="srcIndexHostId" value="ITLHost" />
      <property name="numLogBackups" value="10" />
    </properties>
    <directories>
     <directory name="localIndexDir">./data/agraphs/local_agraph_input</di¬
rectory>
    </directories>
    <args>
      <arg>--no-partial</arg>
    </args>
    <startup-timeout>120</startup-timeout>
  </agraph-defaults>
```

Each Agraph defined in the document inherits from the settings defined in the agraph-defaults element, and also specifies settings that are unique to the Agraph. In addition to standard Agraph configuration and process arguments, Agraphs add a custom property used to define a restart strategy.

`restartGroup` - Indicates the Agraph's membership in a restart group. When applying a new index or configuration updates to a cluster of graphs (or when updating a cluster of graphs with a provisioning change such as a new or modified process argument), the Agraph cluster object applies changes simultaneously to all graphs in a restart group. This means that a few common restart strategies can be applied as follows:

1. Restart/update all Agraphs at once: specify the same `restartGroup` value for each Agraph and associated Dgraphs.
2. Restart/update Agraphs one at a time: specify a unique `restartGroup` value for each Agraph and associated Dgraphs.

3. Restart/update Agraphs on each server simultaneously: specify the same `restartGroup` value for each Agraph and associated Dgraphs on a physical server.

```
<agraph id="Agraph1" host-id="MDEXHost" port="14000">
  <properties>
    <property name="restartGroup" value="A" />
  </properties>
  <dgraph-children>
    <dgraph-child>Dgraph1</dgraph-child>
    <dgraph-child>Dgraph2</dgraph-child>
  </dgraph-children>
  <log-dir>./logs/agraphs/Agraph1</log-dir>
  <input-dir>./data/agraphs/Agraph1/agraph_input</input-dir>
</agraph>
```

Restart group values are arbitrary strings. The Agraph cluster iterates through the groups in alphabetical order, though non-standard characters may result in groups being updated in an unexpected order.

The Agraph component (and agraph-defaults) can specify pre-shutdown and post-startup scripts in the same way as the Dgraph component. Refer to the previous section for details.

Enabling SSL for the Agraph

As with the Dgraph, three SSL elements can be included in the Agraph provisioning definition to specify the certificates to use. The `cert-file` element (for the `eneCert.pem` certificate), the `ca-file` element (for the `eneCA.pem` Certificate Authority file), and the optional `cipher` element can be used. as shown in this example:

```
<agraph id="Agraph1" host-id="MDEXHost" port="14000">
...
   <cert-file>
      C:\Endeca\PlatformServices\workspace\etc\eneCert.pem
   </cert-file>
   <ca-file>
      C:\Endeca\PlatformServices\workspace\etc\eneCA.pem
   </ca-file>
   <cipher>AES128-SHA</cipher>
</agraph>
```

Related Links

Agraph notes on page 48
There are several important things to note about Agraph deployments.

Log server

A LogServer component is defined.

In addition to standard LogServer configuration settings and process arguments, the Deployment Template uses a configurable property for log archiving.

- `numLogBackups` - Number of log directory backups to store.
- `shutdownTimeout` - Number of seconds to wait for a component to stop (after receiving a stop command).
- `numIdleSecondsAfterStop` - Number of seconds to pause/sleep after a component is stopped. Typically, this will be used to ensure that log file locks are release by the component before proceeding.
- `targetReportGenDir` - Directory to which logs will be copied for report generation.
- `targetReportGenHostId` - Host to which logs will be coped for report generation.

- `skipTestingForFilesDuringCleanup` - Used for directory-cleaning operations. If set to "true", will skip the directory-contents test and instead proceed directly to cleaning the directory. The default behavior is to test the directory contents and skip cleanup if the directory is not empty.
- The properties documented in the "Fault tolerance and polling interval properties" topic.

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

Report Generators

Four report generator components are defined.

In addition to standard Report Generator configuration settings and process arguments, the Deployment Template uses a configurable property for log archiving, as well as these configurable properties:

- `skipTestingForFilesDuringCleanup` - Used for directory-cleaning operations. If set to "true", will skip the directory-contents test and instead proceed directly to cleaning the directory. The default behavior is to test the directory contents and skip cleanup if the directory is not empty.
- The properties documented in the "Fault tolerance and polling interval properties" topic.

The configuration file includes the name of an output file for each report generator, which defaults to `report.html` or `report.xml`. This file name is never used when the report generation scripts in the `AppConfig.xml` file are used. During execution, the script re-provisions the report generator to output a file named with a date stamp. This means that the provisioning in the file will always be "out of synch" with the provisioning in the EAC. This will result in the Report Generator's definition changing repeatedly as scripts are executed.

```
<report-generator id="WeeklyReportGenerator" host-id="ITLHost">
    <log-dir>./logs/report_generators/WeeklyReportGenerator</log-dir>
    <input-dir>./reports/input</input-dir>
    <output-file>./reports/weekly/report.xml</output-file>
    <stylesheet-file>
      ./config/report_templates/tools_report_stylesheet.xsl
    </stylesheet-file>
    <settings-file>
      ./config/report_templates/report_settings.xml
    </settings-file>
    <time-range>LastWeek</time-range>
    <time-series>Daily</time-series>
    <charts-enabled>true</charts-enabled>
  </report-generator>
```

Configuration Manager

The Configuration Manager component is a custom component that does not correlate to an Oracle Endeca process.

Instead, this object implements logic used to manage configuration files. Specifically, the current implementation supports retrieving and merging configuration from Developer Studio with files maintained in Oracle Endeca Workbench.

The following configuration properties and custom directories are used to implement the logic of the Config Manager component.

- `webStudioEnabled` - "true" or "false," indicating whether integration with Oracle Endeca Workbench is enabled.
- `webStudioHost` - Hostname of the server on which Oracle Endeca Workbench is running.
- `webStudioPort` - Port on which Oracle Endeca Workbench listens. This is the port of the Endeca Tools Service on the Oracle Endeca Workbench host.
- `webStudioMaintainedFile*` - Specifies the name of a file that will be maintained in Oracle Endeca Workbench. The ConfigManager respects all properties prefixed with "webStudioMaintainedFile" but requires that all properties have unique names. When configuring files, each should be given a unique suffix. Note that the names of files specified may use wildcards (e.g. `<property name="webStudioMaintainedFile1" val¬ ue="merch_rule_group_*.xml" />`).
- `devStudioConfigDir` - Directory from which Developer Studio configuration files are retrieved.
- `webStudioConfigDir` - Directory to which Workbench configuration files are downloaded.
- `webStudioDgraphConfigDir` - Directory from which Developer Studio configuration files are retrieved.
- `mergedConfigDir` - Directory to which merged configuration is copied.
- `webStudioTempDir` - Temporary directory used for Workbench interaction. Post-Forge dimensions are uploaded from this directory to the Workbench.
- `skipTestingForFilesDuringCleanup` - Used for directory-cleaning operations. If set to "true", will skip the directory-contents test and instead proceed directly to cleaning the directory. The default behavior is to test the directory contents and skip cleanup if the directory is not empty.
- The properties documented in the "Fault tolerance and polling interval properties" topic.

```
<!--
########################################################################
# Config Manager. Manages Dev Studio and Web Studio config sources.
#
-->
<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="ws.mycompany.com" />
    <property name="webStudioPort" value="8006" />
    <property name="webStudioMaintainedFile1"
      value="thesaurus.xml" />
    <property name="webStudioMaintainedFile2"
      value="merch_rule_group_default.xml" />
    <property name="webStudioMaintainedFile3"
      value="merch_rule_group_default_redirects.xml" />
  </properties>
  <directories>
    <directory name="devStudioConfigDir">
      ./config/pipeline
    </directory>
    <directory name="webStudioConfigDir">
      ./data/web_studio/config
    </directory>
    <directory name="webStudioDgraphConfigDir">
      ./data/web_studio/dgraph_config
```

```
    </directory>
    <directory name="mergedConfigDir">
      ./data/complete_index_config
    </directory>
    <directory name="webStudioTempDir">
      ./data/web_studio/temp
    </directory>
  </directories>
</custom-component>
```

Agraph notes

There are several important things to note about Agraph deployments.

Split pipeline

In a Parallel Forge Agraph deployment it is necessary to split the data prior to executing the parallel Forge. The Deployment Template uses Forge to accomplish this. It first runs a single Forge on all the data with a rollover element specified in the pipeline which tells Forge to split the data into a number of pieces. Each one of those individual pieces of data is then used as incoming data for its corresponding Parallel Forge client. In some cases some other means of splitting the data may be used or the data may come from its source pre-split. In these cases the split Forge step can be removed and replaced with the alternate splitting method.

Multiple Agidxs

The default Agraph deployment has a single Agidx specified. This is because all of the Dgidx processes are set to run on the same machine. In a more complex deployment you may have Dgidx processes running on a number of machines. This would require you to define an Agidx for each machine that has one or more Dgidx processes running to it. Each Agidx process would run in serial and would specify the previous Agidx processes' output as part of its input.

Agraph and Dgraph restart groups

Each Agraph and Dgraph can specify its own restart group. In most cases, an Agraph and all of its children Dgraphs should specify the same restart group. There is currently no technical restriction requiring this but it is recommended to ensure standard expected graph restart behavior.

Related Links

Agraphs on page 43

If an Agraph deployment type is chosen, an Agraph cluster component is defined.

Configuration overrides

The Deployment Template allows the use of one or more configuration override files.

These files can be used to override or substitute values into the configuration documents. For example, developers may want to separate the specification of environment-specific configuration (e.g. hostnames, ports, etc.) from the application configuration and scripts. This may be useful for making configuration documents portable across environments and for dividing ownership of configuration elements between system administrators and application developers.

Override files are specified by using the `--config-override` flag to the EAC development toolkit's controller. For example, the `runcommand` script in the template includes an environment.properties file by default, though this file only contains examples of overrides and does not specify any active overrides.

Two types of properties can be specified in an override file:

1. `[object].[field] = [value]` - This style of override specifies the name of an object and field and sets the value for that field, overriding any value specified for that field in the XML configuration document or documents. For example:

   ```
   Dgraph1.port = 16000
   Dgraph1.properties['restartGroup'] = B
   ITLHost.hostName = itl.mycompany.com
   ```

2. `[token] = [value]` - This style of override specifies the name of a token defined in the XML config file and substitutes the specified value for that token. For example, if the `AppConfig.xml` defines the following host:

   ```
   <host id="ITLHost" hostName="${itl.host}" port="${itl.port}" />
   ```

   The override can specify the values to substitute for these tokens:

   ```
   itl.host = it.mycompany.com
   itl.port = 8888
   ```

It is important to note that both styles of substitution are attempted for every value defined in the override file. When a token fails to match, a low-severity warning is logged and ignored. This is required because most tokens will only match one of the two styles of substitution. It may be important to avoid using token names that coincide with object names. For example, defining the token `${Forge.tem¬pDir}` will cause the corresponding value to substitute for both the token as well as the `tempDir` field of the Forge component.

Chapter 4

Scripts

This section describes the scripts included with the Deployment Template and provides information about running and configuring them.

Provisioning scripts

The EAC allows scripts to be provisioned and invoked via Web service calls. A script is provisioned by specifying a working directory, a log directory into which output from the script is recorded, and a command to execute the script.

The `AppConfig.xml` document allows defined scripts to be provisioned by specifying the command used to invoke the script from the command line. When the provisioning configuration information is included, the script is provisioned and becomes available for invocation via Web service calls or from the EAC Admin console in Oracle Endeca Workbench. When excluded, the script is not provisioned.

```
  <script id="BaselineUpdate">
    <log-dir>./logs/provisioned_scripts</log-dir>
    <provisioned-script-command>
        ./control/baseline_update.bat
    </provisioned-script-command>
    <bean-shell-script>
      <![CDATA[
...
      ]]>
    </bean-shell-script>
  </script>
```

The command line used to invoke scripts can always be specified in this form, relative to the default Deployment Template working directory:

```
./control/runcommand.[sh|bat] [script id]
```

Dgraph baseline update script

The baseline update script defined in the `AppConfig.xml` document for a Dgraph deployment is included in this section, with numbered steps indicating the actions performed at each point in the script.

```
<script id="BaselineUpdate">
    <![CDATA[
  log.info("Starting baseline update script.");
```

1. Obtain lock. The baseline update attempts to set an `"update_lock"` flag in the EAC to serve as a lock or mutex. If the flag is already set, this step fails, ensuring that the update cannot be started more than once simultaneously, as this would interfere with data processing. The flag is removed in the case of an error or when the script completes successfully.

   ```
   // obtain lock
   if (LockManager.acquireLock("update_lock")) {
   ```

2. Validate data readiness. Check that a flag called "`baseline_data_ready`" has been set in the EAC. This flag is set as part of the data extraction process to indicate that files are ready to be processed (or, in the case of an application that uses direct database access, the flag indicates that a database staging table has been loaded and is ready for processing). This flag is removed as soon as the script copies the data out of the `data/incoming` directory, indicating that new data may be extracted.

   ```
   // test if data is ready for processing
   if (Forge.isDataReady()) {
   ```

3. If Workbench integration is enabled, download and merge Workbench configuration. The `Config¬Manager` copies all Developer Studio config files to the `complete_index_config` directory. Then, all Workbench-maintained configuration files are downloaded. Any files that are configured in the `ConfigManager` component to be maintained by the Oracle Endeca Workbench are copied to the `complete_index_config` directory, overwriting the Developer Studio copy of the same file, if one exists. The final result is a complete set of configuration files for Forge to use. If Workbench integration is not enabled, the `ConfigManager` copies all Developer Studio config files to the `complete_index_config` directory.

   ```
   if (ConfigManager.isWebStudioEnabled()) {
      // get Web Studio config, merge with Dev Studio config
      ConfigManager.downloadWsConfig();
      ConfigManager.fetchMergedConfig();
   } else {
      ConfigManager.fetchDsConfig();
   }
   ```

4. Clean processing directories. Files from the previous update are removed from the `data/processing`, `data/forge_output`, `data/temp`, `data/dgidx_output` and `data/partials/cumulative_partials` directories.

   ```
   // clean directories
   Forge.cleanDirs();
   PartialForge.cleanCumulativePartials();
   Dgidx.cleanDirs();
   ```

5. Copy data to processing directory. Extracted data in `data/incoming` is copied to `data/processing`.

   ```
   // fetch extracted data files to forge input
   Forge.getIncomingData();
   ```

6. Release Lock. The "baseline_data_ready" flag is removed from the EAC, indicating that the incoming data has been retrieved for baseline processing.

```
    LockManager.releaseLock("baseline_data_ready");
```

7. Copy config to processing directory. Configuration files are copied from data/complete_index_config to data/processing.

```
    // fetch config files to forge input
    Forge.getConfig();
```

8. Archive Forge logs. The logs/forges/Forge directory is archived, to create a fresh logging directory for the Forge process and to save the previous Forge run's logs.

```
    // archive logs
    Forge.archiveLogDir();
```

9. Forge. The Forge process executes.

```
    Forge.run();
```

10. Archive Dgidx logs. The logs/dgidxs/Dgidx directory is archived, to create a fresh logging directory for the Dgidx process and to save the previous Dgidx run's logs.

```
    // archive logs
    Dgidx.archiveLogDir();
```

11. Dgidx. The Dgidx process executes.

```
    Dgidx.run();
```

12. Distribute index to each server. A single copy of the new index is distributed to each server that hosts a Dgraph. If multiple Dgraphs are located on the same server but specify different srcIn¬ dexDir attributes, multiple copies of the index are delivered to that server.
13. Update MDEX Engines. The Dgraphs are updated. Engines are updated according to the restartGroup property specified for each Dgraph. The update process for each Dgraph is as follows:

    a. Create dgraph_input_new directory.
    b. Create a local copy of the new index in dgraph_input_new.
    c. Stop the Dgraph.
    d. Archive Dgraph logs (e.g. logs/dgraphs/Dgraph1) directory.
    e. Rename dgraph_input to dgraph_input_old.
    f. Rename dgraph_input_new to dgraph_input.
    g. Start the Dgraph.
    h. Remove dgraph_input_old.

    This somewhat complex update functionality is implemented to minimize the amount of time that a Dgraph is stopped. This restart approach ensures that the Dgraph is stopped just long enough to rename two directories.

```
    // distributed index, update Dgraphs
    DistributeIndexAndApply.run();
```

```
<script id="DistributeIndexAndApply">
    <bean-shell-script>
      <![CDATA[
    DgraphCluster.cleanDirs();
    DgraphCluster.copyIndexToDgraphServers();
    DgraphCluster.applyIndex();
      ]]>
```

```
    </bean-shell-script>
  </script>
```

14. If Workbench integration is enabled, upload post-Forge dimensions to Oracle Endeca Workbench. The latest dimension values generated by the Forge process are uploaded to Oracle Endeca Workbench, to ensure that any new dimension values (including values for autogen dimensions and external dimensions) are available to Oracle Endeca Workbench for use in, for example, dynamic business rule triggers.

Note: This action does not add new dimensions or remove existing dimensions. These changes can be made by invoking the `update_web_studio_config.[bat|sh]` script.

```
    // if Workbench is integrated, update Workbench with latest
    // dimension values
    if (ConfigManager.isWebStudioEnabled()) {
       ConfigManager.cleanDirs();
       Forge.getPostForgeDimensions();
       ConfigManager.updateWsDimensions();
    }
```

15. Archive index and Forge state. The newly created index and the state files in Forge's state directory are archived on the indexing server.

```
    // archive state files, index
    Forge.archiveState();
    Dgidx.archiveIndex();
```

16. Cycle LogServer. The LogServer is stopped and restarted. During the downtime, the LogServer's error and output logs are archived.

```
    // cycle LogServer
    LogServer.cycle();
```

17. Release Lock. The "`update_lock`" flag is removed from the EAC, indicating that another update may be started.

```
    // release lock
    LockManager.releaseLock("update_lock");

    log.info("Baseline update script finished.");
      } else {
    log.warning("Failed to obtain lock.");
      }
    ]]>
  </bean-shell-script>
</script>
```

Related Links

Dgraph partial update script on page 55

The partial update script defined in the `AppConfig.xml` document for a Dgraph deployment is included in this section, with numbered steps indicating the actions performed at each point in the script.

Dgraph partial update script

The partial update script defined in the `AppConfig.xml` document for a Dgraph deployment is included in this section, with numbered steps indicating the actions performed at each point in the script.

```
<script id="PartialUpdate">
  <bean-shell-script>
    <![CDATA[
```

1. Obtain lock. The partial update attempts to set an "`update_lock`" flag in the EAC to serve as a lock or mutex. If the flag is already set, this step fails, ensuring that the update cannot be started more than once simultaneously, as this would interfere with data processing. The flag is removed in the case of an error or when the script completes successfully.

   ```
   log.info("Starting partial update script.");
     // obtain lock
     if (LockManager.acquireLock("update_lock")) {
   ```

2. Validate data readiness. Test that the EAC contains at least one flag with the prefix "`partial_ex¬tract::`". One of these flags should be created for each successfully and completely extracted file, with the prefix "`partial_extract::`" prepended to the extracted file name (e.g. "`partial_ex¬tract::adds.txt.gz`"). These flags are deleted during data processing and must be created as new files are extracted.

   ```
   // test if data is ready for processing
   if (PartialForge.isPartialDataReady()) {
   ```

3. Archive partial logs. The `logs/partial` directory is archived, to create a fresh logging directory for the partial update process and to save the previous run's logs.

   ```
   // archive logs
   PartialForge.archiveLogDir();
   ```

4. Clean processing directories. Files from the previous update are removed from the `data/partials/processing`, `data/partials/forge_output`, and `data/temp` directories.

   ```
   // clean directories
   PartialForge.cleanDirs();
   ```

5. Move data and config to processing directory. Extracted files in `data/partials/incoming` with matching "`partials_extract::`" flags in the EAC are moved to `data/partials/processing`. Configuration files are copied from `config/pipeline` to `data/processing`.

   ```
   // fetch extracted data files to forge input
   PartialForge.getPartialIncomingData();

   // fetch config files to forge input
   PartialForge.getConfig();
   ```

6. Forge. The partial update Forge process executes.

   ```
   // run ITL
   PartialForge.run();
   ```

7. Apply timestamp to updates. The output XML file generated by the partial update pipeline is renamed to include a timestamp, to ensure it is processed in the correct order relative to files generated by previous or following partial update processes.

   ```
   // timestamp partial, save to cumulative partials dir
   PartialForge.timestampPartials();
   ```

8. Copy updates to cumulative updates. The timestamped XML file is copied into the cumulative updates directory.

```
    PartialForge.fetchPartialsToCumulativeDir();
```

9. Distribute update to each server. A single copy of the partial update file is distributed to each server specified in the configuration.

```
    // distribute partial update, update Dgraphs
    DgraphCluster.copyPartialUpdateToDgraphServers();
```

10. Update MDEX Engines. The Dgraph processes are updated. Engines are updated according to the `updateGroup` property specified for each Dgraph. The update process for each Dgraph is as follows:

    a. Copy update files into the `dgraph_input/updates` directory.
    b. Trigger a configuration update in the Dgraph by calling the URL `admin?op=update`.

```
    DgraphCluster.applyPartialUpdates();
```

11. Archive cumulative updates. The newly generated update file (and files generated by all partial updates processed since the last baseline) are archived on the indexing server.

```
    // archive partials
    PartialForge.archiveCumulativePartials();
```

12. Release Lock. The "`update_lock`" flag is removed from the EAC, indicating that another update may be started.

```
    // release lock
    LockManager.releaseLock("update_lock");
    log.info("Partial update script finished.");
      }
      else {
        log.warning("Failed to obtain lock.");
      }
    ]]>
  </bean-shell-script>
</script>
```

Preventing non-nillable element exceptions

When running the partial updates script, you may see a Java exception similar to this example:

```
INFO: Starting copy utility 'copy_partial_update_to_host_MDEXHost1'.
Oct 20, 2008 11:46:37 AM org.apache.axis.encoding.ser.BeanSerializer seri¬
alize
SEVERE: Exception:
java.io.IOException: Non nillable element 'fromHostID' is null.
...
```

If this occurs, make sure that the following properties are defined in the `AppConfig.xml` configuration file:

```
<dgraph-defaults>
  <properties>
      ...
      <property name="srcPartialsDir" value="./data/partials/forge_output"
 />
      <property name="srcPartialsHostId" value="ITLHost" />
      <property name="srcCumulativePartialsDir" value="./data/partials/cu¬
mulative_partials" />
```

```
      <property name="srcCumulativePartialsHostId" value="ITLHost" />
      ...
    </properties>
  ...
</dgraph-defaults>
```

The reason is that the script is obtaining the `fromHostID` value from this section.

Running partial updates with parallel Forge

If you have a configuration with two MDEX Engine servers each of which hosts two Dgraphs, you may want to run partial updates on each of these servers in parallel. This would require you to customize jobs for the EAC to make this happen. If you do this, keep in mind that the EAC Server expects all jobs sent to it to be unique across all servers.

Therefore, if you are customizing more than one job to the EAC Server, ensure that the jobs are created with a different name. This is because, even though each of these jobs runs on a separate MDEX Engine server and is unique on that server, the EAC Server expects all jobs to be unique across all servers.

Related Links

Dgraph baseline update script on page 52

The baseline update script defined in the `AppConfig.xml` document for a Dgraph deployment is included in this section, with numbered steps indicating the actions performed at each point in the script.

Agraph without parallel Forge baseline update script

The baseline update script defined in the `AppConfig.xml` document for an Agraph without parallel Forge deployment is included in this section, with numbered steps indicating the actions performed at each point in the script.

```
<script id="BaselineUpdate">
    <![CDATA[
  log.info("Starting baseline update script.");
```

1. Obtain lock. The baseline update attempts to set an "`update_lock`" flag in the EAC to serve as a lock or mutex. If the flag is already set, this step fails, ensuring that the update cannot be started more than once simultaneously, as this would interfere with data processing. The flag is removed in the case of an error or when the script completes successfully.

   ```
    // obtain lock
    if (LockManager.acquireLock("update_lock")) {
   ```

2. Validate data readiness. Check that a flag called "`baseline_data_ready`" has been set in the EAC. This flag is set as part of the data extraction process to indicate that files are ready to be processed (or, in the case of an application that uses direct database access, the flag indicates that a database staging table has been loaded and is ready for processing). This flag is removed as soon as the script copies the data out of the `data/incoming` directory, indicating that new data may be extracted.

   ```
      // test if data is ready for processing
      if (Forge1.isDataReady())
      {
   ```

3. If Workbench integration is enabled, download and merge Oracle Endeca Workbench configuration. The `ConfigManager` copies all Developer Studio config files to the `complete_index_config` directory. Then, all Workbench-maintained configuration files are downloaded. Any files that are configured in the `ConfigManager` component to be maintained by Oracle Endeca Workbench are copied to the `complete_index_config` directory, overwriting the Developer Studio copy of the same file, if one exists. The final result is a complete set of configuration files for Forge to use. If Workbench integration is not enabled, the `ConfigManager` copies all Developer Studio config files to the `complete_index_config` directory.

```
if (ConfigManager.isWebStudioEnabled()) {
  // get Web Studio config, merge with Dev Studio config
  ConfigManager.downloadWsConfig();
  ConfigManager.fetchMergedConfig();
} else {
  ConfigManager.fetchDsConfig();
}
```

4. Clean processing directories. Files from the previous update are removed from the `data/processing`, `data/forge_output`, `data/temp`, `data/dgidxs/[DgidxID]/dgidx_output`, and `data/agidx_output`directories.

```
// clean directories
Forge1.cleanDirs();
IndexingCluster.cleanDirs();
```

5. Copy data to processing directory. Extracted data in `data/incoming` is copied to `data/processing`.

```
// fetch extracted data files to forge input
Forge1.getIncomingData();
```

6. Release Lock. The "`baseline_data_ready`" flag is removed from the EAC, indicating that the incoming data has been retrieved for baseline processing.

```
LockManager.releaseLock("baseline_data_ready");
```

7. Copy config to processing directory. Configuration files are copied from `data/complete_index_config` to `data/processing`.

```
// fetch config files to forge input
Forge1.getConfig();
```

8. Archive Forge logs. The `logs/forges/Forge1` directory is archived, to create a fresh logging directory for the Forge process and to save the previous Forge's logs.

```
// archive logs and run ITL
Forge1.archiveLogDir();
```

9. Forge. The Forge process executes.

```
Forge1.run();
```

10. Copy data to `dgidx_input` directories. Forged data in `data/forge_output` is copied to `data/dgidxs/[DgidxID]/dgidx_input` directories.

```
IndexingCluster.getDgidxIncomingData();
```

11. Copy config to `dgidx_input` directories. Configuration files in `data/forge_output` are copied to `data/ dgidxs/[DgidxID]/dgidx_input` directories.

```
IndexingCluster.getDgidxConfig();
```

12. Archive Dgidx and Agidx logs. The logs `/dgidx/[DgidxID]` and `logs/agidx/[AgidxID]` directories are archived, to create a fresh logging directory for the indexing processes and to save the previous indexing processes' logs.

```
      IndexingCluster.archiveLogDir();
```

13. Dgidx and Agidx. The Dgidx processes and Agidx process execute.

```
      IndexingCluster.run();
```

14. Distribute index to each server. A single copy of the new index is distributed to each server that hosts a graph. If multiple graphs are located on the same server but specify different `srcIndexDir` attributes, multiple copies of the index will be delivered to that server.
15. Update MDEX Engines. The graphs are updated. Engines are updated according to the `restartGroup` property specified for each graph. The update process for each graph in the restart group is as follows:

    a. Create `agraph_input_new` and create a local copy of the new index in `agraph_input_new` for each Agraph.
    b. Create `dgraph_input_new` and create a local copy of the new index in `dgraph_input_new` for each Dgraph.
    c. Stop each Agraph.
    d. Stop each Dgraph.
    e. Rename `dgraph_input` to `dgraph_input_old` for each Dgraph.
    f. Rename `dgraph_input_new` to `dgraph_input` for each Dgraph.
    g. Rename `agraph_input` to `agraph_input_old` for each Agraph.
    h. Rename `agraph_input_new` to `agraph_input` for each Agraph.
    i. Archive each Dgraph logs (e.g. `logs/dgraphs/Dgraph1`) directory.
    j. Archive each Agraph logs (e.g. `logs/agraphs/Agraph1`) directory.
    k. Start each Dgraph.
    l. Start each Agraph.
    m. Remove `dgraph_input_old` for each dgraph.
    n. Remove `agraph_input_old` for each Agraph.

    This somewhat complex update functionality is implemented to minimize the amount of time that a graph is stopped. This restart approach ensures that the graphs are stopped just long enough to rename two directories for each Dgraph.

```
      // distributed index, update graphs
      DistributeIndexAndApply.run();
```

```
  <script id="DistributeIndexAndApply">
    <bean-shell-script>
      <![CDATA[
    AgraphCluster.cleanDirs();
    AgraphCluster.copyIndexToAgraphServers();
    AgraphCluster.copyIndexToDgraphServers();
    AgraphCluster.applyIndex();
      ]]>
    </bean-shell-script>
  </script>
```

16. If Workbench integration is enabled, upload post-Forge dimensions to Oracle Endeca Workbench. The latest dimensions generated by the Forge process are uploaded to Oracle Endeca Workbench,

to ensure that any new dimensions (including autogen dimensions and external dimensions) are available to Oracle Endeca Workbench for use in, for example, dynamic business rule triggers.

```
      // if Workbench is integrated, update Workbench with latest
      // dimension values
      if (ConfigManager.isWebStudioEnabled()) {
        ConfigManager.cleanDirs();
        Forge1.getPostForgeDimensions();
        ConfigManager.updateWsDimensions();
      }
```

17. Archive index and Forge state. The newly created index and the state files in Forge's state directory are archived on the indexing servers.

```
      // archive state files, index
      Forge1.archiveState();
      IndexingCluster.archiveIndex();
```

18. Cycle LogServer. The LogServer is stopped and restarted. During the downtime, the LogServer's error and output logs are archived.

```
      // cycle LogServer
      LogServer.cycle();
    }
    else
    {
      log.warning("Baseline data not ready for processing.");
    }
```

19. Release Lock. The "`update_lock`" flag is removed from the EAC, indicating that another update may be started.

```
    // release lock
    LockManager.releaseLock("update_lock");

    log.info("Baseline update script finished.");
  }
  else {
    log.warning("Failed to obtain lock.");
  }
    ]]>
  </bean-shell-script>
</script>
```

Related Links

Agraph with parallel Forge baseline update script on page 60
The baseline update script defined in the `AppConfig.xml` document for an Agraph with parallel Forge deployment is included in this section, with numbered steps indicating the actions performed at each point in the script.

Agraph with parallel Forge baseline update script

The baseline update script defined in the `AppConfig.xml` document for an Agraph with parallel Forge deployment is included in this section, with numbered steps indicating the actions performed at each point in the script.

```
<script id="BaselineUpdate">
    <![CDATA[
  log.info("Starting baseline update script.");
```

1. Obtain lock. The baseline update attempts to set an "update_lock" flag in the EAC to serve as a lock or mutex. If the flag is already set, this step fails, ensuring that the update cannot be started more than once simultaneously, as this would interfere with data processing. The flag is removed in the case of an error or when the script completes successfully.

   ```
   // obtain lock
   if (LockManager.acquireLock("update_lock")) {
   ```

2. Validate data readiness. Check that a flag called "baseline_data_ready" has been set in the EAC. This flag is set as part of the data extraction process to indicate that files are ready to be processed (or, in the case of an application that uses direct database access, the flag indicates that a database staging table has been loaded and is ready for processing). This flag is removed as soon as the script copies the data out of the data/incoming directory, indicating that new data may be extracted.

   ```
   // test if data is ready for processing
   if (ForgeSplitData.isDataReady())
   {
   ```

3. If Workbench integration is enabled, download and merge Oracle Endeca Workbench configuration. The ConfigManager copies all Developer Studio config files to the complete_index_config directory. Then, all Workbench-maintained configuration files are downloaded. Any files that are configured in the ConfigManager component to be maintained by Oracle Endeca Workbench are copied to the complete_index_config directory, overwriting the Developer Studio copy of the same file, if one exists. The final result is a complete set of configuration files for Forge to use. If Workbench integration is not enabled, the ConfigManager copies all Developer Studio config files to the complete_index_config directory.

   ```
   if (ConfigManager.isWebStudioEnabled()) {
     // get Web Studio config, merge with Dev Studio config
     ConfigManager.downloadWsConfig();
     ConfigManager.fetchMergedConfig();
   } else {
     ConfigManager.fetchDsConfig();
   }
   ```

4. Clean processing directories. Files from the previous update are removed from the data/processing, data/forge_output, data/temp, data/forges/[ForgeID]/processing, data/forges/[ForgeID]/forge_output, data/forges/[ForgeID]/temp, data/dgidxs/[DgidxID]/dgidx_output, and data/agidx_outputdirectories.

   ```
   // clean directories
   ForgeSplitData.cleanDirs();
   ForgeCluster.cleanDirs();
   IndexingCluster.cleanDirs();
   ```

5. Copy data to processing directory. Extracted data in data/incoming is copied to data/processing.

   ```
   // fetch extracted data files to forge input
   ForgeSplitData.getIncomingData();
   ```

6. Release Lock. The "baseline_data_ready" flag is removed from the EAC, indicating that the incoming data has been retrieved for baseline processing.

   ```
   LockManager.releaseLock("baseline_data_ready");
   ```

7. Copy config to processing directory. Configuration files are copied from `data/complete_index_config` to `data/processing`.

```
// fetch config files to forge input
ForgeSplitData.getConfig();
```

8. Archive Forge logs. The `logs/forges/ForgeSplitData` directory is archived, to create a fresh logging directory for the Forge process and to save the previous Forge's logs.

```
// archive logs and run ITL
ForgeSplitData.archiveLogDir();
```

9. Forge. The Forge process that splits the extracted data executes.

```
ForgeSplitData.run();
```

10. Copy split data to processing directory. Split data in `data/forge_output` is copied to `data/forges/[ForgeID]/processing` for each Forge in the Parallel Forge cluster.

```
ForgeCluster.getData();
```

11. Archive Parallel Forge logs. The `logs/forges/[ForgeID]` directories are archived, to create a fresh logging directory for the Forge processes and to save the previous Forge processes' logs.

```
ForgeCluster.archiveLogDir();
```

12. Forge in parallel. The Forge server and Forge client processes execute in parallel.

```
ForgeCluster.run();
```

13. Copy data to `dgidx_input` directories. Forged data in `data/forges/[ForgeID]/forge_output` is copied to `data/dgidxs/[DgidxID]/dgidx_input` directories.

```
IndexingCluster.getDgidxIncomingData();
```

14. Copy config to `dgidx_input` directories. Configuration files in `data/forges/[ForgeID]/forge_output` are copied to `data/dgidxs/[DgidxID]/dgidx_input` directories.

```
IndexingCluster.getDgidxConfig();
```

15. Archive Dgidx and Agidx logs. The `logs/dgidx/[DgidxID]` and `logs/agidx/[AgidxID]` directories are archived to create a fresh logging directory for the indexing processes and to save the previous processes' logs.

```
IndexingCluster.archiveLogDir();
```

16. Dgidx and Agidx. The Dgidx processes and Agidx process execute.

```
IndexingCluster.run();
```

17. Distribute index to each server. A single copy of the new index is distributed to each server that hosts a graph. If multiple graphs are located on the same server but specify different `srcIndexDir` attributes, multiple copies of the index will be delivered to that server.
18. Update MDEX Engines. The graphs are updated. Engines are updated according to the `restartGroup` property specified for each graph. The update process for each graph in the restart group is as follows:
    a. Create `agraph_input_new` and create a local copy of the new index in `agraph_input_new` for each Agraph.
    b. Create `dgraph_input_new` and create a local copy of the new index in `dgraph_input_new` for each Dgraph.

c. Stop each Agraph.
d. Stop each Dgraph.
e. Rename `dgraph_input` to `dgraph_input_old` for each Dgraph.
f. Rename `dgraph_input_new` to `dgraph_input` for each Dgraph.
g. Rename `agraph_input` to `agraph_input_old` for each Agraph.
h. Rename `agraph_input_new` to `agraph_input` for each Agraph.
i. Archive each Dgraph logs (e.g. `logs/dgraphs/Dgraph1`) directory.
j. Archive each Agraph logs (e.g. `logs/agraphs/Agraph1`) directory.
k. Start each Dgraph.
l. Start each Agraph.
m. Remove `dgraph_input_old` for each dgraph.
n. Remove `agraph_input_old` for each Agraph.

This somewhat complex update functionality is implemented to minimize the amount of time that a graph is stopped. This restart approach ensures that the graphs are stopped just long enough to rename two directories for each Dgraph.

```
      // distributed index, update graphs
      DistributeIndexAndApply.run();
```

```
  <script id="DistributeIndexAndApply">
    <bean-shell-script>
      <![CDATA[
    AgraphCluster.cleanDirs();
    AgraphCluster.copyIndexToAgraphServers();
    AgraphCluster.copyIndexToDgraphServers();
    AgraphCluster.applyIndex();
      ]]>
    </bean-shell-script>
  </script>
```

19. If Workbench integration is enabled, upload post-Forge dimensions to Oracle Endeca Workbench. The latest dimensions generated by the Forge process are uploaded to Oracle Endeca Workbench, to ensure that any new dimensions (including autogen dimensions and external dimensions) are available to Oracle Endeca Workbench for use in, for example, dynamic business rule triggers.

```
      // if Workbench is integrated, update Workbench with latest
      // dimension values
      if (ConfigManager.isWebStudioEnabled()) {
        ConfigManager.cleanDirs();
        ForgeServer.getPostForgeDimensions();
        ConfigManager.updateWsDimensions();
      }
```

20. Archive index and Forge state. The newly created index and the state files in Forge's state directory are archived on the indexing servers.

```
      // archive state files, index
      ForgeCluster.archiveState();
      IndexingCluster.archiveIndex();
```

21. Cycle LogServer. The LogServer is stopped and restarted. During the downtime, the LogServer's error and output logs are archived.

```
      // cycle LogServer
      LogServer.cycle();
    }
    else
```

```
    {
      log.warning("Baseline data not ready for processing.");
    }
```

22. Release Lock. The "`update_lock`" flag is removed from the EAC, indicating that another update may be started.

```
    // release lock
    LockManager.releaseLock("update_lock");

    log.info("Baseline update script finished.");
  }
  else {
    log.warning("Failed to obtain lock.");
  }
    ]]>
  </bean-shell-script>
</script>
```

Related Links

Agraph without parallel Forge baseline update script on page 57

The baseline update script defined in the `AppConfig.xml` document for an Agraph without parallel Forge deployment is included in this section, with numbered steps indicating the actions performed at each point in the script.

Configuration update script

The configuration update script defined in the `AppConfig.xml` document is included in this section, with numbered steps indicating the actions performed at each point in the script.

Note that the script starts by checking if Oracle Endeca Workbench integration is enabled, taking no action (other than logging a message) if disabled.

```
<script id="ConfigUpdate">
    <bean-shell-script>
      <![CDATA[
    log.info("Starting dgraph config update script.");
    if (ConfigManager.isWebStudioEnabled()) {
```

1. Download the Oracle Endeca Workbench Dgraph config files. Download Workbench-maintained configuration files that can be applied to a Dgraph. Remove any files from this set that are not configured to be maintained in Oracle Endeca Workbench in the `ConfigManager` component.

   ```
   ConfigManager.downloadWsDgraphConfig();
   ```

2. Clean working directories. Clear any files in the local Dgraph configuration directories to which files are distributed on each Dgraph server.

   ```
   DgraphCluster.cleanLocalDgraphConfigDirs();
   ```

3. Distribute configuration files to each server. A single copy of the Dgraph configuration files is distributed to each server specified in the configuration.

   ```
   DgraphCluster.copyDgraphConfigToDgraphServers();
   ```

4. Update MDEX Engines. The Dgraph processes are updated. Engines are updated according to the `restartGroup` property specified for each Dgraph. The update process for each Dgraph is as follows:

a. Copy configuration files into the `dgraph_input` directory.
b. Trigger a configuration update in the Dgraph by calling the URL `config?op=update`.
c. Flush the Dgraph's dynamic cache to ensure the new configuration is applied by calling the URL `admin?op=flush`.

This somewhat complex update functionality is implemented to minimize the amount of time that a graph is stopped. This restart approach ensures that the graphs are stopped just long enough to rename two directories for each Dgraph.

```
      DgraphCluster.applyConfigUpdate();
    } else {
      log.warning("Workbench integration is disabled. No action will be
 taken.");
    }
    log.info("Finished updating dgraph config.");
      ]]>
    </bean-shell-script>
  </script>
```

Report generation

Four report generation scripts are defined in the `AppConfig.xml` document.

Two of the scripts are used to generate XML reports for Oracle Endeca Workbench and two generate HTML reports that can be viewed in a browser. All scripts share similar functionality, so only one is included below, with numbered steps indicating the actions performed at each point in the script.

```
  <script id="DailyReports">
    <bean-shell-script>
      <![CDATA[
    log.info("Starting daily Workbench report generation script.");
```

1. Obtain lock. The report generation script attempts to set a "`report_generator_lock`" flag in the EAC to serve as a lock or mutex. If the flag is already set, this step fails, ensuring that the report generator cannot be started more than once simultaneously, as the default report generators share input directories and working directories. The flag is removed in the case of an error or when the script completes successfully.

   ```
      if (LockManager.acquireLock("report_generator_lock")) {
   ```

2. Clean working directories. Clear any files in the report generator's input directory.

   ```
      // clean report gen input dir
      DailyReportGenerator.cleanInputDir();
   ```

3. Distribute configuration files to each server. A single copy of the Dgraph configuration files is distributed to each server specified in the configuration.

   ```
   DgraphCluster.copyDgraphConfigToDgraphServers();
   ```

4. Roll LogServer. If the LogServer is actively writing to a file and the file is required for the specified time range, the LogServer needs to be rolled in order to free up the log file. This code handles that test and invokes the roll administrative URL command on the LogServer, if necessary.

   ```
      // roll the logserver, if the report requires the active log file
      if (LogServer.isActive() &&
          LogServer.yesterdayIncludesLatestLogFile()) {
        LogServer.callLogserverRollUrl();
      }
   ```

5. Retrieve logs for specified report. The LogServer identifies log files in its output directory that are required to generate a report for the requested date range. Those files are copied to the target directory configured for the LogServer. Note that this step could be modified to include retrieving logs from multiple LogServers, if more than one is deployed.

```
      // retrieve required log files for processing
      LogServer.copyYesterdayLogFilesToTargetDir();
```

6. Update Report Generator to the appropriate time range and output file name. Oracle Endeca Workbench requires reports to be named according to a time stamp convention. The Report Generator component's provisioning is updated to specify the appropriate time range, time series and output filename. The output file path in the existing provisioning is updated to use the same path, but to use the date stamp as the filename. Files default to a "`.xml`" extension, though the component will attempt to retain a "`.html`" extension, if specified in the `AppConfig.xml`.

```
      // update report generator to the appropriate dates, time series
      // and to output a timestamped file, as required by Workbench
      DailyReportGenerator.updateProvisioningForYesterdayReport();
```

7. Archive logs. If one or more files were copied into the report generator's input directory, report generation will proceed. Start by archiving logs associated with the previous report generator execution.

```
      if (DailyReportGenerator.reportInputDirContainsFiles()) {
        // archive logs
        DailyReportGenerator.archiveLogDir();
```

8. Run report generator. Execute the report generation process.

```
        // generate report
        DailyReportGenerator.run();
```

9. Copy report to Oracle Endeca Workbench report directory. By default, Oracle Endeca Workbench reads reports from a directory in its workspace. Typically, the directory is `[ENDECA_TOOLS_CONF]/reports/[appName]/daily` or `[Endeca_TOOLS_CONF]/reports/[appName]/weekly`. Starting in Oracle Endeca Workbench 1.0.1, this location can be configured by provisioning a host named "`webstudio`" with a custom directory named "`webstudio-report-dir`." The Deployment Template provisions this directory and delivers generated reports to that location for Workbench to read. The report file (and associated charts) will be copied to this directory, as specified in the `AppConfig.xml`, which defaults to `[appdir]/reports`. Note that this step is not necessary for HTML reports, as those reports are not viewed in Oracle Endeca Workbench.

```
        // copy generated report and charts to web studio directory
        // defined in "webstudio" host and its "webstudio-report-dir"
        // directory
        reportHost = "webstudio";
       absDestDir = PathUtils.getAbsolutePath(webstudio.getWorkingDir(),

            webstudio.getDirectory("webstudio-report-dir"));
        isDaily = true;
        DailyReportGenerator.copyReportToWebStudio(reportHost,
            absDestDir, isDaily);
      }
      else {
        log.warning("No log files for report generator to process.");
      }

      LockManager.releaseLock("report_generator_lock");
      log.info("Finished daily Workbench report generation.");
    }
```

```
    else {
      log.warning("Failed to obtain lock.");
    }
      ]]>
    </bean-shell-script>
  </script>
```

Chapter 5

Oracle Endeca Workbench Integration and Deployment

This section describes the Deployment Template functionality that allows deployments to integrate with Oracle Endeca Workbench.

Oracle Endeca Workbench Integration functions

The Deployment Template scripts include functionality that allows deployments to integrate with Oracle Endeca Workbench. Integration involves three functions: configuration management, process control, and reporting.

- Configuration management. Oracle Endeca Workbench is primarily a tool that allows business users to maintain a subset of application configuration files. Scripts that integrate with Oracle Endeca Workbench need to retrieve configuration files from the Workbench configuration store and manage the process of merging those configuration files with ones created and maintained by developers in Developer Studio. In addition, scripts may need to upload configuration into the Workbench, to ensure that the tool is up to date with the configuration being used in the production environment.
- Process control. Oracle Endeca Workbench requires an Oracle Endeca instance in order to apply and preview changes to the configuration files maintained in the tool. Scripts that integrate with the Workbench must define and maintain an MDEX Engine that the Workbench will use to preview changes before they are applied to the production environment.
- Reporting. Oracle Endeca Workbench provides a convenient interface for viewing reports produced by the Report Generator. Scripts that generate reports need to deliver files to a location from which Oracle Endeca Workbench will load reports in order for users to view them.

Configuration management

The Deployment Template implements a configuration management approach based on a simple business process.

The template requires that all configuration files be managed in a single location. That is, files are either maintained by Oracle Endeca Workbench or by Developer Studio, but multiple sources of configuration for the same file are not allowed. Given this rule, the template assumes that all files are maintained in Developer Studio with the exception of those explicitly identified (in the `ConfigManager` component definition) as being maintained in Oracle Endeca Workbench. When creating the complete

configuration of the application, the Deployment Template takes the Developer Studio copy of each configuration file, overwriting those that are to be maintained in Oracle Endeca Workbench with the copies downloaded from the Workbench.

About updating Workbench configuration

One implication of the single-source business process is that configuration does not need to be uploaded to Oracle Endeca Workbench after it is initially uploaded.

That is, the Workbench is used as the primary source of configuration for a subset of files, and need not be updated with configuration from other sources. Because of this, the Deployment Template does not upload configuration settings to the Workbench during baseline or partial update scripts. It is expected that the Workbench is initialized with configuration when the application is deployed and that additional configuration updates are uploaded manually when necessary.

When deploying the application, users who have enabled integration with Oracle Endeca Workbench will run the `initialize_services.[bat|sh]` script to upload configuration to the Workbench. On an ad hoc basis, configuration can be uploaded to the Workbench to update any files that are not maintained by the Workbench. For example, if a new rule group is created and needs to be made available in Oracle Endeca Workbench, users can run the `update_web_studio_config.[bat|sh]` script to upload the latest configuration files to the Workbench. Note that this script merges the configuration files according to the previously defined process (skipping the merge step if no files are retrieved from the Workbench) before uploading to the Workbench, ensuring that only files that are not maintained in the Workbench are updated, and all changes developed and maintained in the Workbench remain there. The process of uploading configuration to Workbench requires obtaining locks on all Workbench configuration resources. This means that no locks may be held by users. The update script does not attempt to break locks, expecting that system administrators will ensure that Oracle Endeca Workbench is free of users before attempting to update the tool with new configuration.

About extending Workbench configuration

As with any other component in the template, configuration management may be modified and extended to meet the needs of the project.

In cases where the business process defined by the Deployment Template is not suitable to a project, a new Configuration Management object can be implemented to perform more complex tasks such as comparing and merging files maintained in both Developer Studio and Oracle Endeca Workbench.

About promoting configuration to production

The Deployment Template `BaselineUpdate` script retrieves Workbench-maintained configuration files and applies them when the update is being processed.

In addition, the `ConfigUpdate` script retrieves Workbench-maintained Dgraph configuration files (i.e. files that do not require a baseline update to apply) and applies them to the Dgraph cluster.

The default behavior is to promote saved configuration with regularly scheduled baseline index updates. Users may also want to automate the `ConfigUpdate` script to promote changes more frequently. In some cases, it may be appropriate to expose an interface for allowing users to promote changes on demand. The `ConfigUpdate` script is provisioned and can be executed via a Web service invocation

(through any tool or custom UI that can make a Web service call) or from the EAC Admin console in the Oracle Endeca Workbench.

Process control

Different project and deployment needs require varying deployment approaches to meet a project's requirements. The following sections describe a few approaches to deploying Oracle Endeca Workbench and how the Deployment Template can be configured for each approach.

No Workbench integration

Some deployments do not require Workbench integration.

These simple cases require a single configuration change in the `AppConfig.xml` document. This change can be selected during installation, when the user is prompted about whether Oracle Endeca Workbench integration should be enabled.

The following configuration should be changed in the `ConfigManager` component to disable Workbench integration:

```
<property name="webStudioEnabled" value="false" />
```

Oracle Endeca Workbench deployed in a preview environment

This deployment approach calls for an environment outside of production to be deployed for Oracle Endeca Workbench.

This environment requires a standalone Deployment Template installation, which runs its own baseline and partial index updates. In the separate production environment, configuration files are retrieved from Oracle Endeca Workbench and promoted to the production cluster during updates.

By default, Oracle Endeca Workbench uses an EAC script to update MDEX Engines with new configuration each time the **Save Changes** button is pressed. The default implementation of this script applies configuration to each Dgraph provisioned in the EAC for the application with which the Workbench application is associated. With this deployment approach, this functionality is appropriate as the application deployed in the preview environment is dedicated to preview and any Dgraphs associated with the deployment can be updated by the Workbench.

Related Links

Configuring an Oracle Endeca Workbench deployment in a preview environment on page 72
This deployment approach requires a standalone Deployment Template installation, which runs its own baseline and partial index updates.

Oracle Endeca Workbench deployed with a preview Dgraph on page 73
This deployment approach calls for a new Dgraph to be configured in the production environment to serve as the Workbench preview Dgraph.

Configuring an Oracle Endeca Workbench deployment in a preview environment

This deployment approach requires a standalone Deployment Template installation, which runs its own baseline and partial index updates.

To configure this type of deployment, install the Deployment Template on the server or servers designated as the preview environment.

The following configuration update should be specified in the `AppConfig.xml` document in this preview environment.

1. Remove the step that uploads post-Forge dimensions to Oracle Endeca Workbench during the baseline update.

   This step should only be performed in production, to ensure that new dimension values are only generated in the production environment and that all configuration created in Oracle Endeca Workbench is based on dimension values in the production environment.

   The following lines should be removed from the preview environment's baseline update script.

   ```
   ConfigManager.cleanDirs();
   Forge.getPostForgeDimensions();
   ConfigManager.updateWsDimensions();
   ```

2. In the separate production environment, the following configuration is required:
   a) Update `AppConfig.xml` to specify the host and port of Oracle Endeca Workbench in the preview environment.
   b) As these deployments may end up on different sides of a production firewall, this deployment may also require opening the Oracle Endeca Workbench port for communication through the firewall. This communication goes in both directions, as the production environment needs to retrieve files from Oracle Endeca Workbench and also update dimension values in the Workbench.
3. In addition to these configuration changes, it may be necessary to keep the preview environment up to date with the latest data and configuration changes applied to production. The following approaches may be taken to fulfill this requirement.

   - Deploy the production index on the preview server. Each time a new index is built by the production baseline update process, the index can be copied to the preview environment and applied to the preview Dgraph(s). This approach may require deploying partial updates onto the preview server as they are processed in production, if it is important for Oracle Endeca Workbench to see these partial updates while previewing.
   - Deploy production data and configuration on the preview server. Each time new data is made available for processing in production or new configuration files are deployed to production, the same data and configuration should be made available to the preview environment. With this approach, the preview environment runs baseline and partial updates to process the same data as production, ensuring that the same index is built in the preview environment just after that index is built or updated in production. This approach may also require copying the Forge autogen state file from the production environment to the preview environment, so that the dimension values used in the preview index are identical to those used in production. To ensure that no new values are generated in the preview environment, the `--noAutoGen` may be added to the Forge processes in the preview environment.

Related Links

Oracle Endeca Workbench deployed in a preview environment on page 71

This deployment approach calls for an environment outside of production to be deployed for Oracle Endeca Workbench.

Oracle Endeca Workbench deployed with a preview Dgraph

This deployment approach calls for a new Dgraph to be configured in the production environment to serve as the Workbench preview Dgraph.

This Dgraph is updated along with others in the production environment each time an update is processed, but it does not serve traffic for the production application. Instead, this Dgraph is used for preview by Oracle Endeca Workbench.

By default, Oracle Endeca Workbench uses an EAC script to update MDEX Engines with new configuration each time the **Save Changes** button is pressed. With this deployment approach, the default functionality would update each production Dgraph in addition to the preview Dgraph. There are two ways to override this default behavior.

Related Links

Overriding the default behavior of the update functionality on page 73
By default, Oracle Endeca Workbench uses an EAC script to update MDEX Engines with new configuration each time the **Save Changes** button is pressed. With the preview Dgraph deployment approach, the default functionality would update each production Dgraph in addition to the preview Dgraph. There are two ways to override this default behavior.

Oracle Endeca Workbench deployed in a preview environment on page 71
This deployment approach calls for an environment outside of production to be deployed for Oracle Endeca Workbench.

Overriding the default behavior of the update functionality

By default, Oracle Endeca Workbench uses an EAC script to update MDEX Engines with new configuration each time the **Save Changes** button is pressed. With the preview Dgraph deployment approach, the default functionality would update each production Dgraph in addition to the preview Dgraph. There are two ways to override this default behavior.

- A script may be provisioned with a reserved name, instructing Oracle Endeca Workbench to execute the custom script rather than using its default update script.
  The following configuration excerpt demonstrates how this script might be implemented.

```
<script id="EndecaMDEXUpdateScript">
  <log-dir>./logs</log-dir>
  <provisioned-script-command>
    ./control/runcommand.bat EndecaMDEXUpdateScript
  </provisioned-script-command>
  <bean-shell-script>
    <![CDATA[
  log.info("Applying config to Web Studio preview dgraph.");

  // don't filter, apply all rules to preview dgraph
  filterInactiveRules = false;

  // download MDEX config from Web Studio to
  // preview dgraph's input directory
  getWsDgraphFiles = new GetWSDgraphFilesUtility(WSDgraph.getAppName(),

    WSDgraph.getEacHost(), WSDgraph.getEacPort(),
```

```
    WSDgraph.isSslEnabled(), WSDgraph.getDataPrefix());
  getWsDgraphFiles.init(WSDgraph.getHostId(), ConfigManager.getWsHost(),

    ConfigManager.getWsPort(), WSDgraph.getInputDir(),
    filterInactiveRules, WSDgraph.getWorkingDir());
  getWsDgraphFiles.run();

  // trigger config update (and cache flush) in preview dgraph
  WSDgraph.callDgraphConfigUpdateUrl();

  log.info("Web Studio preview dgraph updated.");
    ]]>
  </bean-shell-script>
</script>
```

Note that this script has the ID "`EndecaMDEXUpdateScript`." This is the reserved script name that must be used when provisioning the script, to ensure that Oracle Endeca Workbench recognizes and invokes the script when saving configuration changes.

a) In addition, note that the script updates a Dgraph named `WSDgraph`, which can be configured as follows:

```
<dgraph id="WSDgraph" host-id="ITLHost" port="1">
  <properties>
    <property name="restartGroup" value="WebStudio" />
  </properties>
  <log-dir>./logs/dgraphs/WSDgraph</log-dir>
  <input-dir>./data/dgraphs/WSDgraph/dgraph_input</input-dir>
  <update-dir>./data/dgraphs/WSDgraph/dgraph_input/updates</update-dir>

  <app-config-dir>./data/dgraphs/WSDgraph/dgraph_input</app-config-dir>
</dgraph>
```

b) The new Dgraph should also be added to the `DgraphCluster`, so that it is updated along with other Dgraphs during baseline and partial updates.

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

Once these changes have been made, the environment has a Dgraph dedicated to Workbench preview and uses the `BaselineUpdate`, `PartialUpdate`, and `ConfigUpdate` script to apply changes saved in the Workbench to the production Dgraphs.

Note: Oracle Endeca Workbench's dynamic business rule preview functionality always selects the first Dgraph provisioned for the application to test which business rules have fired. In order to have this select the dedicated Oracle Endeca Workbench preview Dgraph, the WSDgraph should be specified first in the `AppConfig.xml` file definition of the `DgraphCluster`. This ensures that, when provisioning the cluster, the Workbench Dgraph is added first, and is selected by the Workbench to determine status messages when previewing rules.

- By provisioning one or more Dgraphs with a pre-defined set of properties, Oracle Endeca Workbench can be instructed to perform its default update operation on that subset of Dgraphs. To configure a `WSDgraph` as described above, start by configuring all other Dgraphs not to be updated by Oracle

Endeca Workbench. This can be done by setting the property "WebStudioSkipConfigUpdate" in the DgraphDefaults:

```
  <dgraph-defaults>
    <properties>
...
      <property name="WebStudioSkipConfigUpdate" value="true" />
    </properties>
...
  </dgraph-defaults>
```

Note that this property can also be set individually for each Dgraph, rather than in the defaults. Once set, the property must be set to "false" for the WSDgraph, to ensure it is updated by the Oracle Endeca Workbench. In addition, another property, "webStudioMDEX" should be set on the WSDgraph to indicate to Oracle Endeca Workbench that this is the Dgraph to use when previewing the rules that have been triggered.

```
<dgraph id="WSDgraph" host-id="ITLHost" port="1">
  <properties>
    <property name="restartGroup" value="WebStudio" />
    <property name="WebStudioSkipConfigUpdate" value="false" />
    <property name="WebStudioMDEX" value="true" />
  </properties>
  <log-dir>./logs/dgraphs/WSDgraph</log-dir>
  <input-dir>./data/dgraphs/WSDgraph/dgraph_input</input-dir>
  <update-dir>./data/dgraphs/WSDgraph/dgraph_input/updates</update-dir>
  <app-config-dir>./data/dgraphs/WSDgraph/dgraph_input</app-config-dir>
</dgraph>
```

For details about these properties, refer to the *Oracle Endeca Workbench Administrator's Guide*.

There are two implications to deploying either of these approaches that should be considered.

- When using the first approach, the custom EndecaMDEXUpdateScript implementation will take longer to execute than default Workbench implementation. The script described above is simple and executes relatively quickly, but cannot execute as quickly as native Workbench code. Business users should expect a brief delay when saving changes in Oracle Endeca Workbench. For this reason, the second approach (configuring the default Oracle Endeca Workbench script) may be a better choice.
- For both approaches, the preview Dgraph's index is updated simultaneously with the production Dgraph cluster. This means that Oracle Endeca Workbench changes that require a baseline update to apply are not previewed, but are immediately applied to production. This includes changes to stop word configuration.

Related Links

Oracle Endeca Workbench deployed with a preview Dgraph on page 73
This deployment approach calls for a new Dgraph to be configured in the production environment to serve as the Workbench preview Dgraph.

Oracle Endeca Workbench deployed in a production environment

This deployment approach calls for Oracle Endeca Workbench to act directly on the production Dgraph cluster.

Note: This approach is often not recommended, as it allows changes to be made to a production application and requires unregulated updates to the production Dgraph cluster. However, some deployments may not find these concerns significant and may choose to deploy with this approach.

Without any configuration, the Deployment Template integrates with a Workbench in its local environment and Oracle Endeca Workbench assumes direct control of all of the Dgraphs provisioned for the deployed application. This means that no further configuration is required when deploying this approach in production.

It is important to note that most changes saved in Oracle Endeca Workbench are deployed to the production servers immediately when the **Save Changes** button is clicked in the Workbench. There are two exceptions to this behavior.

1. Changes that require a baseline update to apply are picked up by a scheduled baseline update. This includes changes to stop word configuration.
2. Changes to dynamic business rules must go through Oracle Endeca Workbench approval workflow before they are exposed in the production Dgraph. These changes are applied to the production Dgraphs when the **Save Changes** button is pressed, but may not be exposed to the end user until they have been approved in Oracle Endeca Workbench.

Reporting

Oracle Endeca Workbench provides an interface for viewing and analyzing reports produced by the Report Generator.

In order for Oracle Endeca Workbench to display these reports, report files and associated charts need to be created and delivered to a directory in Oracle Endeca Workbench's workspace. Alternatively, a "`webstudio`" host can be provisioned with a "`webstudio-report-dir`" custom directory, which indicates to Oracle Endeca Workbench where it should read reports for the application. In addition, the files need to be named with a date stamp to conform to Oracle Endeca Workbench's naming convention. The Deployment Template includes report generation scripts that perform these naming and copying steps to deliver reports for Oracle Endeca Workbench to read. Common extension or customization of this functionality may occur when one or more of the components in the reporting lifecycle run in different environments. The `AppConfig.xml` allows components to work independently of each other. Specifically, the LogServer can be configured to deliver files to an arbitrary directory, from where the files can be copied to another environment for report generation. Similarly, the Report Generator's output report can be delivered to an arbitrary target directory, from where the files can be copied to another environment for display in Oracle Endeca Workbench.

Chapter 6

Integrating and Running CAS Crawls

The Deployment Template provides support for running CAS crawls. This section describes the configuration changes necessary to integrate and run CAS crawls using the Deployment Template. You must upgrade the Deployment Template with the CAS WSDL client stub files for CAS integration to work. If you have not upgraded the Deployment Template, see the CAS Installation Guide before continuing with this chapter.

About storage types for CAS crawls

The Content Acquisition System can write output from a crawl to either a Record Store instance (the default) or to a record output file. The configuration of the Deployment Template varies depending on whether a crawl is configured to write output to a Record Store instance or configured to store crawl output in record output files. This topic describes the main differences between the storage types as they affect the operations of the Deployment Template.

Note: Although both storage types are fully supported, Oracle recommends configuring a CAS crawl to write to a Record Store instance rather than a record output file. This approach simplifies both the operational model and Deployment Template configuration.

Characteristics of a Record Store instance versus record output files

- **Files** - In a Record Store instance, there are no individual files to manipulate. For crawls that write to record output files, there are one or more files to manipulate for both full and incremental crawls. You do not need to fetch files for a baseline crawl that is configured to write Endeca records to a Record Store instance. In crawls that write to a Record Store instance, a baseline pipeline uses a custom record adapter to read records directly from a Record Store instance. There is no fetching or copying record output files. For details about configuring a custom record adapter to read from a Record Store instance, see the *CAS Developer's Guide*.
- **Operational instructions** - A crawl that writes output to a Record Store instance does not require Deployment Template configuration for output destinations, file names, or require instructions to move, copy, or fetch files. A crawl that writes to record output files requires configuration output destinations, file names, and instructions to move, copy, or fetch files.
- **Configuration properties** - For crawls that write to a Record Store instance, the CAS server configuration properties in the `custom-component` need to specify the host and port of the CAS Server. No properties are required for output destinations. For crawls that write to a record output file, the CAS server configuration properties in the `custom-component` need to specify the host and port of the CAS Server, and output destinations for full and incremental crawl files.

About Deployment Template files for both storage types

This topic describes the files you modify in order to integrate and run CAS crawls as part of the Deployment Template.

Deployment Template files for CAS crawls that write to a Record Store instance

For crawls that write output to a Record Store instance, there are no additional configuration or script files. This is because the operational model interacting with a Record Store is relatively simple: there are no files to fetch, move, copy and so on. You edit `AppConfig.xml` to specify the required CAS Server host, the pipeline, and the baseline or incremental crawl that you want to run.

Deployment Template files for CAS crawls that write to record output files

For crawls that write output to record output files, the associated configuration and script files are in the `[appdir]/config/script` directory and their purpose is as follows:

- The `[appdir]/config/script/fetchCasCrawlDataConfig.xml` file is the global CAS crawl configuration for the application. The file provides two major functions. First, it provides set of global configuration settings that are used for all file system and CMS crawls, such as the location of the CAS Server. Second, it provides two scripts for fetching baseline and incremental output files (i.e., transferring the crawl output files to the source data destination directories).
- There is a `[crawlname]CasCrawlConfig.xml` document for each configured CAS crawl. For example, the sample `EndecaCasCrawlConfig.xml` is for a crawl that was created with a name of "Endeca". The document contains two scripts: one for baseline crawls and one for incremental crawls.

EAC Component API methods for CAS

In your `AppConfig.xml` code, you can coordinate CAS crawls and baseline or partial updates using the methods available in `ContentAcquisitionServerComponent`.

The more important methods available in `ContentAcquisitionServerComponent` are listed in the following table and are used in examples throughout this chapter.

Method	Description
`runBaselineCasCrawl()`	Runs a full CAS crawl to completion. This method starts the crawl and also calls `waitForFinished()` to check for completion. The method throws an exception if the crawl does not complete successfully.
`startBaselineCasCrawl()`	Starts a baseline crawl but does not check for completion. If the crawl is not already provisioned, or if the CAS can't be reached, this method throws an exception.
`runIncrementalCasCrawl()`	Runs an incremental crawl to completion. This method starts the crawl and calls `waitForFinished()` to check for completion. The method throws an exception if the crawl does not complete successfully.
`startIncrementalCasCrawl()`	Starts an incremental crawl but does not check for completion. If the crawl is not already provisioned, or if the CAS Service can't be reached, this method throws an exception.

Method	Description
`waitForFinished()`	Waits for the specified crawl to finish (to be in a non-active state) or for the maximum amount of wait time to elapse. (The wait time is specified by the `maxCasCrawlWaitSeconds` property. The default is -1 which indicates the script should wait indefinately until the CAS crawl completes.) While waiting, the method thread sleeps, swallowing any `InterruptedExceptions` that are caught.
`getCrawlStatus()`	Retrieves the status of the specified crawl.
`isCrawlActive()`	Test whether the crawl is active (running or stopping).
`exportDimensionValueIdMappings()`	Exports dimension value Id mappings from a Dimension Value Id Manager to a CSV file at the path specified on the CAS Server.
`importDimensionValueIdMappings()`	Imports dimension value Id mappings from a CSV file located at the specified path to a specified Dimension Value Id Manager.

For additional details about `ContentAcquisitionServerComponent`, see the Javadoc installed in `CAS\<version>\doc\cas-dt-javadoc`.

Integrating and running CAS crawls that write to Record Store instances

This section describes the high-level steps for integrating and running a CAS crawl that writes output to a Record Store instance.

This section assumes you have done the following:

- Installed and started the CAS Service and CAS Console.
- Installed an EAC Agent and started it on the server running the CAS Service.
- Deployed an application using the template.
- Configured an application by editing the `AppConfig.xml`.

The steps below are described in their own topics.

1. Create a CAS crawl.
2. Specify a CAS Server host in `AppConfig.xml`.
3. Specify a CAS Server as a custom component.
4. Specify a pipeline to run in `AppConfig.xml`.
5. Add code to run a CAS crawl (full or incremental).
6. Run a CAS crawl (full or incremental).
7. Coordinate CAS crawls and baseline or partial updates.

Related Links

Deploying an EAC application on UNIX on page 15
This section describes the steps for deploying an EAC application in a UNIX environment using the provided `deploy.sh` file.

Deploying an EAC application on Windows on page 13
This section describes the steps for deploying an EAC application in a Windows environment using the provided `deploy.bat` file.

Configuring an application on page 21
This section guides you through the process of configuring the deployment to run your application.

Creating a CAS crawl

Create and configure a CAS crawl using either the Oracle Endeca CAS Console, the CAS Server API, or the CAS Server Command-line Utility. For details about creating and configuring a CAS crawl, see the *Oracle Endeca CAS Developer's Guide* or the *Oracle Endeca CAS API Guide*.

Specifying a CAS Server as a custom component for Record Store output

A `custom-component` element defines the configuration properties of a specific CAS Server. This topic describes the configuration properties for a CAS Server that writes output to a Record Store instance.

Note: The configuration examples below show the CAS 3.0.x deployment template component name (i.e. the `class` attribute) rather than the CAS 2.2.x deployment template component.

Note: The Deployment Template checks the host and port definition in `AppConfig.xml`, *Name*`CasCrawlConfig.xml`, and `fetchCasCrawlDataConfig.xml`. If host or port information conflicts in any of these three files, errors will occur. Either make sure the host and port configuration is same in the three files, or comment out host and port configuration in both *Name*`CasCrawlConfig.xml` and `fetchCasCrawlDataConfig.xml` and put the configuration in `AppConfig.xml`.

To specify a CAS Server as a custom component for Record Store output:

1. Open `fetchCasCrawlDataConfig.xml` in a text editor and cut out the custom component for the Content Acquisition System Server configuration. (This prevents configuration conflicts as described in the note above.)
   For example, remove the following from `fetchCasCrawlDataConfig.xml`:

```
<!--

##########################################################################

    # Content Acquisition System Server
    #

  <custom-component id="CAS" host-id="CASHost" class="com.endeca.eac.toolk¬
it.component.cas.ContentAcquisitionServerComponent">
    <properties>
      <property name="casHost" value="localhost" />
      <property name="casPort" value="8500" />
      <property name="casCrawlFullOutputDestDir" value="./data/com¬
plete_cas_crawl_output/full" />
     <property name="casCrawlIncrementalOutputDestDir" value="./data/com¬
plete_cas_crawl_output/incremental" />
      <property name="casCrawlOutputDestHost" value="CASHost" />
    </properties>
  </custom-component>

  -->
```

2. Open the `AppConfig.xml` file in a text editor and paste in the custom component for the Content Acquisition System Server configuration.
3. In the Content Acquisition System Server configuration, remove the properties for directories and keep the properties that specify the Record Store instance host and port. Ensure that the following attributes are set correctly:
   - An `id` attribute that assigns a unique ID to a specific CAS Server. (The example in this documentation use `CAS` for the `id`.)
   - The `host-id` attribute points back to the id attribute of the `host` global configuration element.
   - The `class` attribute specifies the class that implements the CAS deployment template component. If you are using CAS 2.2.x, specify `class="com.endeca.soleng.eac.toolkit.component.ContentAcquisitionServerComponent"`. If you are using CAS 3.0.x, specify `class="com.endeca.eac.toolkit.component.cas.ContentAcquisitionServerComponent"`.
   - A `casHost` property to indicate the CAS Server which manages the crawls.
   - A `casPort` property to indicate the port on which the CAS Server is listening. This is the port number you specified when you installed CAS.

   For example:

```
<!--

################################################################################

    # Content Acquisition System Server
    #

  <custom-component id="CAS" host-id="CASHost" class="com.endeca.eac.toolkit.component.cas.ContentAcquisitionServerComponent">
    <properties>
      <property name="casHost" value="localhost" />
      <property name="casPort" value="8500" />
    </properties>
  </custom-component>

  -->
```

Specifying a pipeline to run in AppConfig.xml (for Record Store output)

The `devStudioConfigDir` attribute in the `ConfigManager` custom component specifies the Forge pipeline to run. By default, the Deployment Template checks the `[appdir]/config/pipeline` for the Forge pipeline to run. This includes baseline updates and partial updates. It is simplest to put your pipeline files in this directory.

You specify the Forge pipeline that you configured to read from one or more Record Store instances. (For details about creating a Forge pipeline, see the *Oracle Endeca CAS Developer's Guide*.)

To specify a Forge pipeline to run in `AppConfig.xml`:

1. Ensure that your Forge pipeline files are located in `[appdir]/config/pipeline`.
2. Alternatively, modify the `devStudioConfigDir` property in the `ConfigManager` custom component to reference the pipeline directory. Also, modify the `configDir` property in the Partial update Forge section to reference the pipeline directory.

In this example, the Forge pipeline is stored in the `pipeline` directory:

```
<custom-component id="ConfigManager" host-id="ITLHost"
  class="com.Endeca.soleng.eac.toolkit.component.ConfigManagerComponent">

  <properties>
  ...
  </properties>
  <directories>
    <directory
      name="devStudioConfigDir">./config/pipeline
    </directory>
    ...
  </directories>
```

Add code to run a CAS crawl

Add code to `AppConfig.xml` that specifies the CAS crawl to run. Depending on your environment, you may need a script that runs a full CAS crawl and a script that runs an incremental CAS crawl. In either case, the code typically locks the crawl (to wait for any running crawls to complete), runs the crawl, and release the lock.

To add code to run a CAS crawl:

1. Open the `AppConfig.xml` file in a text editor.
2. To run a full CAS crawl, add code that locks the crawl, runs the crawl, and releases the lock. For example:

```
<!--

###############################################################################

    # full crawl script
    #
 -->

  <script id="Endeca_fullCasCrawl">
    <log-dir>./logs/provisioned_scripts</log-dir>
    <provisioned-script-command>./control/runcommand.bat Endeca_full¬
CasCrawl run</provisioned-script-command>
    <bean-shell-script>
      <![CDATA[
    crawlName = "Endeca";

    log.info("Starting full CAS crawl '" + crawlName + "'.");

    // obtain lock
    if (LockManager.acquireLock("crawl_lock_" + crawlName)) {

      CAS.runBaselineCasCrawl(crawlName);

      LockManager.releaseLock("crawl_lock_" + crawlName);
    }
    else {
      log.warning("Failed to obtain lock.");
    }

    log.info("Finished full CAS crawl '" + crawlName + "'.");
      ]]>
```

```
    </bean-shell-script>
  </script>
```

3. To run a incremental CAS crawl, add code that locks the crawl, runs the crawl, and releases the lock.
   For example:

```
   <!--

##########################################################################

    # incremental crawl script
    #
  -->
  <script id="Endeca_incrementalCasCrawl">
    <log-dir>./logs/provisioned_scripts</log-dir>
   <provisioned-script-command>./control/runcommand.bat Endeca_incremen¬
talCasCrawl run</provisioned-script-command>
    <bean-shell-script>
      <![CDATA[
    crawlName = "Endeca";

    log.info("Starting incremental CAS crawl '" + crawlName + "'.");

    // obtain lock
    if (LockManager.acquireLock("crawl_lock_" + crawlName)) {

      CAS.runIncrementalCasCrawl(crawlName);

      LockManager.releaseLock("crawl_lock_" + crawlName);
    }
    else {
      log.warning("Failed to obtain lock.");
    }

    log.info("Finished incremental CAS crawl '" + crawlName + "'.");
      ]]>
    </bean-shell-script>
  </script>
```

Running a CAS crawl

You can run either a full or incremental CAS crawl script to produce records in the Record Store instance.

Alternatively, you can run the CAS crawl as part of a baseline or partial update as described in *Coordinating CAS crawls and baseline or partial updates* on page 84.

To run a CAS crawl script:

1. To run a full crawl, run the full crawl script you specified in `AppConfig.xml` and specify the name of the crawl.
   For example:

```
C:\Endeca\apps\DocApp\control>runcommand.bat Endeca_fullCasCrawl
```

2. To run an incremental crawl, run the incremental crawl script you specified in `AppConfig.xml` and specify the name of the crawl.

For example:

```
C:\Endeca\apps\DocApp\control>runcommand.bat Endeca_incrementalCasCrawl
```

Coordinating CAS crawls and baseline or partial updates

After running a CAS crawl, you run a baseline or partial update that incorporates the records from a Record Store instance. How you configure the coordination between your CAS crawls and baseline or partial updates depends upon how complicated your environment is. This topic describes several scenarios.

A single CAS crawl and then a Forge update

There is a simple case where the Deployment Template runs a *full* CAS crawl and then runs a *baseline* update that incorporates the records from the crawl. To create this sequential workflow of CAS crawl and then baseline update, you can doc the following:

- Remove the default `Forge.isDataReady` check from the baseline update script. This call checks whether a flag was set that indicates the incoming data files are ready for Forge. Removing this call means that the lock manager does not check the flag or wait on the flag to be cleared before running a CAS crawl.
- add a call to `runBaselineCasCrawl()` to run the full CAS crawl.

For example, this baseline update script calls `CAS.runBaselineCasCrawl("MyCrawl")` which runs a full CAS crawl named `MyCrawl`. Then the script continues with baseline update processing.

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

    # Baseline update script
    #
  -->
  <script id="BaselineUpdate">
    <log-dir>./logs/provisioned_scripts</log-dir>
   <provisioned-script-command>./control/baseline_update.bat</provisioned-
script-command>
    <bean-shell-script>
      <![CDATA[
    log.info("Starting baseline update script.");
    // obtain lock
    if (LockManager.acquireLock("update_lock")) {

        // call the baseline crawl script to run a full CAS
        // crawl.
        CAS.runBaselineCasCrawl("MyCrawl");

        if (ConfigManager.isWebStudioEnabled()) {
          // get Web Studio config, merge with Dev Studio config
          ConfigManager.downloadWsConfig();
          ConfigManager.fetchMergedConfig();
        } else {
          ConfigManager.fetchDsConfig();
        }

        // clean directories
        Forge.cleanDirs();
        PartialForge.cleanCumulativePartials();
        Dgidx.cleanDirs();
```

```
      // fetch extracted data files to forge input
      Forge.getIncomingData();
      LockManager.removeFlag("baseline_data_ready");

      // fetch config files to forge input
      Forge.getConfig();

      // archive logs and run ITL
      Forge.archiveLogDir();
      Forge.run();
      Dgidx.archiveLogDir();
      Dgidx.run();

      // distributed index, update Dgraphs
      DistributeIndexAndApply.run();

      // if Web Studio is integrated, update Web Studio with latest
      // dimension values
      if (ConfigManager.isWebStudioEnabled()) {
        ConfigManager.cleanDirs();
        Forge.getPostForgeDimensions();
        ConfigManager.updateWsDimensions();
      }

      // archive state files, index
      Forge.archiveState();
      Dgidx.archiveIndex();

      // (start or) cycle the LogServer
      LogServer.cycle();

    // release lock
    LockManager.releaseLock("update_lock");
    log.info("Baseline update script finished.");
    } else {
         log.warning("Failed to obtain lock.");
    }
    ]]>
  </bean-shell-script>
</script>
```

You run the baseline update by running `baseline_update` in the `apps/[appDir]/control` directory.

For example:

```
C:\Endeca\apps\DocApp\control>baseline_update.bat
```

There is also the very similar case where the Deployment Template runs an *incremental* CAS crawl and then runs a *partial* update that incorporates the records from the crawl. To create this sequential workflow of incremental CAS crawl and then partial update, you can do the following:

- Remove the default `PartialForge.isPartialDataReady` check from the partial update script. This call checks whether a flag was set that indicates the incoming data files are ready for Forge. Removing this call means that the lock manager does not check the flag or wait on the flag to be cleared before running a CAS crawl.
- Add a call `runIncrementalCasCrawl()` to run the incremental CAS crawl.
- Remove the call to `PartialForge.getPartialIncomingData()` that fetches extracted data files.

For example, this partial update script calls `CAS.runIncrementalCasCrawl("MyCrawl")` which runs an incremental CAS crawl named `MyCrawl`. Then the script continues with partial update processing.

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

    # Partial update script
    #
  -->
  <script id="PartialUpdate">
    <log-dir>./logs/provisioned_scripts</log-dir>
    <provisioned-script-command>./control/partial_update.bat</provisioned-
script-command>
    <bean-shell-script>
      <![CDATA[
    log.info("Starting partial update script.");

    // obtain lock
    if (LockManager.acquireLock("update_lock")) {

       // call the partial crawl script to run an incremental
       // CAS crawl.
       CAS.runIncrementalCasCrawl("MyCrawl");

       // archive logs
        PartialForge.archiveLogDir();

        // clean directories
        PartialForge.cleanDirs();

        // fetch config files to forge input
        PartialForge.getConfig();

        // run ITL
        PartialForge.run();

        // timestamp partial, save to cumulative partials dir
        PartialForge.timestampPartials();
        PartialForge.fetchPartialsToCumulativeDir();

        // distribute partial update, update Dgraphs
        DgraphCluster.cleanLocalPartialsDirs();
        DgraphCluster.copyPartialUpdateToDgraphServers();
        DgraphCluster.applyPartialUpdates();

        // archive partials
        PartialForge.archiveCumulativePartials();

      // release lock
      LockManager.releaseLock("update_lock");
      log.info("Partial update script finished.");
    } else {
      log.warning("Failed to obtain lock.");
    }
      ]]>
    </bean-shell-script>
  </script>
```

You run the partial update by running `partial_update` in the `apps/[appDir]/control` directory.

For example:

```
C:\Endeca\apps\DocApp\control>partial_update.bat
```

Multiple CAS crawls and multiple Forge updates

There is a more complicated case where multiple CAS crawls are running on their own schedules, and updates are running on their own schedules. To coordinate this asynchronous workflow of CAS crawls and baseline or partial updates, you add code that calls methods in `ContentAcquisition¬ ServerComponent`.

For details about `ContentAcquisitionServerComponent`, see *EAC Component API Reference for CAS Server (Javadoc)* installed in `<Endeca installation path>\CAS\<version>\doc\cas-dt-javadoc`.

In your `AppConfig.xml` code, the main coordination task is one of determining how you time running CAS crawls and how you time running baseline or partial updates that consume records from those crawls. For example, suppose you have an application that runs three full CAS crawls and those records are consumed by a single baseline update. In that scenario, each of the three full crawls has its own full crawl script in `AppConfig.xml` that runs on a nightly schedule. And the `AppConfig.xml` file contains a baseline update that runs nightly to consume the latest generation of records from each of the three crawls. The `Forge.isDataReady` check is not required in the baseline update script because the source data is not locked.

Here is an example script for one of the full CAS crawls named `endeca`.

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

    # full crawl script
    #
 -->

  <script id="endeca_fullCasCrawldoc">
    <log-dir>./logs/provisioned_scripts</log-dir>
   <provisioned-script-command>./control/runcommand.bat endeca_fullCasCrawl¬
doc</provisioned-script-command>
    <bean-shell-script>
      <![CDATA[
    crawlName = "endeca";

    log.info("Starting full CAS crawl '" + crawlName + "'.");

    // obtain lock
    if (LockManager.acquireLock("crawl_lock_" + crawlName)) {

      CAS.runBaselineCasCrawl(crawlName);

      LockManager.releaseLock("crawl_lock_" + crawlName);
    }
    else {
      log.warning("Failed to obtain lock.");
    }

    log.info("Finished full CAS crawl '" + crawlName + "'.");
      ]]>
    </bean-shell-script>
  </script>
```

Here is an example script for a Forge baseline update that processes records from the three CAS crawls mentioned above (The record adapter configuration in the Developer Studio project specifies that Forge reads from multiple Record Store instances).

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

    # Baseline update script
    #
  -->
  <script id="BaselineUpdate">
    <log-dir>./logs/provisioned_scripts</log-dir>
   <provisioned-script-command>./control/baseline_update.bat</provisioned-
script-command>
    <bean-shell-script>
      <![CDATA[
    log.info("Starting baseline update script.");
    // obtain lock
    if (LockManager.acquireLock("update_lock")) {

        if (ConfigManager.isWebStudioEnabled()) {
          // get Web Studio config, merge with Dev Studio config
          ConfigManager.downloadWsConfig();
          ConfigManager.fetchMergedConfig();
        } else {
          ConfigManager.fetchDsConfig();
        }

        // clean directories
        Forge.cleanDirs();
        PartialForge.cleanCumulativePartials();
        Dgidx.cleanDirs();

        // fetch extracted data files to forge input
        Forge.getIncomingData();
        LockManager.removeFlag("baseline_data_ready");

        // fetch config files to forge input
        Forge.getConfig();

        // archive logs and run ITL
        Forge.archiveLogDir();
        Forge.run();
        Dgidx.archiveLogDir();
        Dgidx.run();

        // distributed index, update Dgraphs
        DistributeIndexAndApply.run();

        // if Web Studio is integrated, update Web Studio with latest
        // dimension values
        if (ConfigManager.isWebStudioEnabled()) {
          ConfigManager.cleanDirs();
          Forge.getPostForgeDimensions();
          ConfigManager.updateWsDimensions();
        }

        // archive state files, index
        Forge.archiveState();
        Dgidx.archiveIndex();
```

```
        // (start or) cycle the LogServer
        LogServer.cycle();

      // release lock
      LockManager.releaseLock("update_lock");
      log.info("Baseline update script finished.");
      } else {
            log.warning("Failed to obtain lock.");
      }
      ]]>
    </bean-shell-script>
  </script>
```

Integrating and running CAS crawls that write to record output files

This section describes the high-level steps for integrating and running a CAS crawl that writes output to a record output file.

This section assumes you have done the following:

- Installed and started the CAS Service and CAS Console.
- Installed an EAC Agent and started it on the server running the CAS Service.
- Deployed an application using the template.
- Configured an application by editing the `AppConfig.xml`.

The steps below are described in their own topics.

1. Create a CAS crawl.
2. Specify a CAS Server host in `AppConfig.xml`.
3. Specify a CAS Server as a custom component for any CAS crawl that writes to record output files.
4. Specify a pipeline to run in `AppConfig.xml`.
5. Edit `fetchCasCrawlDataConfig.xml` to reflect the details of your crawling environment.
6. Create a CAS crawl script, for the crawl you created in step 1, by running the `[appdir]/control/cas/make_cas_crawl_scripts` script.
7. Run a baseline CAS crawl (a full crawl) using the sample CAS crawl pipeline. If the baseline update runs without failing, you can start to make further modifications to your deployment, such as using your custom pipeline.
8. Optionally, run an incremental CAS crawl. These steps verify that your configuration files are correct.
9. Load the crawl files generated in the previous step to be processed by the sample CAS crawl pipeline.
10. Run a baseline update using the sample CAS crawl pipeline with the new crawl record output files.

Note: The instructions provided in this section apply to the Dgraph deployment type. If your are using an Agraph, all of the file system crawl integration components are deployed and work the same way, but you need to customize the `cas_crawl_pipeline` (or create your own pipeline) to process the crawl data into an Agraph. Essentially, the crawl functionality works exactly the same way, but the Deployment Template does not provide a sample pipeline for the Agraph case.

Related Links

Deploying an EAC application on UNIX on page 15

This section describes the steps for deploying an EAC application in a UNIX environment using the provided `deploy.sh` file.

Deploying an EAC application on Windows on page 13
This section describes the steps for deploying an EAC application in a Windows environment using the provided `deploy.bat` file.

Configuring an application on page 21
This section guides you through the process of configuring the deployment to run your application.

Creating a CAS crawl

Create and configure a CAS crawl using either the Oracle Endeca CAS Console, the CAS Server API, or the CAS Server Command-line Utility. For details about creating and configuring a CAS crawl, see the *Oracle Endeca CAS Developer's Guide* or the *Oracle Endeca CAS API Guide*.

Specifying a CAS Server host

The `host` element defines the specifics of the machine on which the CAS crawls are deployed. The `host` configuration is the same for crawls that write output to a Record Store instance and crawls that write to a record output file.

The `hostName` is set automatically when you run the deployment script. You therefore do not typically have to change the hostname unless you decide to start using a CAS Server on another machine.

Note: The Deployment Template checks the host and port definition in `AppConfig.xml`, *Name*`CasCrawlConfig.xml`, and `fetchCasCrawlDataConfig.xml`. If host or port information conflicts in any of these three files, errors will occur. Either make sure the host and port configuration is same in the three files, or comment out host and port configuration in both *Name*`CasCrawlConfig.xml` and `fetchCasCrawlDataConfig.xml` and put the configuration in `AppConfig.xml`.

To specify a CAS Server host:

1. Open the `AppConfig.xml` file in a text editor.
2. Locate the `Servers/hosts` section and do the following:
   - Add a `host` element and specify an `id` attribute for the machine running the CAS Server.
   - Specify a `hostname` attribute for the machine name running the EAC Agent.
   - Specify a `port` attribute specifies the port on which the EAC Agent is listening. This is the Endeca HTTP Service port on that server (`8888` in the example).

   For example:

```
<!–
##############################################################################
# CAS servers/hosts
-->
<host id="CASHost" hostName="WEB009.mycompany.com" port="8888" />
```

Specifying a CAS Server as a custom component for record output files

The `custom-component` section defines the configuration properties of a specific CAS Server. This topic describes the configuration properties for a CAS crawl that writes output to a record output file.

Note: The configuration example below shows the CAS 3.0.x deployment template component name (i.e. the `class` attribute) rather than the CAS 2.2.x deployment template component.

To specify a CAS Server as a custom component for record output files:

Add custom-component with the following configuration:

- An `id` attribute that assigns a unique ID to a specific CAS Server.
- The `host-id` attribute points back to the id attribute of the `host` global configuration element.
- The `class` attribute specifies the class that implements the CAS deployment template component. If you are using CAS 2.2.x, specify `class="com.endeca.soleng.eac.toolk¬it.component.ContentAcquisitionServerComponent"`. If you are using CAS 3.0.x, specify `class="com.endeca.eac.toolkit.component.cas.ContentAcquisition¬ServerComponent"`.
- A `casHost` property to indicate the CAS Service which manages the file system and CMS crawls.
- A `casPort` property to indicate the port on which the CAS Service is listening. The port number must match the `com.endeca.cas.port` value that is used in the cas-server startup script. The cas-server startup configuration is in `Endeca\CAS\workspace\conf\jetty.xml`. Be sure to change the port number if it is not the same as the `com.endeca.cas.port` value.
- A `casCrawlFullOutputDestDir` property to indicate the destination directory to which the crawl output file will be copied after a baseline crawl. Note that this is not the directory to which the CAS crawl writes its output; that output directory is set as part of the crawl configuration.
- A `casCrawlIncrementalOutputDestDir` property to indicate the destination directory to which the crawl output file will be copied after an incremental crawl. As with the previous property, this is not the directory to which the CAS crawl writes its output. If you run incremental crawls, the default settings assume that the output format will be compressed binary files.
- A `casCrawlOutputDestHost` property to indicate the ID of the host on which the destination directories (specified by the previous two properties) reside.

For example:

```
<!—
#############################################################################
# Content Acquisition System Server
#
-->
<custom-component id="CAS" host-id="CASHost" class="com.endeca.eac.toolk¬
it.component.cas.ContentAcquisitionServerComponent">
  <properties>
    <property name="casHost" value="WEB009.mycompany.com" />
    <property name="casPort" value="8500" />
    <property name="casCrawlFullOutputDestDir"
       value="./data/complete_cas_crawl_output/full" />
    <property name="casCrawlIncrementalOutputDestDir"
       value="./data/complete_cas_crawl_output/incremental" />
    <property name="casCrawlOutputDestHost" value="CASHost" />
  </properties>
< /custom-component>
```

Specifying a pipeline to run in `AppConfig.xml` (for record output files)

The `devStudioConfigDir` attribute in the `ConfigManager` custom component specifies the pipeline to run. This topic describes the how to specify a pipeline for record output file configurations. Sample scripts are provided for record output file configuration.

To specify a pipeline to run in `AppConfig.xml`:

1. Edit the value of the `devStudioConfigDir` attribute in the `ConfigManager` custom component to reference the `config/cas_crawl_pipeline` directory.
   For example:

```
<custom-component id="ConfigManager" host-id="ITLHost"
  class="com.Endeca.soleng.eac.toolkit.component.ConfigManagerComponent">

  <properties>
  ...
  </properties>
  <directories>
    <directory
      name="devStudioConfigDir">./config/cas_crawl_pipeline
    </directory>
    ...
  </directories>
```

2. Edit the value of the `configDir` attribute in the Partial update Forge section to reference the `config/cas_crawl_pipeline` directory.
   For example:

```
<!—
#############################################################################
# Partial update Forge
-->
<forge id="PartialForge" host-id="ITLHost">
  <properties>
  ...
  </properties>
  <directories>
    ...
    <directory name="configDir">./config/cas_crawl_pipeline</directory>
    ...
  </directories>
```

Editing `fetchCasCrawlDataConfig.xml` for your crawling environment

The `[appdir]/config/script/fetchCasCrawlDataConfig.xml` file is the global CAS crawl configuration for the application.

Note: The configuration examples below show the CAS 3.0.x deployment template component name (i.e. the `class` attribute) rather than the CAS 2.2.x deployment template component.

To edit `fetchCasCrawlDataConfig.xml` for your crawling environment:

Check the settings of the following sections:

- CAS Server port: The custom-component section defines the configuration properties of a specific CAS Server. In particular, check the port number specified in the `casPort` property,

and check the `class` attribute. It specifies the class that implements the CAS deployment template component. If you are using CAS 2.2.x, specify `class="com.Ende¬ca.soleng.eac.toolkit.component.ContentAcquisitionServerComponent"`. If you are using CAS 3.0.x, specify `class="com.Endeca.eac.toolkit.component.cas.Con¬tentAcquisitionServerComponent"` :

```
<custom-component id="CAS" host-id="CASHost" class="com.Oracle Ende¬
ca.eac.toolkit.component.cas.ContentAcquisitionServerComponent">
  <properties>
    <property name="casHost" value="WEB009.mycompany.com" />
    <property name="casPort" value="8500" />
    <property name="casCrawlFullOutputDestDir"
       value="./data/complete_cas_crawl_output/full" />
    <property name="casCrawlIncrementalOutputDestDir"
       value="./data/complete_cas_crawl_output/incremental" />
    <property name="casCrawlOutputDestHost" value="CASHost" />
  </properties>
< /custom-component>
```

If the port on which the CAS Service was started is not the default `8500` number, change the value of the property.

- Incremental output definition: If you run incremental crawls, the default settings assume that the output format are compressed binary files. If your crawls are configured to use other output formats (uncompressed binary, compressed XML, or uncompressed XML), you must make the following changes.

  For an XML output format (either compressed or uncompressed), locate the `<script id="fetchIncrementalCasCrawlData">` section of the script and change the filename extension of the placeholder file from "`placeholder.bin`" to "`placeholder.xml`", so that the code looks like this:

  ```
  if (! fileUtil.dirContainsFiles(incrDestDir, Forge.getHostId())) {
    placeholder = incrDestDir + "/placeholder.xml";
  ```

  For uncompressed formats (either binary or XML), remove or comment out these statements that run a shell script to compress the placeholder file:

  ```
  shell.init("zip_incremental_cas_crawl_placeholder",
    Forge.getHostId(), zipCmd, Forge.getWorkingDir());
  shell.run();
  ```

Creating a CAS crawl script using `make_cas_crawl_scripts`

After you have created a CAS crawl, use the `[appdir]/control/cas/make_cas_crawl_scripts` script to create a CAS crawl script for your deployment.

To create a CAS crawl script:

Run the `[appdir]/control/cas/make_cas_crawl_scripts` script and specify the name of the file system or CMS crawl you created previously.
In this Windows example, "MyCrawl" is the name of the file system or CMS crawl you have created and you are currently in the `[appdir]/control/cas` directory.

```
C:\Endeca\Apps\WineApp\control\cas>make_cas_crawl_scripts MyCrawl
```

As a result, the script creates a configuration file (named `MyCrawlCasCrawlConfig.xml`) in the `[appdir]/config/script` directory. Note that the script does not need to be customized, so it can be used as-is.

Running a baseline or incremental CAS crawl to record output files

Use one of the scripts located in the `[appdir]/control/cas` directory to run a CAS crawl.

To run a baseline or an incremental CAS crawl:

1. To run a baseline, run the `baseline_cas_crawl` script and specify the name of the crawl to run.

   Note that a baseline crawl will remove any existing incremental crawl output files. After running a baseline crawl, you can run multiple incremental crawls, because an incremental crawl will not remove any previous incremental data.
2. To run an incremental crawl, run the `incremental_cas_crawl` script and specify the name of the crawl to run.

Note: Note that you cannot specify multiple crawl names. To run multiple crawls, you will need to invoke the command multiple times. Multiple crawls can be run simultaneously, sequentially, or completely independently with this approach.

When a baseline crawl runs, the following happens:

1. The script deletes the previous baseline and incremental output files from the crawl output directory configured for the CAS crawl.
2. The CAS Server first writes the output file to the output directory configured for the crawl. The filename uses the naming convention specified for the crawl (for example, `CrawlerOutput-FULL-sgmt000.bin.gz`).
3. The script prepends the crawl name to the output filename (for example, the name becomes `MyCrawl_CrawlerOutput-FULL-sgmt000.bin.gz`).
4. The script deletes the previous baseline and incremental output files (i.e., the files in the `casCrawlFullOutputDestDir` and `casCrawlIncrementalOutputDestDir` directories).
5. The script copies the baseline output file to the directory specified by the `casCrawlFullOutput¬DestDir` property.

When an incremental crawl runs, the following happens:

1. The script deletes the previous baseline and incremental output files from the crawl output directory configured for the CAS crawl.
2. The CAS Server first writes the output file to the output directory configured for the crawl. The filename uses the naming convention specified in the crawl configuration (for example, `CrawlerOutput-INCR-sgmt000.bin.gz`).
3. The script prepends both the crawl name and a timestamp to the output filename (for example, `MyCrawl_2008.02.26.04.49.39_CrawlerOutput-INCR-sgmt000.bin.gz`).
4. The script copies the incremental output file to the directory specified by the `casCrawlIncremen¬talOutputDestDir` property.

Note: Because the incremental output files are timestamped, previous incrementals are not deleted.

The following example shows a baseline crawl being run on a Windows machine:

```
C:\Endeca\Apps\WineApp\control\cas>baseline_cas_crawl MyCrawl
[07.17.08 14:42:40] INFO: Checking definition from AppConfig.xml,
MyCrawlCasCrawlConfig.xml, fetchCasCrawlDataConfig.xml against
existing EAC provisioning.
[07.17.08 14:42:47] INFO: Setting definition for host 'CASHost'.
[07.17.08 14:43:17] INFO: Setting definition for script
'MyCrawl_baselineCasCrawl'.
[07.17.08 14:43:18] INFO: Setting definition for script
'MyCrawl_incrementalCasFileSystemCrawl'.
[07.17.08 14:43:20] INFO: Setting definition for custom component 'CAS'.
[07.17.08 14:43:20] INFO: Updating provisioning for host 'CASHost'.
[07.17.08 14:43:20] INFO: Updating definition for host 'CASHost'.
[07.17.08 14:43:22] INFO: [CASHost] Starting shell utility
'mkpath_-data-complete-cas-crawl-output-incremental'.
[07.17.08 14:43:27] INFO: [CASHost] Starting shell utility
'mkpath_-data-complete-cas-crawl-output-full'.
[07.17.08 14:43:30] INFO: Setting definition for script
'fetchFullCasCrawlData'.
[07.17.08 14:43:32] INFO: Setting definition for script
'fetchIncrementalCasCrawlData'.
[07.17.08 14:43:32] INFO: Definition updated.
[07.17.08 14:43:33] INFO: Starting full CAS crawl 'MyCrawl'.
[07.17.08 14:43:33] INFO: Acquired lock 'crawl_lock_MyCrawl'.
[07.17.08 14:43:33] INFO: Starting baseline CAS crawl with id 'MyCrawl'.
[07.17.08 14:45:05] INFO: Acquired lock
'complete_cas_crawl_data_lock'.
[07.17.08 14:45:06] INFO: [CASHost] Starting copy utility
'copy_MyCrawl_crawl_output_to_dest_dir'.
[07.17.08 14:45:07] INFO: Released lock
'complete_cas_crawl_data_lock'.
[07.17.08 14:45:08] INFO: Released lock 'crawl_lock_MyCrawl'.
[07.17.08 14:45:08] INFO: Finished full CAS crawl 'MyCrawl'.
```

Loading crawl record output files for use in the sample CAS pipeline

To move the crawl output files to the appropriate incoming directory, use one of the scripts located in the `[appdir]/control/cas` directory.

To load crawl record output files for use in the sample CAS pipeline:

1. To load baseline crawl data to the `incoming/full` directory and also copy any incremental data to the `incoming/incremental` directory, run the `load_full_cas_crawl_data` script.

   The record adapters in the pipeline look in these directories for the data files. Use this script if you are running baseline updates or delta updates.

2. To load incremental crawl data to the `data/partials/incoming` directory, run the `load_incremental_cas_crawl_data` script.

   This script is used for partial updates.

Note that by default, the `load_full_cas_crawl_data` script renames the incremental data files when they are copied to the `incoming/incremental` directory. For example, this file in the `data/complete_cas_crawl_output/incremental` directory:

```
MyCrawl_2008.07.18.02.46.15_CrawlerOutput-INCR-sgmt000.bin.gz
```

will be renamed to this when copied to the `incoming/incremental` directory:

```
000001_MyCrawl_2008.07.18.02.46.15_CrawlerOutput-INCR-sgmt000.bin.gz
```

The reason is that the delta update pipeline must keep the most up-to-date copy of any incremental record. To make sure this happens, the incremental crawl files must be read in reverse chronological order (i.e., the file order must be reversed) so that the first copy of each record read (which is the only record the pipeline keeps) is the most recent copy.

This reordering is not required for a partial update pipeline, which reads the incremental files in chronological order and creates updates that will be progressed by the Dgraph in the same order (effectively applying the most recent updates). Therefore, incremental files being copied to the `data/partials/incoming` directory are not renamed.

Running the sample CAS pipeline using the CAS crawl record output files

The Deployment Template includes a sample CAS crawl pipeline that is located in the `[appdir]/config/cas_crawl_pipeline` directory.

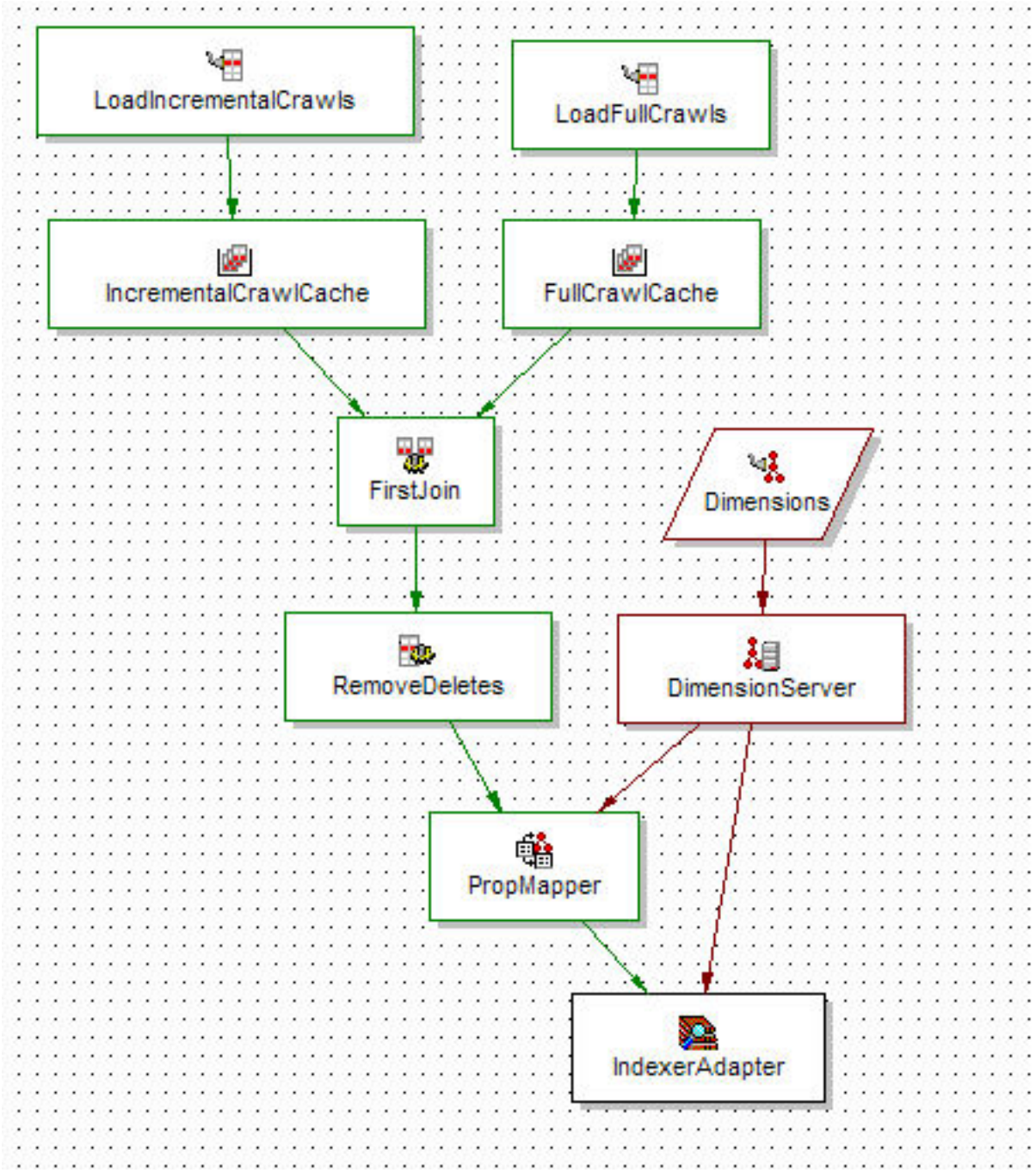
Make sure that you have updated the `AppConfig.xml` file to use the sample CAS crawl pipeline located in `[appdir]/config/cas_crawl_pipeline`. Specifically, you will need to update the `devStudioConfigDir` property in the `ConfigManager` section of the document.

Also, make sure you revise both `pipeline.epx` and `partial_pipeline.epx` so that the record adapter reads records in the format that the CAS crawl output. The format can be either XML or binary and the URL indicates whether the format is compressed or uncompressed. See the `FORMAT` and `URL` attributes in the following XML. For example, the following record adapter reads compressed XML:

```
<RECORD_ADAPTER COL_DELIMITER="" COMPRESSION_LEVEL="1" DIRECTION="INPUT"
FILTER_EMPTY_PROPS="TRUE" FORMAT="XML" FRC_PVAL_IDX="FALSE"
MULTI="TRUE" NAME="LoadFullCrawls" PREFIX="" REC_DELIMITER="" REQUIRE_DA¬
TA="FALSE" ROW_DELIMITER="" STATE="FALSE" URL="./full/*.xml.gz">
<COMMENT></COMMENT>
</RECORD_ADAPTER>
```

The pipeline is configured for delta updates, so that it reads in a baseline crawl data file (or multiple baseline crawl data files, if you create and run multiple crawls in the CAS Server) and one or more incremental crawl data files.

The pipeline looks as follows:



As mentioned above, the `LoadFullCrawls` record adapter reads in the baseline crawl data while the `LoadIncrementalCrawls` record adapter reads in all the incremental data. A record assembler performs a First Record join on all the data sets, while the `RemoveDeletes` record manipulator removes any record whose `Endeca.Action` property has a value of "`DELETE`".

To run the sample CAS pipeline using the CAS crawl record output files:

Run the `baseline_update` script located in `[appdir]/control`.

The `baseline_update` script displays the following informational message when the process is complete:

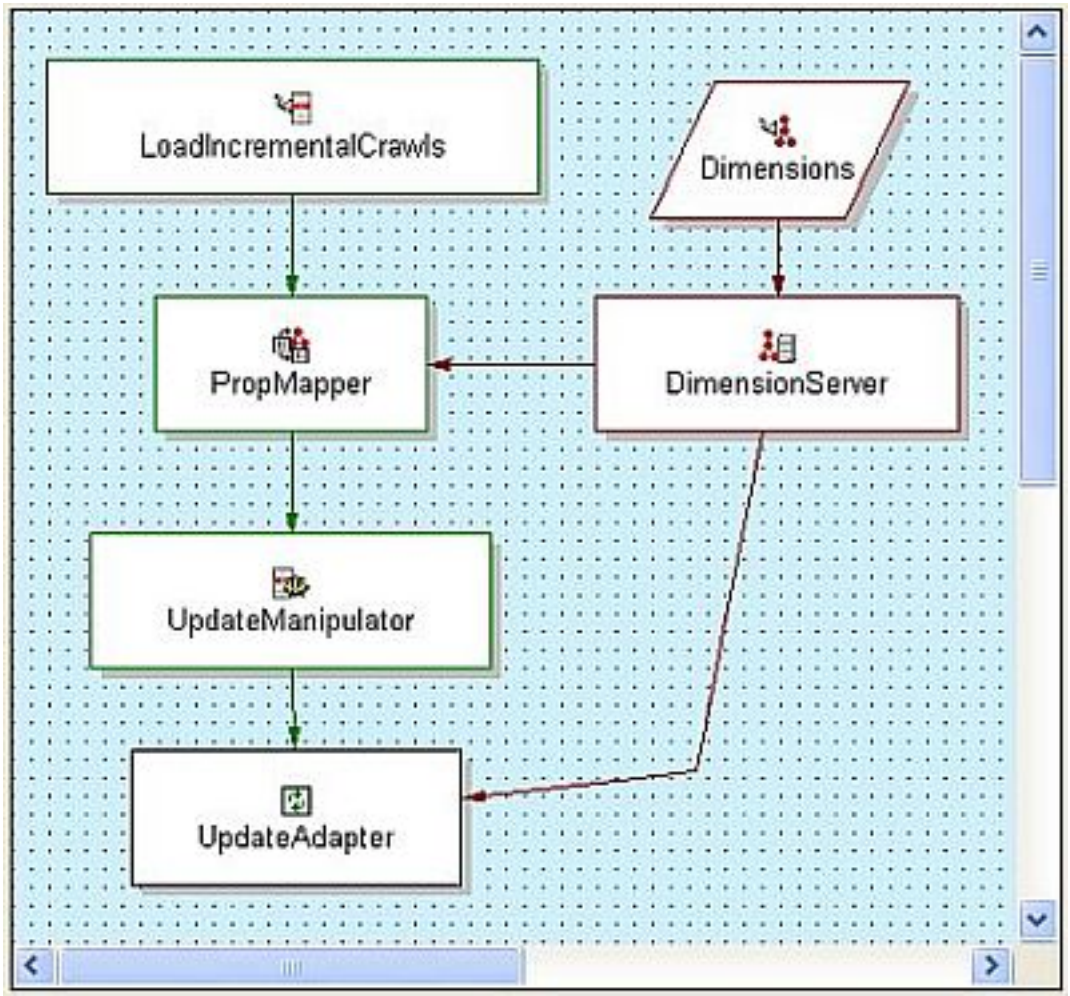
```
INFO: Baseline update script finished.
```

After completion, the Dgraph should be running on the host and port specified in the `AppConfig.xml` configuration file.

About running partial updates

The sample CAS crawl pipeline also supports partial updates, as an alternative way to apply incremental CAS crawl updates to a Dgraph.

The partial pipeline looks as follows:



The `LoadIncrementalCrawls` record adapter reads in the incremental crawl data.

The main work is done by the `UpdateManipulator` (a record manipulator), which uses expressions that evaluate the value of the Endeca.Action property on a record:

- If the value of the `Endeca.Action` property is "`UPSERT`", then use an `UPDATE_RECORD` expression with an action of `ADD_OR_REPLACE` in order to add or update the record.
- If the value of the `Endeca.Action` property is "`DELETE`", then use an `UPDATE_RECORD` expression with an action of `DELETE_OR_IGNORE` in order to remove the record.

A partial update can only be run after a full crawl has been run and processed with a baseline update. In addition, if partial updates are being used, the baseline pipeline should be run only after a full crawl and not after incremental crawls.

Note: For completeness, when using partial updates, the sample baseline pipeline should be updated so that it no longer reads and joins incremental crawl files (i.e., the delta functionality should be removed from the pipeline), since the incremental files will be processed and applied as partial updates instead.

Running partial updates using record output files

The Deployment Template includes a sample CAS crawl pipeline that you can also use to run partial updates.

Make sure that you have updated the `AppConfig.xml` file to use the sample CAS crawl pipeline located in `[appdir]/config/cas_crawl_pipeline`. Specifically, you will need to update the `configDir` in the `PartialForge` section of the document.

To run the sample partial update pipeline:

1. Run the `load_incremental_cas_crawl_data` located in the `[appdir]/control/cas` directory.

   This script copies incremental crawl data to the `data/partials/incoming` directory.
2. Run the `partial_update` script located in `[appdir]/control`.

The `partial_update` script displays the following informational message when the process is complete:

```
INFO: Partial update script finished.
```

The Dgraph should now be updated with the changes from your incremental crawls.

Note on running partial updates with XML input files

If you are using uncompressed XML files as the source files for partial updates, you should keep in mind that the `data/partials/processing` directory also contains other XML files (i.e., the project configuration XML files).

Therefore, if you are using a wildcard in the URL field for the input record adapter, you must make sure that it does not also reference the configuration XML files.

For example, avoid specifying a URL setting like this:

```
URL: *.xml
```

Instead, you should use a URL setting that references the `INCR` prefix of the output filename, similar to this example:

```
URL: *-INCR.xml
```

This will load files such as `CrawlerOutput-INCR.xml` but not the configuration files. See the *Oracle Endeca CAS Developer's Guide* for details on the naming format of the crawl output files.

Crawler scripts for record output files

There are four Deployment Template scripts that are used with CAS crawls that write output to record output files. This section describes how to configure the scripts to manage record output files.

- Baseline crawl script (in `[crawlname]CasCrawlConfig.xml`)
- Incremental crawl script (in `[crawlname]CasCrawlConfig.xml`)
- Baseline update fetch script (in `fetchCasCrawlDataConfig.xml`). Used for baseline updates that may or may not use incremental (delta) updates.
- Incremental fetch script (in `fetchCasCrawlDataConfig.xml`). Used for partial updates.

Baseline crawl script

This topic describes how to set up a baseline crawl script that manages record output files.

Because the script is basically the same for all file system and CMS crawl configurations, the `EndecaCasCrawlConfig.xml` sample is used to illustrate the script (for a crawl named `Endeca`).

```
<script id="Endeca_baselineCasCrawl">
    <![CDATA[
  crawlName = "Endeca";
```

1. Check if the crawl is set to write output to a record output file, and throw an exception if the crawl is set to output to a Record Store instance.

```
  if (!CAS.isCrawlFileOutput(crawlName)) {
      throw new UnsupportedOperationException("The crawl " + crawlName
+
      " does not have a File System output type. The only supported
       output type for this script is File System.");
      }
   log.info("Starting full CAS crawl '" + crawlName + "'.");
```

2. Obtain a lock on the crawl. The baseline crawl attempts to set a flag in the EAC to serve as a lock or mutex. The name of the flag is the string "`crawl_lock_`" plus the name of the crawl (such as "`crawl_lock_Endeca`" for this example). If the flag is already set, this step fails, ensuring that a crawl (either baseline or incremental) cannot be started more than once simultaneously, as this would interfere with data processing. The flag is removed in the case of an error or when the script completes successfully.

```
  // obtain lock
  if (LockManager.acquireLock("crawl_lock_" + crawlName)) {
```

3. Clean the output directories. Baseline and incremental crawl output files from the previous crawls are removed from the crawl's configured output directory.

```
  CAS.cleanOutputDir(crawlName);
```

4. Run the baseline crawl. The baseline crawl is run with the crawl name as the ID.

```
  CAS.runBaselineCasCrawl(crawlName);
```

5. Rename the output file. The baseline crawl output file is renamed by prefixing the crawl name.

```
  CAS.renameBaselineCrawlOutput(crawlName);
```

6. Obtain a second lock on the complete crawl data directory. The script attempts to set a flag to serve as a lock on the `data/complete_cas_crawl_output` directory. If the flag is already set, this step idles for up to ten minutes, waiting for the flag to become available. If the flag remains set for 10 minutes, this action fails, meaning that the renamed output file is not copied. This step ensures that access to the directory is synchronized, so that a downstream process like the baseline update does not retrieve a half-delivered crawl file.

```
  // try to acquire a lock on the complete crawl data directory
  // for up to 10 minutes
  if (LockManager.acquireLockBlocking("complete_cas_crawl_data_lock",
      600))
```

7. Get the path of the output destination directory. The path of the destination directory (to which the baseline crawl output file will be copied) is obtained. The directory name is specified by the `casCrawlFullOutputDestDir` property in the `fetchCasCrawlDataConfig.xml` file, which is `data/complete_cas_crawl_output/full` by default.

```
destDir = PathUtils.getAbsolutePath(CAS.getWorkingDir(),
  CAS.getCasCrawlFullOutputDestDir());
```

8. Create the destination directory. The destination directory for the crawl output file is created if it does not exist. The name is in the `destDir` variable.

```
// create the target dir, if it doesn't already exist
mkDirUtil = new CreateDirUtility(CAS.getAppName(),
  CAS.getEacHost(), CAS.getEacPort(), CAS.isSslEnabled());
mkDirUtil.init(Forge.getHostId(), destDir, CAS.getWorkingDir());
mkDirUtil.run();
```

9. Delete existing baselines. To ensure that no previous baseline crawl files are left, all baseline output files (if they exist) must be removed.

```
// clear the destination dir of full crawl files, in case
// we are not overwriting the same file such as when the
// crawl output format has changed.
CAS.clearFullCrawlOutputFromDestinationDir(crawlName);
```

10. Delete existing incrementals. Because this is a baseline crawl, existing incremental output files must be removed.

```
// remove previously collected incremental crawl files,
// which are expected to be incorporated in this full crawl
CAS.clearIncrementalCrawlOutputFromDestinationDir(crawlName);
```

11. Copy the output file to the destination directory. The renamed baseline crawl output file is copied from the original output directory to the destination directory (`data/complete_cas_crawl_output/full` by default).

```
// deliver crawl output to destination directory
CAS.copyBaselineCrawlOutputToDestinationDir(crawlName);
```

12. Release the second lock. The "`complete_cas_crawl_data_lock`" flag is removed from the EAC, indicating that the copy operation was successful.

```
// release lock on the crawl data directory
LockManager.releaseLock("complete_cas_crawl_data_lock");
```

13. Release the first lock. The "`crawl_lock_Endeca`" flag is removed from the EAC (indicating that the crawl operation was successful) and a "`finished`" message is logged.

```
LockManager.releaseLock("crawl_lock_" + crawlName);
...
log.info("Finished full CAS crawl '" + crawlName + "'.");
```

Incremental crawl script

This topic describes how to set up an incremental crawl script that manages record output files.

Because the script is basically the same for all file system and CMS crawl configurations, the `EndecaCasCrawlConfig.xml` sample is used to illustrate the script (for a crawl named `Endeca`).

```
<script id="Endeca_incrementalCasCrawl">
    <![CDATA[
  crawlName = "Endeca";
```

1. Check if the crawl is set to write output to a record output file, and throw an exception if the crawl is set to output to a Record Store instance.

   ```
   if (!CAS.isCrawlFileOutput(Endeca)) {
       throw new UnsupportedOperationException("The crawl " +
       crawlName + " does not have a File System output type.
       The only supported output type for this script is
       File System.");
       }
     log.info("Starting incremental CAS crawl '" + crawlName + "'.");
   ```

2. Obtain a lock on the crawl. The incremental crawl attempts to set a flag in the EAC to serve as a lock or mutex. The name of the flag is the string "`crawl_lock_`" plus the name of the crawl (such as "`crawl_lock_Endeca`" for this example). If the flag is already set, this step fails, ensuring that a crawl (either baseline or incremental) cannot be started more than once simultaneously, as this would interfere with data processing. The flag is removed in the case of an error or when the script completes successfully.

   ```
   // obtain lock
   if (LockManager.acquireLock("crawl_lock_" + crawlName)) {
   ```

3. Clean the output directories. Any previous crawl output file (either baseline or incremental) is removed from the crawl's configured output directory (that is, the output directory that was configured when the crawl was created). The `data/complete_cas_crawl_output` directory is not affected.

   ```
   CAS.cleanOutputDir(crawlName);
   ```

4. Run the incremental crawl. The incremental crawl is run with the crawl name as the ID.

   ```
   CAS.runIncrementalCasCrawl(crawlName);
   ```

5. Rename the output file. The incremental crawl output file is renamed by prefixing the crawl name and a timestamp, to indicate the order in which the incremental crawl file was generated relative to others.

   ```
   CAS.renameIncrementalCrawlOutput(crawlName);
   ```

6. Obtain a second lock on the complete crawl data directory. The script will attempt to set a flag to serve as a lock on the `data/complete_cas_crawl_output` directory. If the flag is already set, this step will idle for up to ten minutes, waiting for the flag to become available. If the flag remains set for 10 minutes, this action will fail, meaning that the renamed output file is not copied. This step ensures that access to the directory is synchronized, so that a downstream process like the baseline update does not retrieve a half-delivered crawl file.

   ```
   // try to acquire a lock on the complete crawl data directory
   // for up to 10 minutes
   if (LockManager.acquireLockBlocking("complete_cas_crawl_data_lock",
       600))
   ```

7. Get the path of the output destination directory. The path of the destination directory (to which the incremental crawl output file will be copied) is obtained. The directory name is specified by the `casCrawlIncrementalOutputDestDir` property (which is

`data/complete_cas_crawl_output/incremental` by default) in the `fetchCasCrawlDataConfig.xml` file.

```
destDir = PathUtils.getAbsolutePath(CAS.getWorkingDir(),
  CAS.getFsCrawlIncrementalOutputDestDir());
```

8. Create the destination directory. The destination directory for the crawl output file is created if it does not exist. The name is in the `destDir` variable.

```
// create the target dir, if it doesn't already exist
mkDirUtil = new CreateDirUtility(CAS.getAppName(),
  CAS.getEacHost(), CAS.getEacPort(), CAS.isSslEnabled());
mkDirUtil.init(Forge.getHostId(), destDir, CAS.getWorkingDir());
mkDirUtil.run();
```

9. Copy the output file to the destination directory. The renamed incremental crawl output file is copied from the original output directory to the destination directory (`data/complete_cas_crawl_output/incremental` by default).

```
// deliver crawl output to destination directory
CAS.copyIncrementalCrawlOutputToDestinationDir(crawlName);
```

10. Release the second lock. The "`complete_cas_crawl_data_lock`" flag is removed from the EAC, indicating that the copy operation was successful.

```
// release lock on the crawl data directory
LockManager.releaseLock("complete_cas_crawl_data_lock");
```

11. Release the first lock. The "`crawl_lock_Endeca`" flag is removed from the EAC (indicating that the crawl operation was successful) and a "`finished`" message is logged.

```
LockManager.releaseLock("crawl_lock_" + crawlName);
...
log.info("Finished incremental CAS crawl '" + crawlName +
  "'.");
```

Fetch baseline crawl data script

This fetch script is used to copy the crawl data to the appropriate directories for all baseline update operations, including those performed with a delta update pipeline. The script is included in this section, with numbered steps indicating the actions performed at each point in the script.

Note that the script does not actually perform the baseline update itself; that update operation is managed by scripts in the `AppConfig.xml` document.

```
<script id="fetchFullCasCrawlData">
    <![CDATA[
  log.info("Fetching full CAS crawl data for processing.");
```

1. Obtain a lock on the complete crawl data directory. The script attempts to set a flag to serve as a lock on the `data/complete_cas_crawl_output` directory. If the flag is already set, this step idles for up to ten minutes, waiting for the flag to become available. If the flag remains set for 10 minutes, this action fails, meaning that the renamed output file is not copied. This step ensures that access to the directory is synchronized, so that a downstream process like the baseline update does not retrieve a half-delivered crawl file.

```
// try to acquire a lock on the complete crawl data directory
// for up to 10 minutes
if (LockManager.acquireLockBlocking("complete_cas_crawl_data_lock",
    600))
```

2. Release the baseline data ready lock. The "`baseline_data_ready`" flag is removed from the EAC, which ensures that the baseline update does not start until all the data sources have been copied to the proper directories.

```
// remove baseline data ready flag, ensuring baseline doesn't start
// before data is completely copied and ready for processing
LockManager.removeFlag("baseline_data_ready");
```

3. Get the paths of the source data directories. The paths of the source data directories are obtained. The directory names are set by the `fsCrawlFullOutputDestDir` and `fsCrawlIncremen¬talOutputDestDir` properties of the custom-component section.

```
fullSrcDir = PathUtils.getAbsolutePath(CAS.getWorkingDir(),
  CAS.getCasCrawlFullOutputDestDir()) + "/\\*";
incrSrcDir = PathUtils.getAbsolutePath(CAS.getWorkingDir(),
  CAS.getCasCrawlIncrementalOutputDestDir()) + "/\\*";
```

4. Get the paths of the destination directories. The paths of the destination directories (to which crawl output files will be copied) are obtained. The directory name is specified by the `IncomingDataDir` property (`./data/incoming` is the default) in the Forge section of the `AppConfig.xml` file. Note that `/full` and `/incremental` will be added to the incoming name.

```
fullDestDir = PathUtils.getAbsolutePath(Forge.getWorkingDir(),
  Forge.getIncomingDataDir()) + "/full";
incrDestDir = PathUtils.getAbsolutePath(Forge.getWorkingDir(),
  Forge.getIncomingDataDir()) + "/incremental";
```

5. Create the destination directories. The destination directories for the source data files are created if they do not exist. The directory names are in the `fullDestDir` and `incrDestDir` variables.

```
// create destination directories
mkDirUtil = new CreateDirUtility(Forge.getAppName(),
  Forge.getEacHost(), Forge.getEacPort(), Forge.isSslEnabled());
mkDirUtil.init(Forge.getHostId(), fullDestDir, Forge.getWorkingDir());

mkDirUtil.run();

mkDirUtil.init(Forge.getHostId(), incrDestDir, Forge.getWorkingDir());

mkDirUtil.run();
```

6. Copy the source data to the destination directories. Instantiate a `CopyUtility` object (named `crawlDataCopy`) and use it to copy the full and incremental source data to the `data/incoming` directories.

```
crawlDataCopy = new CopyUtility(Forge.getAppName(),
  Forge.getEacHost(), Forge.getEacPort(), Forge.isSslEnabled());

// copy full crawl data
crawlDataCopy.init("copy_complete_cas_full_crawl_data",
  CAS.getCasCrawlOutputDestHost(),Forge.getHostId(), fullSrcDir,
  fullDestDir, true);
crawlDataCopy.run();

// copy incremental crawl data
crawlDataCopy.init("copy_complete_cas_incremental_crawl_data",
  CAS.getCasCrawlOutputDestHost(),Forge.getHostId(), incrSrcDir,
  incrDestDir, true);
crawlDataCopy.run();
```

7. If no incremental files exist, create a dummy file. Forge will fail when running the delta pipeline if there are no incremental files. Therefore, verify whether incremental files exist and, if none exist,

create a dummy file named "`placeholder.bin.gz`". If at least one incremental file exists, skip to Step 11 (the else statement). Note that the comments in the following code were added to explain the steps.

Note: This step is required to support the behavior of the default pipeline included with the Deployment Template. Specifically, the baseline pipeline always expects to read a set of full crawl files and a set of incremental crawl files and joins these by keeping the most recent copy of each record that's available between the files. Forge fails when no incremental files are available, so this dummy file ensures that the pipeline works when a full crawl has been run, but no incremental crawls have been run. For pipeline implementations that do not require such a dummy file (e.g., pipelines that only process full crawls), this step can be removed.

```
// test for existing incremental files, since the dummy file is only
// needed when there are no real incremental files
if (! fileUtil.dirContainsFiles(incrDestDir, Forge.getHostId())) {
  // create a variable for the dummy file name and location
  placeholder = incrDestDir + "/placeholder.bin";
  // create Unix touch and gzip commands
  touchCmd = "touch " + placeholder;
  zipCmd = "gzip " + placeholder;
  // for Windows platforms, rewrite the commands using Win commands
  if (System.getProperty("os.name").startsWith("Win")) {
    touchCmd = "%ENDECA_ROOT%\\utilities\\touch.exe " + placeholder;
    zipCmd = "%ENDECA_ROOT%\\utilities\\gzip.exe " + placeholder;
  }
  // use a ShellUtility to touch (i.e. create) the dummy file
  shell = new ShellUtility(Forge.getAppName(), Forge.getEacHost(),
    Forge.getEacPort(), Forge.isSslEnabled());
  shell.init("create_incremental_cas_crawl_placeholder",
    Forge.getHostId(),touchCmd, Forge.getWorkingDir());
  shell.run();
  // use the same ShellUtility to produce a .bin.gz compressed file
  shell.init("zip_incremental_cas_crawl_placeholder",
    Forge.getHostId(),zipCmd, Forge.getWorkingDir());
  shell.run();
} // end of if clause
```

8. If the test at Step 7 showed the incremental directory was not empty, rename the incremental files (which have timestamped names) so they are read in reverse chronological order. This means that the name of the latest (most recent) file must begin with 000001, the next one with 000002, and so on.

Note: As with the previous step, this logic is required to support the behavior of the default pipeline. This step ensures that Forge keeps the most up-to-date copy of any record, ignoring any older copies. Since files are read and processed in alphanumeric order, renaming them ensures that the most recent records are processed first.

```
// incremental files do exist, so rename them
else {
  // get the number of files, to be used to generate the prefix
  incrFiles = fileUtil.getDirContents(incrDestDir, Forge.getHostId());

  fileNum = incrFiles.size();
  // import Java classes we will use for the renaming
  import java.text.NumberFormat;
  import java.text.DecimalFormat;
  import java.util.SortedMap;
  import java.util.TreeMap;
```

```
    import java.io.File;
    // instantiate a NumberFormat to format the prefix name
    NumberFormat formatter = new DecimalFormat("000000");
    // instantiate a SortedMap and add the file names,
    // which will be in an ascending key order
    SortedMap sortedFiles = new TreeMap();
    sortedFiles.putAll(incrFiles);
    // loop through the sorted treemap
    for (incrFile : sortedFiles.keySet()) {
      // generate a filename prefix, based on the number of files left
      prefix = formatter.format(fileNum);
      // get the original filename and prepend the generated prefix
      origFileName = PathUtils.getFileNameFromPath(incrFile);
      newFileName = prefix + "_" + origFileName;
      // generate the pathname to which we will rename the file
      absNewFile = PathUtils.getAbsolutePath(Forge.getWorkingDir(),
        Forge.getIncomingDataDir()) + File.separator + "incremental" +
        File.separator + newFileName;
      // use the LocalMoveUtility to rename the file
      renameUtil = new LocalMoveUtility(Forge.getAppName(),
        Forge.getEacHost(), Forge.getEacPort(), Forge.isSslEnabled());
      renameUtil.init(Forge.getHostId(), incrFile, absNewFile,
        Forge.getWorkingDir());
      renameUtil.run();
      // decrease the fileNum variable by one so that the name of the
      // next file will be numerically more recent
      fileNum--;
    } // end of for loop
  } // end of else clause
```

9. Set the baseline data ready flag. Set the "`baseline_data_ready`" flag in the EAC, which means that baseline updates can be performed at any time.

```
  // (re)set flag indicating that the baseline can process incoming data

  LockManager.setFlag("baseline_data_ready");
```

10. Release the lock. The "`complete_cas_crawl_data_lock`" flag is removed from the EAC, indicating that the fetch operation was successful. A "`finished`" message is also logged.

```
  // release lock on the crawl data directory
  LockManager.releaseLock("complete_cas_crawl_data_lock");
  ...
  log.info("Crawl data fetch script finished.");
```

Fetch incremental crawl data script

This fetch script is used to copy the incremental crawl output files to the appropriate directory for a partial update. The script is included in this section, with steps indicating the actions performed at each point in the script.

The script does not actually perform the partial update itself; that update operation is managed by scripts in the `AppConfig.xml` document.

```
<script id="fetchIncrementalCasCrawlData">
   <![CDATA[
 log.info("Fetching incremental CAS crawl data for processing.");
```

1. Obtain a lock on the complete crawl data directory. The script will attempt to set a flag to serve as a lock on the `data/complete_cas_crawl_output` directory. The flag will be removed in the case of an error or when the script completes successfully.

   ```
   // try to acquire a lock on the complete crawl data directory
   // for up to 10 minutes
   if (LockManager.acquireLockBlocking("complete_cas_crawl_data_lock",
       600))
   ```

2. Get the path of the source data directory. The path of the source data directory is obtained. The directory name is set by the `casCrawlIncrementalOutputDestDir` property of the custom-component section.

   ```
   incrSrcDir = PathUtils.getAbsolutePath(CAS.getWorkingDir(),
     CAS.getCaCrawlIncrementalOutputDestDir()) + "/\\*";
   ```

3. Get the path of the destination directory. The path of the destination directory (to which incremental output files will be copied) is obtained. The directory name is specified by the `IncomingDataDir` property (`./data/partials/incoming` is the default) in the `PartialForge` section of the `AppConfig.xml` file.

   ```
   incrDestDir = PathUtils.getAbsolutePath(PartialForge.getWorkingDir(),
     PartialForge.getIncomingDataDir());
   ```

4. Copy the incremental source data to the destination directory. Instantiate a `CopyUtility` object (named `crawlDataCopy`) and use it to copy the incremental source data to the `data/partials/incoming` directories.

   ```
   // copy incremental crawl data
   crawlDataCopy = new CopyUtility(PartialForge.getAppName(),
     PartialForge.getEacHost(), PartialForge.getEacPort(),
     PartialForge.isSslEnabled());
   crawlDataCopy.init("copy_complete_cas_incremental_crawl_data",
    CAS.getFsCrawlOutputDestHost(),PartialForge.getHostId(), incrSrcDir,

     incrDestDir, true);
   crawlDataCopy.run();
   ```

5. Set flags to indicate that files are ready for partial update processing. Default partial update functionality in the `AppConfig.xml` script expects flags to be set to indicate which files are ready for partial update processing. For each file delivered in the previous step, a flag is set with the name of the file prefixed by the string "`partial_extract::`".

   ```
   // (re)set flags indicating which partial update files are ready
   // for processing -- convention is "partial_extract::[filename]"
   fileUtil = new FileUtility(PartialForge.getAppName(),
     PartialForge.getEacHost(), PartialForge.getEacPort(),
     PartialForge.isSslEnabled());
   dirContents = fileUtil.getDirContents(incrDestDir,
     PartialForge.getHostId());

   for (file : dirContents.keySet()) {
     fileName = PathUtils.getFileNameFromPath(file);
     LockManager.setFlag("partial_extract::" + fileName);
   }
   ```

6. Release the lock. The "`complete_cas_crawl_data_lock`" flag is removed from the EAC, indicating that the fetch operation was successful. A "`finished`" message is also logged .

```
// release lock on the crawl data directory
LockManager.releaseLock("complete_cas_crawl_data_lock");
...
log.info("Crawl data fetch script finished.");
```

Chapter 7

Inserting a Custom Pipeline

This section describes how to modify an existing pipeline or create a new pipeline that takes advantage of the Deployment Template structure.

About the sample pipelines

For testing purposes, the Deployment Template includes two sets of pipelines and configuration files.

The two sets of pipelines and configuration files are located in two sub-directories of the `[appdir]/config` directory:

- The `pipeline` directory contains configuration files from the sample_wine_data reference project that ships with Oracle Endeca Guided Search. The corresponding source data is in the `[appdir]/test_data` directory.

  Note: This guide assumes that you will be placing the configuration files for your pipeline in this directory.

- The `cas_crawl_pipeline` directory contains a delta update pipeline that is tailored for Oracle Endeca CAS Server file system and CMS crawls. No source data is provided, so you will have to run a crawl.

While these sample pipelines facilitate testing of the deployment template, these files are intended to be replaced with project-specific files immediately after a deployment template has been properly configured.

This document describes how to modify or create a pipeline that is designed for integration within the deployment template operational structure. This includes pipeline naming requirements, common errors encountered, etc.

This document assumes that the reader has created a new deployment template application in the `[appdir]` directory, and has run the deployment script (`deploy.bat` or `deploy.sh`). Note that the usage guide also documents how to configure and run CAS crawls using the deployment scripts.

Sample pipeline overview

This section describes the high-level steps that are necessary to integrate a new/existing pipeline with a deployment template.

Additional detail on each of these steps is provided in later sections.

1. Ensure that the application name and pipeline configuration prefix match the data prefix configured in the deployment template.
2. Place pipeline configuration files in the `[appdir]/config/pipeline/` directory of the primary server.
3. In order to enable partial updates, ensure that the project is configured with a record spec (i.e., a unique record identifier property).
4. Ensure that any input Record Adapters requiring filenames specify the file location relative to the `[appdir]/data/processing/` (or `[appdir]/data/partials/processing`) directory.

Location of pipeline configuration files

A deployment template's pipeline configuration files should be located on the primary server (i.e., the server on which the deployment template is installed, and usually the server hosting the EAC Central Server).

Agent servers in a standard deployment template do not require copies of the pipeline configuration file.

Pipeline configuration files can be found in the following location:

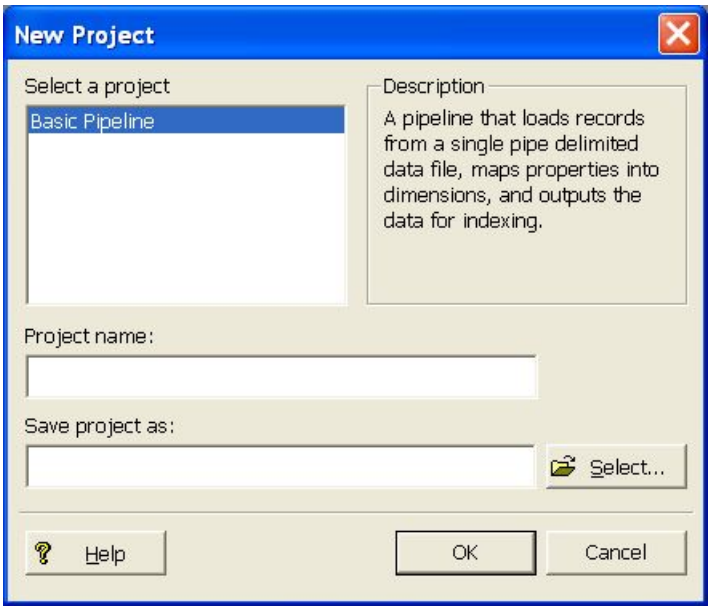
```
Primary Server: [appdir]/config/pipeline/*
```

Note: When inserting a project-specific pipeline into a deployment template, make sure to first delete the existing contents of the `config/pipeline` directory.

Creating a new project

Once the reference configuration files have been deleted, a new pipeline configuration project can be created.

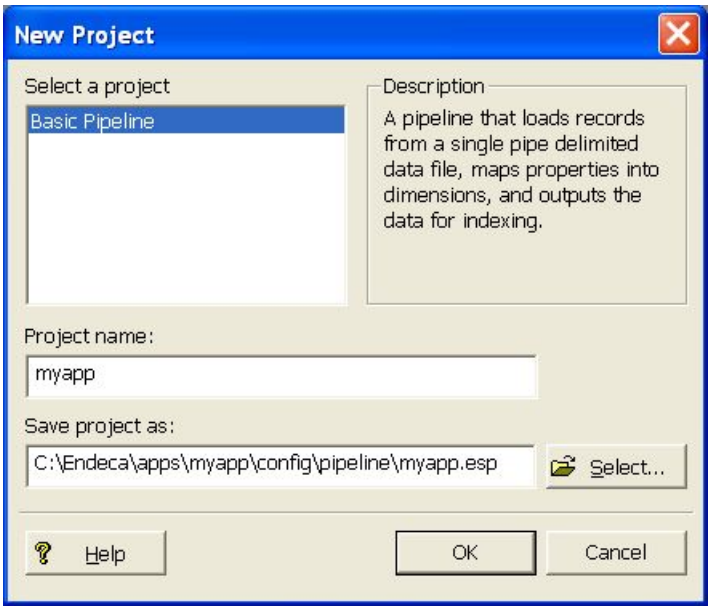
When creating a new project using the Oracle Endeca Developer Studio, the user will be prompted with the following dialog box:



To create a new project:

1. In order for a new pipeline to be run properly within the deployment template, the following must be properly specified:
   a) The **Project Name** field must be the same as the data prefix specified for the "app" element in `[appdir]/config/script/AppConfig.xml`. By default, this data prefix will have been set to the name of the application that was specified when running `deploy.bat` or `deploy.sh`.
   b) Recall that the `[appname]` specified was also used to create the base `[appdir]` directory. For example, if "myapp" was supplied as the `[appname]`, and "`c:\Endeca\apps`" was supplied as the Deployment Directory, then `[appdir]` would be `c:\Endeca\apps\myapp`. In this example, the Project Name should also be specified as "myapp".
2. The **Save Project As** field should be `[appdir]\config\pipeline\[appname].esp`

   In the example above, the **Save Project As** field would be `c:\Endeca\apps\myapp\config\pipeline\myapp.esp`.



After clicking the "**OK**" button, a number of files will be created in the `[appdir]/config/pipeline/` directory. The primary files to be concerned with are listed below:

File name	Description
`pipeline.epx`	This is the main pipeline file that the deployment template will reference when running forge.
`[appname].esp`	This is the Developer Studio project file that will be used whenever reopening the project. Although this file does not actually require the `[appname]` prefix, it is good practice to keep it consistent with other project files.
`[appname].*.xml`	These are the various configuration files that will be used later by the indexer and MDEX Engine processes. It is important that they have the same prefix as the deployment template `ApplicationName`.
`dimensions.xml`	This is the dimension file referenced by the default Dimension Adapter.

Modifying an existing project

Modifying an existing Developer Studio project to match a new deployment template application is a somewhat tedious task. In fact, it is often easier to simply create a new deployment template application instead.

The important key is that the `[appname].*.xml` files share the same `[appname]` as the deployment template project. Since there are 30+ XML files, you can either:

- Rename each of the XML files with a new prefix, and update the [appname].esp file to reference each new file.
- Update the deployed application's `AppConfig.xml` file to specify the `[appname]` of your configuration files. For example, if your configuration files are named `myapp.*.xml`, update the configuration as follows:

```
<app appName="myapp" eacHost="host1.company.com" eacPort="8888"
    dataPrefix="myapp" sslEnabled="false"
    lockManager="LockManager">
  <working-dir>C:\Endeca\apps\myapp</working-dir>
  <log-dir>./logs/baseline</log-dir>
</app>
```

In most cases, the `appName` attribute and the `dataPrefix` attribute will be identical. However, this is not required and an application can be configured to support files with a data prefix other than the application name. If the data prefix is not specified, the application defaults to using the application name.

Note that opening an existing project in the Oracle Endeca Developer Studio and using the **Save As** feature will not rename the corresponding `*.xml` files. It will only rename the `[appname].esp` file. The prefix for the XML files can only be specified when a new project is created.

Related Links

Common errors on page 117
This section provides troubleshooting information for commonly received errors.

Configuring a record specifier

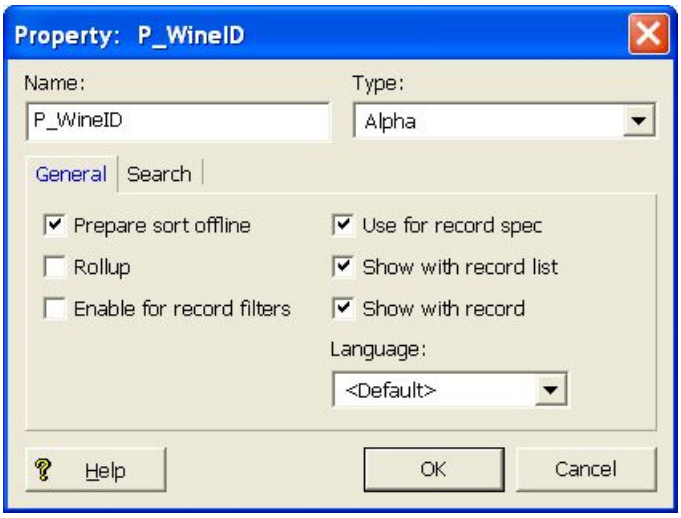
The deployment includes support for both baseline and partial index updates. In order to support partial updates, an application must include a record specifier, which is a property marked as the unique identifier of records in the index.

For details about the record specifier property, refer to the *Oracle Endeca Forge Guide*.

When configuring your application, identify a property for which each record will have a unique assigned value. For example, in the reference `sample_wine_data` project, each record in the data set includes a unique `Wine ID`.

To enable the use of that property as a record spec:

1. Open the **Property** dialog box in Developer Studio.
2. Check the box labeled "**Use for record spec**."



Forge flags

In order to reduce the amount of configuration required to integrate a pipeline into a deployment template, a standard deployment template application runs the primary and partial update Forge processes with an abbreviated set of flags.

Since the deployment template already specifies directory structures and file prefixes, the following flags are used to override a pipeline's input and output components, specifying the appropriate directories and prefixes for either reading or writing data.

Primary Forge flags

Flag	Description
`--inputDir`	`[appdir]/data/processing`
`--stateDir`	`[appdir]/data/state`
`--tmpDir`	`[appdir]/data/forge_temp`
`--logDir`	`[appdir]/logs/baseline`
`--outputDir`	`[appdir]/data/forge_output`
`--outputPrefix`	`[dataPrefix]`

Partial update Forge flags

Flag	Description
`--inputDir`	`[appdir]/data/partials/processing`
`--stateDir`	`[appdir]/data/state`
`--tmpDir`	`[appdir]/data/forge_temp`
`--logDir`	`[appdir]/logs/partial`
`--outputDir`	`[appdir]/data/partials/forge_output`

Flag	Description
`--outputPrefix`	`[dataPrefix]`

Input record adapters

The record adapters load the source data.

To start, here is a quick review of how sample data included with the deployment template is processed. The sample wine application includes a sample dataset in the *[appdir]/test_data/baseline* directory. When processing the sample wine data, the load_baseline_test_data.bat or load_baseline_test_data.sh scripts are used to copy the contents of this directory into the [appdir]/data/incoming/ directory, as well as to set a flag in the EAC. This flag, named baseline_data_ready, indicates to the deployment template scripts that the data extraction process is complete and data is ready for processing. Once that has occurred, the deployment template's baseline update process copies these files into the [appdir]/data/processing/ directory before running the primary Forge process.

When using a default deployment template application, it is therefore necessary for all input record adapters to look in the [appdir]/data/processing/ directory for incoming data extracts. The deployment template handles this automatically by specifying the --inputDir flag when running the primary forge process. This flag overrides any absolute path specified for specific input adapters with the proper deployment template path: [appdir]/data/processing/. However, the --inputDir flag respects relative paths, resolving them relative to the path specified as the input directory.

The URL property of any record adapter component therefore only needs to specify the relative path to a specific file or subdirectory within the [appdir]/data/incoming/ directory. (Remember that files and subdirectories in the incoming directory are copied to the processing directory by the deployment template before Forge is run.)

For example, if a single extract file called data.txt is copied into the [appdir]/data/incoming/ directory before running a baseline, the URL property of that data's input record adapter should specify a URL of data.txt.

For a more complex deployment where, for instance, multiple text extract files are copied into the [appdir]/data/incoming/extracted_data/ directory before running a baseline update, the URL property of a single input record adapter configured to read these files should be set to extracted_data/*.txt.

Related Links

Output record adapters on page 116

Output record adapters are often used to generate debug or state information. By default, the location to which this data is written will be overridden by the `--outputDir` flag.

Dimension adapters

The `--inputDir` flag specified to forge overrides the input URL for dimension adapters.

Since the dimensions for a project are usually stored in the `[appdir]/config/pipeline` directory along with other configuration files, the deployment template copies these files into the `[appdir]/data/processing/` directory before running the Forge process. The URLs specified in dimension adapters should follow the same rules as those described for input record adapters, specifying dimension XML file URLs relative to the `--inputDir` directory. In most cases, this is as simple as

specifying the URL for the main dimension adapter as `Dimensions.xml`, which is the value used by the default "Dimensions" adapter created by Developer Studio's project template.

More complex deployments that include multiple dimension adapters or external delivery of dimension files should ensure that the dimension XML files are copied into the `[appdir]/data/incoming/` directory before the forge process runs.

Indexer adapters

Since `--outputPrefix` and `--outputDir` flags are both included, the deployment template will override any values specified for the Indexer Adapter "URL" and "Output prefix" properties.

Therefore, it is unnecessary to modify these properties in most cases.

For Agraph deployments, the `PROP_NAME` attribute of the `ROLLOVER` element in your indexer adapter must be specified. It should be set to a property which is the unique identifier for records in your data set. For standard Forge deployments, this element should be specified in the main pipeline. For Parallel Forge deployments, this element should be specified in the data splitter pipeline.

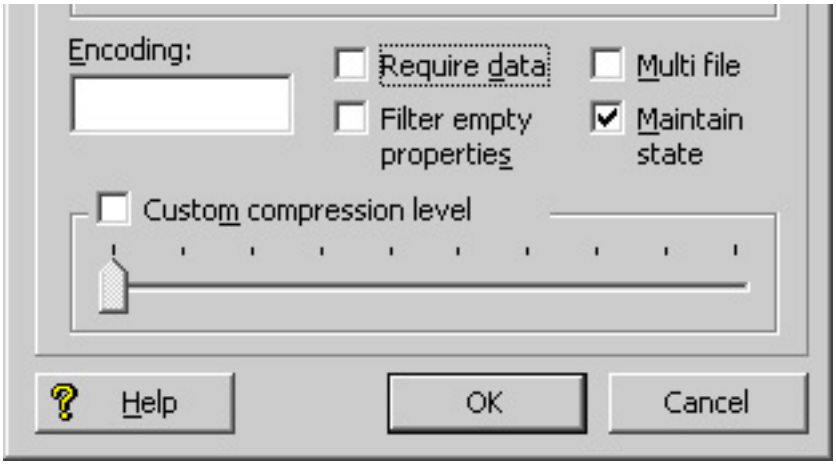
Note: The number of partitions for Agraph deployments need not be set in the pipeline, as the value will be specified as a forge command line argument, overriding the value in the pipeline. However, the rollover key on which data will be partitioned must be specified as described above.

Output record adapters

Output record adapters are often used to generate debug or state information. By default, the location to which this data is written will be overridden by the `--outputDir` flag.

In most cases, however, it is undesirable for these files to be written to the same location as the Forge output files.

In these cases, an output record adapter can be configured to instead respect the `--stateDir` flag by selecting the "**Maintain State**" checkbox.



Now any files generated by this output record adapter will be written to the `[appdir]/data/state/` directory.

Note that the output file name must still be specified in the "URL" property of the record adapter. The `--outputPrefix` flag only overrides the indexer adapter output file names, not output record adapter file names.

Related Links

Input record adapters on page 115
The record adapters load the source data.

Dimension servers

The `--stateDir` flag will override the URL value for all Dimension Server components, and place any autogen state files in the `[appdir]/data/state/` directory.

Common errors

This section provides troubleshooting information for commonly received errors.

Unable to Find Pipeline.epx

If Forge fails, check the logs (`[appdir]/logs/baseline/err.forge`) to make sure that Forge was able to find the `pipeline.epx` file in its proper location. Remember that a basic deployment template application assumes that it will find the project's `pipeline.epx` file in `[appdir]\config\pipeline\`.

On UNIX platforms, file names are case sensitive. The deployment template expects the primary pipeline file to be named `pipeline.epx` and the partial update pipeline (if one is required for the deployed application) to be named `partial_pipeline.epx`. Ensure that the files in your deployment use this capitalization.

Missing Configuration Files

This more common error is also more difficult to detect. Since all pipelines created by the Oracle Endeca Developer Studio typically contain a `Pipeline.epx` file, it is unlikely that the Forge process will be unable to find the file, unless it was placed in the wrong directory. If the XML configuration files, however, have a different prefix from the deployment template `[appname]`, these files will not be copied into the `[appdir]/data/forge_output/` , `[appdir]/data/dgidx_output/`, and `[appdir]/data/dgraphs/*/dgraph_input/` directories. All processes will likely complete successfully, but any configuration information specified by these XML files, such as search interfaces, business rules, sort keys, etc. will be missing from the resulting MDEX Engine. To correct this problem, check the XML files located in `[appdir]/config/pipeline/` and make sure they have the correct prefix. Also check the directories mentioned above to make sure that these XML files are being properly copied.

MDEX Engine Fails to Start

If an MDEX Engine fails to start, check the log for the appropriate Dgraph in `[appdir]/logs/dgraphs/[dgraph]/[dgraph].log`. If the log indicates that the Dgraph failed to start because no record specifier was found, follow the steps in this document to create a unique record specifier property for you project.

Record Adapter Unable to Open File

Another common error may occur if a record adapter is unable to find or open a specified file for either input or output. In this case, the Forge error log (`[appdir]/logs/baseline/err.forge`) should specify which file or directory could not be found. To correct this problem, make sure the files or directories specified by the record adapters correspond to the directory structure established by the deployment template application. Note that this error may be masked if the "Require Data" property is not checked for a given input adapter, since Forge will only log a warning instead of a fatal error.

Related Links

Input record adapters on page 115
The record adapters load the source data.

Output record adapters on page 116
Output record adapters are often used to generate debug or state information. By default, the location to which this data is written will be overridden by the `--outputDir` flag.

Appendix A

EAC Development Toolkit

The EAC Development Toolkit provides a common set of objects, a standard and robust configuration file format and a lightweight controller implementation that developers can leverage in order to implement operational controller applications. The toolkit is designed to enable quick deployment, while providing complete flexibility for developers to extend and override any part of the implementation to create custom, project-specific functionality.

EAC Development Toolkit distribution and package contents

The EAC Development Toolkit is distributed as a set of JAR files bundled with the Deployment Template.

The toolkit consists of three JAR files and depends on two others that are distributed with this package. The following sections describe the JAR files. Details about classes and methods can be found in Javadoc distributed with the EAC Development Toolkit. These JAR files must be on the classpath of any application built using the EAC Development Toolkit.

`eacToolkit.jar`

This JAR contains the source and compiled class files for the core EAC Development Toolkit classes. These classes encompass core EAC functionality, from which all component implementations extend. Included are low-level classes that access the EAC's central server via SOAP calls to its Web Service interface as well as higher level objects that wrap logic and data associated with hosts, components, scripts and utilities. In addition, this JAR includes the controller implementation used to load the Toolkit's application configuration file, and to invoke actions based on the configuration and the user's command line input.

`eacComponents.jar`

This JAR contains the source and compiled class files for common implementations of Oracle Endeca components. These classes extend core functionality in `eacToolkit.jar` and implement standard versions of Forge, Dgidx, Dgraph and other components of an Oracle Endeca deployment.

`eacHandlers.jar`

This JAR contains the source and compiled class files for parsing application configuration documents. In addition, the EAC Dev Toolkit's application configuration XML document format is defined by an XSD file packaged with this JAR. Finally, the JAR includes files required to register the schema and the toolkit's namespace with Spring, the framework used to load the toolkit's configuration.

`spring.jar`

The toolkit uses the Spring framework for configuration management.

`bsh-2.0b4.jar`

The toolkit uses BeanShell as the scripting language used by developers to write scripts in their application configuration documents.

EAC Development Toolkit usage

The EAC Development Toolkit provides a library of classes that developers can use to develop and configure EAC scripts.

Classes in the library expose low level access to the EAC's web services and implement high level functionality common to many EAC scripts. Developers may implement applications by simply configuring functionality built in the toolkit or by extending the toolkit at any point to develop custom functionality.

This document discusses the toolkit's configuration file format, BeanShell scripting, command invocation and logging. This document does not provide a reference of the classes in the toolkit, or the functionality implemented in various objects and methods. Developers should refer to Javadoc or Java source files distributed with this package for details about the implementation.

Appendix B

Application Configuration File

The EAC toolkit uses an XML configuration file to define the elements that make up an application. In most deployments, this document will serve as the primary interface for developers and system administrators to configure, customize, and maintain a deployed application.

Spring framework

The EAC Development Toolkit uses the Spring Framework's Inversion of Control container to load an EAC application based on configuration specified in an XML document.

A great deal of functionality and flexibility is provided in Spring's IoC Container and in the default bean definition XML file handled by Spring's `XmlBeanDefinitionReader` class. For details about either of these, refer to Spring Framework documentation and JavaDoc.

The EAC Development Toolkit uses a customized document format and includes a schema and custom XML handlers to parse the custom document format. It uses Spring to convert this customized configuration metadata into a system ready for execution. Specifically, the toolkit uses Spring to load a set of objects that represent an EAC application with the configuration specified for each object in the configuration document.

XML schema

A customized document format is used to provide an intuitive configuration format for EAC script developers and system administrators.

However, this customization restricts the flexibility of the configuration document. The following sections describe elements available in the custom namespace defined by the `eacToolkit.xsd` XML schema. Each element name is followed by a brief description and an example configuration excerpt. For details, refer to the `eacToolkit.xsd` schema file distributed within the file `eacHandlers.jar`.

Related Links

Application elements on page 122
This section describes the application elements available in the custom namespace defined by the `eacToolkit.xsd` XML schema.

Hosts on page 122
This section describes the host element available in the custom namespace defined by the `eacToolkit.xsd` XML schema.

Application elements

This section describes the application elements available in the custom namespace defined by the `eacToolkit.xsd` XML schema.

For more details, refer to the `eacToolkit.xsd` schema file distributed within the file `eacHandlers.jar`.

Element	Description

`app`	This element defines the global application settings inherited by all other objects in the document, including application name, EAC central server host and port, data file prefix, the lock manager used by the application and whether or not SSL is enabled. In addition, this object defines global defaults for the working directory and the logs directory, which can be inherited or overridden by objects in the document.
	`<app appName="myApp" eacHost="devhost.company.com" eac¬ Port="8888"`
	`  dataPrefix="myApp" sslEnabled="false" lockManager="Lock¬ Manager" >`
	`  <working-dir>C:\Endeca\apps\myApp</working-dir>`
	`  <log-dir>./logs/baseline</log-dir>`
	`</app>`

`lock-manag¬ er`	This element defines a LockManager object used by the application to interact with the EAC's synchronization web service. Lock managers can be configured to release locks when a failure is encountered, ensuring that the system returns to a "neutral" state if a script or component fails. Multiple lock managers can be defined. `<lock-manager id="LockManager" releaseLocksOnFailure="true" />`

Hosts

This section describes the host element available in the custom namespace defined by the `eacToolkit.xsd` XML schema.

The `host` element defines a host associated with the application, including the `ID`, `hostname` and EAC agent port of the host. Multiple host elements can be defined.

```
<host id="ITLHost" hostName="itlhost.company.com" port="8888" />
```

Components

This section describes the component elements available in the custom namespace defined by the `eacToolkit.xsd` XML schema.

For more details, refer to the `eacToolkit.xsd` schema file distributed within the file `eacHandlers.jar`.

Element	Description

`forge`	This element defines a Forge component, including attributes that define the functionality of the Forge process as well as custom properties and directories used to configure the functionality of the Forge object's methods. Multiple `forge` elements can be defined.
	`<forge id="Forge" host-id="ITLHost">`
	`  <properties>`
	`  <property name="numStateBackups" value="10" />`
	`  <property name="numLogBackups" value="10" />`
	`  </properties>`
	`  <directories>`
	`  <directory name="incomingDataDir">./data/incoming</di¬`
	`rectory>`
	`  <directory name="configDir">./data/complete_config</di¬`
	`rectory>`
	`  <directory name="wsTempDir">./data/web_stu¬`
	`dio_temp_dir</directory>`
	`  </directories>`
	`  <args>`
	`  <arg>-vw</arg>`
	`  </args>`
	`  <input-dir>./data/processing</input-dir>`
	`  <output-dir>./data/forge_output</output-dir>`
	`  <state-dir>./data/state</state-dir>`
	`  <temp-dir>./data/temp</temp-dir>`
	`  <num-partitions>1</num-partitions>`
	`  <pipeline-file>./data/processing/pipeline.epx</pipeline-`
	`file>`
	`</forge>`

`forge-cluster`	This element defines a Forge cluster, including a list of ID references to the Forge components that belong to this cluster. This object can be configured to distribute data to Forge servers serially or in parallel.
	`<forge-cluster id="ForgeCluster" getDataInParallel="true">`
	
	`  <forge ref="ForgeServer" />`
	`  <forge ref="ForgeClient1" />`
	`  <forge ref="ForgeClient2" />`
	`</forge-cluster>`

Element	Description

`dgidx`	This element defines a Dgidx component, including attributes that define the functionality of the Dgidx process as well as custom properties and directories used to configure the functionality of the Dgidx object's methods. Multiple `dgidx` elements can be defined.
	`<dgidx id="Dgidx" host-id="ITLHost">`
	`  <args>`
	`  <arg>-v</arg>`
	`  </args>`
	`  <input-dir>./data/forge_output</input-dir>`
	`  <output-dir>./data/dgidx_output</output-dir>`
	`  <temp-dir>./data/temp</temp-dir>`
	`  <run-aspell>true</run-aspell>`
	`</dgidx>`

`indexing-clus¬ter`	This element defines an indexing cluster, including a list of ID references to the Dgidx and Agidx components that belong to this cluster. This object can be configured to distribute data to indexing servers serially or in parallel.
	`<indexing-cluster id="IndexingCluster" getDataInParal¬lel="true">`
	`  <agidx ref="Agidx1" />`
	`  <dgidx ref="Dgidx1" />`
	`  <dgidx ref="Dgidx2" />`
	`</indexing-cluster>`

`agidx`	This element defines an Agidx component, including attributes that define the functionality of the Agidx process as well as custom properties and directories used to configure the functionality of the Agidx object's methods. Multiple `agidx` elements can be defined.
	`<agidx id="Agidx1" host-id="ITLHost">`
	`  <properties>`
	`  <property name="agidxGroup" value="A" />`
	`  <property name="numLogBackups" value="10" />`
	`  <property name="numIndexBackups" value="3" />`
	`  </properties>`
	`  <args>`
	`  <arg>-v</arg>`
	`  </args>`
	`  <input-prefix-list>`
	`  <input-prefix>`
	`  ./data/dgidxs/Dgidx1/dgidx_output/project_name-part0`
	`  </input-prefix>`
	`  <input-prefix>`
	`  ./data/dgidxs/Dgidx2/dgidx_output/project_name-part1`
	`  </input-prefix>`
	`  </input-prefix-list>`
	`  <log-dir>./logs/agidxs/Agidx1</log-dir>`
	`  <output-dir>./data/agidx_output</output-dir>`
	`</agidx>`

Element	Description

dgraph	This element defines a Dgraph component, including attributes that define the functionality of the Dgraph process as well as custom properties and directories used to configure the functionality of the Dgraph object's methods. Multiple dgraph elements can be defined. Each dgraph element inherits, and potentially overrides, configuration specified in the dgraph-defaults element (see below).
	`<dgraph id="Dgraph1" host-id="MDEXHost" port="15000">`
	`  <properties>`
	`  <property name="restartGroup" value="A" />`
	`  <property name="updateGroup" value="a" />`
	`  </properties>`
	`  <log-dir>./logs/dgraphs/Dgraph1</log-dir>`
	`  <input-dir>./data/dgraphs/Dgraph1/dgraph_input</input-dir>`
	`  <update-dir>./data/dgraphs/Dgraph1/dgraph_input/up¬dates</update-dir>`
	`  </dgraph>`

dgraph-defaults	This element defines the default settings inherited by all dgraph elements specified in the document. This enables a single point of configuration for common Dgraph configuration such as command line arguments, and script directory configuration. Only one dgraph-defaults element can be defined.
	`<dgraph-defaults>`
	`  <properties>`
	`  <property name="srcIndexDir" value="./data/dgidx_out¬put" />`
	`  <property name="srcIndexHostId" value="ITLHost" />`
	`  <property name="numLogBackups" value="10" />`
	`  </properties>`
	`  <directories>`
	`  <directory name="localIndexDir">`
	`  ./data/dgraphs/local_dgraph_input`
	`  </directory>`
	`  </directories>`
	`  <args>`
	`  <arg>--threads</arg>`
	`  <arg>2</arg>`
	`  <arg>--spl</arg>`
	`  <arg>--dym</arg>`
	`  </args>`
	`  <startup-timeout>120</startup-timeout>`
	`</dgraph-defaults>`

dgraph-cluster	This element defines a Dgraph cluster, including a list of ID references to the Dgraph components that belong to this cluster. This object can be configured to distribute data to Dgraph servers serially or in parallel.
	`<dgraph-cluster id="DgraphCluster" getDataInParal¬lel="true">`
	`  <dgraph ref="Dgraph1" />`
	`  <dgraph ref="Dgraph2" />`
	`</dgraph-cluster>`

Element	Description

`agraph`	This element defines an Agraph component, including attributes that define the functionality of the Agraph process as well as custom properties and directories used to configure the functionality of the Agraph object's methods. Multiple `agraph` elements can be defined. Each `agraph` element inherits, and potentially overrides, configuration specified in the `agraph-defaults` element (see below).
	`<agraph id="Agraph1" host-id="MDEXHost" port="14000">`
	`  <properties>`
	`  <property name="restartGroup" value="A" />`
	`  </properties>`
	`  <dgraph-children>`
	`  <dgraph-child>Dgraph1</dgraph-child>`
	`  <dgraph-child>Dgraph2</dgraph-child>`
	`  </dgraph-children>`
	`  <log-dir>./logs/agraphs/Agraph1</log-dir>`
	`  <input-dir>./data/agraphs/Agraph1/agraph_input</input-dir>`
	`  </agraph>`

`agraph-defaults`	This element defines the default settings inherited by all `agraph` elements specified in the document. This enables a single point of configuration for common Agraph configuration such as command line arguments, and script directory configuration. Only one `agraph-defaults` element can be defined.
	`<agraph-defaults>`
	`  <properties>`
	`  <property name="srcIndexDir" value="./data/agidx_out¬put" />`
	`  <property name="srcIndexHostId" value="ITLHost" />`
	`  <property name="numLogBackups" value="10" />`
	`  </properties>`
	`  <directories>`
	`  <directory name="localIndexDir">`
	`  ./data/agraphs/local_agraph_input`
	`  </directory>`
	`  </directories>`
	`  <args>`
	`  <arg>--no-partial</arg>`
	`  </args>`
	`  <startup-timeout>120</startup-timeout>`
	`</agraph-defaults>`

`agraph-cluster`	This element defines an Agraph cluster, including a list of ID references to the Agraph and Dgraph components that belong to this cluster. This object can be configured to distribute data to graph servers serially or in parallel.
	`<agraph-cluster id="AgraphCluster" getDataInParal¬lel="true">`
	`  <agraph ref="Agraph1" />`
	`  <agraph ref="Agraph2" />`
	`  <dgraph ref="Dgraph1" />`
	`  <dgraph ref="Dgraph2" />`
	`</agraph-cluster>`

Element	Description

`logserver`	This element defines a LogServer component, including attributes that define the functionality of the LogServer process as well as custom properties and directories used to configure the functionality of the LogServer object's methods. Multiple `logserver` elements can be defined.
	`<logserver id="LogServer" host-id="ITLHost" port="15002">`
	
	`  <properties>`
	`  <property name="numLogBackups" value="10" />`
	`  <property name="targetReportGenDir" value="./reports/in¬`
	`put" />`
	`  <property name="targetReportGenHostId" value="ITLHost"`
	`/>`
	`  </properties>`
	`  <log-dir>./logs/logserver</log-dir>`
	`  <output-dir>./logs/logserver_output</output-dir>`
	`  <startup-timeout>120</startup-timeout>`
	`  <gzip>false</gzip>`
	`</logserver>`

`report-genera¬`
`tor`
	This element defines a ReportGenerator component, including attributes that define the functionality of the ReportGenerator process as well as custom properties and directories used to configure the functionality of the ReportGenerator object's methods. Multiple `report-generator` elements can be defined.
	`<report-generator id="WeeklyReportGenerator" host-`
	`id="ITLHost">`
	`  <properties>`
	`  <property name="webStudioReportDir"`
	`value="C:\Endeca\MDEXEngine\workspace/reports/MyApp" />`
	`  <property name="webStudioReportHostId" value="ITLHost"`
	`/>`
	`  </properties>`
	`  <log-dir>./logs/report_generators/WeeklyReportGenera¬`
	`tor</log-dir>`
	`  <input-dir>./reports/input</input-dir>`
	`  <output-file>./reports/weekly/report.xml</output-file>`
	`  <stylesheet-file>`
	`  ./config/report_templates/tools_report_stylesheet.xsl`
	
	`  </stylesheet-file>`
	`  <settings-file>`
	`  ./config/report_templates/report_settings.xml`
	`  </settings-file>`
	`  <time-range>LastWeek</time-range>`
	`  <time-series>Daily</time-series>`
	`  <charts-enabled>true</charts-enabled>`
	`</report-generator>`

`custom-compo¬` `nent`	This element defines a custom component, including custom properties and directories used to configure the functionality of the custom component object's methods. Multiple `custom-component` elements can be defined, though each

Element	Description
	must specify the name of the implemented class that extends `com.Endeca.soleng.eac.toolkit.component.CustomComponent`. `<custom-component id="ConfigManager" host-id="ITLHost" class="com.Endeca.soleng.eac.toolkit.component.ConfigManagerComponent"> <properties> <property name="webStudioHost" value="sshusteff-lt1.ne.Endeca.com" /> <property name="webStudioPort" value="8888" /> <property name="webStudioMaintainedFile1" value="thesaurus.xml" /> <property name="webStudioMaintainedFile2" value="merch_rule_group_default.xml" /> <property name="webStudioMaintainedFile3" value="merch_rule_group_default_redirects.xml" /> </properties> <directories> <directory name="mergedConfigDir">./data/complete_config</directory> <directory name="devStudioConfigDir">./config/pipeline</directory> <directory name="webStudioConfigDir"> ./data/web_studio/config </directory> <directory name="webStudioDgraphConfigDir"> ./data/web_studio/dgraph_config </directory> <directory name="webStudioTempDir"> ./data/web_studio_temp_dir </directory> </directories> </custom-component>`

Related Links

Display component status on page 139
The controller provides a convenience method for displaying the status of all components defined in the configuration document.

Utilities

This section describes the utility elements available in the custom namespace defined by the `eacToolkit.xsd` XML schema.

For more details, refer to the `eacToolkit.xsd` schema file distributed within the file `eacHandlers.jar`.

Element	Description
`copy`	This element defines a copy utility invocation, including the source and destination and whether or not the source pattern should be interpreted recursively. Multiple `copy` elements can be defined. `<copy id="CopyData" src-host-id="ITLHost" dest-host-id="ITLHost"`

Element	Description
	recursive="true" > <src>./data/incoming/*.txt</src> <dest>./data/processing/</dest> </copy>
shell	This element defines a shell utility invocation, including the command to execute and the host on which the command will be executed. Multiple shell elements can be defined. <shell id="ProcessData" host-id="ITLHost" > <command>perl procesDataFiles.pl ./data/incoming/data.txt</com¬ mand> </shell>
backup	This element defines a backup utility invocation, including the directory to archive, how many archives should be saved and whether the archive should copy or move the source directory. Multiple backup elements can be defined. <backup id="ArchiveState" host-id="ITLHost" move="true" num- backups="5"> <dir>C:\Endeca\apps\myApp\data\state</dir> </backup>
rollback	This element defines a rollback utility invocation, including the directory whose archive should be recovered. Multiple rollback elements can be defined. <rollback id="RollbackState" host-id="ITLHost"> <dir>./data/state</dir> </rollback>

Customization/extension within the toolkit's schema

Most configuration tasks are performed by simply altering an element in the configuration document, by adding elements to the document, or by removing elements from the configuration.

These three actions enable users to alter the behavior of objects in their application, change which objects make up their application and change the way scripts acts on the objects in their application.

In addition to these simple actions, users can customize the behavior of objects in their application or create new objects while continuing to use the EAC development toolkit's XML configuration document format. The following are examples of customization that are possible within the constructs of the XML schema defined in the eacToolkit.xsd schema file.

Implement a custom component

Users can develop new custom components by extending the class com.Endeca.soleng.eac.toolkit.component.CustomComponent. This class and its associated XML element allow any number of properties and directories to be specified and accessed by methods in the object. This customization method may be appropriate for cases where functionality needs to be developed that is not directly associated with an Oracle Endeca process.

Extend an existing object

Users can implement customizations on top of existing objects by creating a new class that extends an object in the toolkit. Most elements in the configuration document (with the notable exception of the "app" element, which specifies global configuration, but does not directly correspond to an object instance) can specify a class attribute to override the default class associated with each element. For example, a user could implement a `MyForgeComponent` class by extending the toolkit's `ForgeCom¬ponent` class.

```
package com.Endeca.soleng.eac.toolkit.component;

import java.util.logging.Logger;

import com.Endeca.soleng.eac.toolkit.exception.AppConfigurationException;
import com.Endeca.soleng.eac.toolkit.exception.EacCommunicationException;
import com.Endeca.soleng.eac.toolkit.exception.EacComponentControlException;

public class MyForgeComponent extends ForgeComponent
{
  private static Logger log =
    Logger.getLogger(MyForgeComponent.class.getName());

  protected void getIncomingData() throws AppConfigurationException,
      EacCommunicationException, EacComponentControlException,
      InterruptedException
  {

  // custom data retrieval implementation

  }
}
```

The new class can override method functionality to customize the behavior of the object. As long as the new object does not require configuration elements unknown to the `ForgeComponent` from which it inherits, it can continue to use the forge element in the XML document to specify object configuration.

```
<forge class="com.Endeca.soleng.eac.toolkit.component.MyForgeComponent"
  id="CustomForge" host-id="ITLHost">
...
</forge>
```

Implement custom functionality in BeanShell scripts

Users can implement custom functionality by writing new code in the XML document in new or existing BeanShell scripts. This form of customization can be used to add new functionality or to override functionality that is built in to toolkit objects. While this customization approach is very flexible, it can become unwieldy and hard to maintain and debug if a large amount of custom code needs to be written.

Customization/extension beyond the toolkit's schema

Customization approaches within the existing schema will be sufficient for the majority of applications, but some developers will require even greater flexibility than can be supported by the XML document exposed by the toolkit.

This type of customization can still be achieved, by switching out of the default `eacToolkit` namespace in the XML document and leveraging the highly flexible and extensible Spring Framework bean definition format.

As an example, a developer might implement a new class, `PlainOldJavaObject`, which needs to be loaded and accessed by EAC scripts. If the object is implemented, compiled and added to the classpath, it can be loaded based on configuration in the XML document by specifying its configuration using the "`spr`" namespace.

```
<spr:bean id="MyPOJO" class="com.company.PlainOldJavaObject">
  <spr:constructor-arg>true</spr:constructor-arg>
  <spr:property name="Field1" value="StrValue" />
  <spr:property name="Map1">
    <spr:map>
      <spr:key>one</spr:key>
      <spr:value>1</spr:value>
      <spr:key>two</spr:key>
      <spr:value>2</spr:value>
    </spr:map>
  </spr:property>
</spr:bean>
```

Appendix C

BeanShell Scripting

The EAC Development Toolkit uses BeanShell to interpret and execute scripts defined in the app configuration document. The following sections describe the toolkit's use of the BeanShell interpreter and provide sample BeanShell script excerpts.

Script implementation

In the toolkit, the `com.Endeca.soleng.eac.toolkit.script.Script` class implements scripts.

This class exposes simple execution logic that either uses a BeanShell interpreter to execute the script specified in the configuration file or, if no BeanShell script is specified in the script's configuration, uses the Script object's `scriptImplementation` method. By default, the `scriptImplementation` method has no logic and must be overridden by an extending class to take any action. This allows developers to leverage BeanShell to implement their scripts or to extend the Script object, overriding and implementing the `scriptImplementation` method.

By implementing scripts as BeanShell scripts configured in the toolkit's XML configuration document, developers can quickly develop and adjust scripts, and system administrators can adjust script implementations without involving developers. The scripting language should be familiar to any Java developer, as it is a Java based scripting language that can interpret strict Java code (i.e. code that could be compiled as a Java class). BeanShell also provides a few flexibilities that are not available in Java; for example, BeanShell allows developers to import classes at any point in the script, rather than requiring all imports to be defined up front. In addition, BeanShell allows variables to be declared without type specification.

Note: For details about BeanShell and ways in which it differs from Java, developers should refer to BeanShell documentation and Javadoc.

BeanShell interpreter environment

The most common use of BeanShell scripts in the EAC Development Toolkit is to orchestrate the elements defined in the application configuration document.

More precisely, BeanShell scripts are used to orchestrate the execution of methods on the objects that are loaded from the configuration document. In order to enable this, when the toolkit constructs the BeanShell Interpreter environment, it sets internal variables associated with each element defined

in the configuration document. While additional variables can be declared at any point in a script, this allows scripts to immediately act on objects defined in the document without declaring any variables.

Take, for example, the following configuration document:

```
<app appName="myApp" eacHost="devhost.company.com" eacPort="8888"
    dataPrefix="myApp" sslEnabled="false" lockManager="LockManager" >
  <working-dir>C:\Endeca\apps\myApp</working-dir>
  <log-dir>./logs/baseline</log-dir>
</app>

<host id="ITLHost" hostName="itlhost.company.com" port="8888" />

<copy id="CopyData" src-host-id="ITLHost" dest-host-id="ITLHost"
    recursive="true" >
  <src>./data/incoming/*.txt</src>
  <dest>./data/processing/</dest>
</copy>

<backup id="ArchiveState" host-id="ITLHost" move="true" num-backups="5">
  <dir>C:\Endeca\apps\myApp\data\state</dir>
</backup>

<forge id="Forge" host-id="ITLHost">
  <properties>
    <property name="numStateBackups" value="10" />
    <property name="numLogBackups" value="10" />
  </properties>
  <directories>
    <directory name="incomingDataDir">./data/incoming</directory>
    <directory name="configDir">./data/processing</directory>
  </directories>
  <args>
    <arg>-vw</arg>
  </args>
  <input-dir>./data/processing</input-dir>
  <output-dir>./data/forge_output</output-dir>
  <state-dir>./data/state</state-dir>
  <temp-dir>./data/temp</temp-dir>
  <num-partitions>1</num-partitions>
  <pipeline-file>./data/processing/pipeline.epx</pipeline-file>
</forge>
```

A BeanShell script defined in this document will have five variables immediately available for use: `ITLHost`, `CopyData`, `ArchiveState`, `Forge`, and `log`. Note that there is no variable associated with the `app` element in the document, as this element does not correspond to an object instance. Each of the other elements is instantiated, loaded with data based on its configuration and made available in the BeanShell interpreter. In addition, a special variable called log is always created for each script with a `java.util.Logger` instance.

A simple BeanShell script can then be written without importing a single class or instantiating a single variable.

```
<script id="SimpleForgeScript">
  <bean-shell-script>
    <![CDATA[
  log.info("Starting Forge script.");
  CopyData.run();
  Forge.run();
  ArchiveState.setNumBackups(Forge.getProperty("numStateBackups"));
  ArchiveState.run();
```

```
  log.info("Finished Forge script.");
    ]]>
  </bean-shell-script>
</script>
```

In addition to exposing objects defined in the document, the toolkit imports and executes a default script each time a BeanShell script is invoked. If a file named "`beanshell.imports`" is successfully loaded as a classpath resource, that file is executed each time a BeanShell script is executed. This allows a default set of imports to be defined. For example, the following default file imports all of the classes in the toolkit, exposing them to BeanShell scripts:

```
import com.Endeca.soleng.eac.toolkit.*;
import com.Endeca.soleng.eac.toolkit.application.*;
import com.Endeca.soleng.eac.toolkit.base.*;
import com.Endeca.soleng.eac.toolkit.component.*;
import com.Endeca.soleng.eac.toolkit.component.cluster.*;
import com.Endeca.soleng.eac.toolkit.exception.*;
import com.Endeca.soleng.eac.toolkit.host.*;
import com.Endeca.soleng.eac.toolkit.logging.*;
import com.Endeca.soleng.eac.toolkit.script.*;
import com.Endeca.soleng.eac.toolkit.utility.*;
import com.Endeca.soleng.eac.toolkit.utility.perl.*;
import com.Endeca.soleng.eac.toolkit.utility.webstudio.*;
import com.Endeca.soleng.eac.toolkit.utility.wget.*;
import com.Endeca.soleng.eac.toolkit.utils.*;
```

About implementing logic in BeanShell

BeanShell scripts will typically be used to orchestrate method execution for objects defined in the configuration document.

However, scripts can also implement logic, instantiating objects to provide a simple point of extension for developers to implement new logic without compiling additional Java classes.

For example, the following script excerpt demonstrates how a method can be defined and referenced in a script:

```
<script id="Status">
    <bean-shell-script>
      <![CDATA[

        // define function for printing component status
        import com.Endeca.soleng.eac.toolkit.component.Component;
        void printStatus( Component component ) {
          log.info(component.getAppName() + "." +
            component.getElementId() + ": " +
            component.getStatus().toString() );
        }

        // print status of forge, dgidx, logserver
        printStatus( Forge );
        printStatus( Dgidx );
        printStatus( LogServer );

        // print status for dgraph cluster
        dgraphs = DgraphCluster.getDgraphs().iterator();
        while( dgraphs.hasNext() ) {
          printStatus( dgraphs.next() );
        }
```

```
    ]]>
  </bean-shell-script>
</script>
```

Appendix D

Command Invocation

The toolkit provides a simple interface for invoking commands from the command line.

Invoke a method on an object

By default, the controller tries to invoke a method called "`run`" with no arguments on the specified object.

The following simple command invokes the run method on the BaselineUpdate script object:

```
java Controller --app-config AppConfig.xml BaselineUpdate
```

If a method name is specified, the controller looks for a method with that name on the specified object and invokes it. For example, the following command executes the `applyIndex` method on the `DgraphCluster` object:

```
java Controller --app-config AppConfig.xml DgraphCluster applyIndex
```

In addition to no-argument method invocation, the controller allows any number of String arguments to be passed to a method. The following example shows the `releaseLock` method being invoked on the LockManager object with the single String argument "`update_lock`" specifying the name of the lock to release:

```
java Controller --app-config AppConfig.xml LockManager releaseLock
  update_lock
```

Identify available methods

In order to help users identify the objects and methods available for invocation, the controller provides a help argument that can be called to list all available objects or methods available on an object.

If specified with an app configuration document, the help command displays usage and available objects:

```
java Controller --app-config AppConfig.xml --help

...

The following objects are defined in document 'AppConfig.xml':
[To see methods available for an object, use the --help command line argument
 and specify the name of the object.]
```

```
  [com.Endeca.soleng.eac.toolkit.base.LockManager]
    LockManager
  [com.Endeca.soleng.eac.toolkit.component.ConfigManagerComponent]
    ConfigManager
  [com.Endeca.soleng.eac.toolkit.component.DgidxComponent]
    Dgidx
  [com.Endeca.soleng.eac.toolkit.component.DgraphComponent]
    Dgraph1
    Dgraph2
  [com.Endeca.soleng.eac.toolkit.component.ForgeComponent]
    Forge
    PartialForge
  [com.Endeca.soleng.eac.toolkit.component.LogServerComponent]
    LogServer
  [com.Endeca.soleng.eac.toolkit.component.ReportGeneratorComponent]
    WeeklyReportGenerator
    DailyReportGenerator
  [com.Endeca.soleng.eac.toolkit.component.cluster.DgraphCluster]
    DgraphCluster
  [com.Endeca.soleng.eac.toolkit.host.Host]
    ITLHost
    MDEXHost
  [com.Endeca.soleng.eac.toolkit.script.Script]
    BaselineUpdate
    DistributeIndexAndApply
    PartialUpdate
    DistributePartialsAndApply
    ConfigUpdate
```

The name of each object loaded from the configuration document is printed along with the object's class. To identify the available methods, the help command can be invoked again with the name of an object in the document:

```
java Controller --app-config AppConfig.xml --help DgraphCluster
...

The following methods are available for object 'DgraphCluster':
[Excluded: private, static and abstract methods; methods inherited from
Object; methods with names that start with 'get', 'set' or 'is'. For details,
 refer to Javadoc for class com.Endeca.soleng.eac.toolkit.component.clus¬
ter.DgraphCluster.]

  start(), stop(), removeDefinition(), updateDefinition(), cleanDirs(),
  applyIndex(), applyPartialUpdates(), applyConfigUpdate(),
  cleanLocalIndexDirs(), cleanLocalPartialsDirs(),
  cleanLocalDgraphConfigDirs(), copyIndexToDgraphServers(),
  copyPartialUpdateToDgraphServers(),
  copyCumulativePartialUpdatesToDgraphServers(),
  copyDgraphConfigToDgraphServers(),  addDgraph(DgraphComponent)
```

Note that not all methods defined for the class `com.Endeca.soleng.eac.toolkit.compo¬ nent.cluster.DgraphCluster` are displayed. As the displayed message notes, methods declared as private, static or abstract are excluded, as are methods inherited from Object, getters and setters, and a few reserved methods that are known not to be useful from the command line. These restrictions are intended to make the output of this help command as useful as possible, but there are likely to be cases when developers will need to refer to Javadoc to find methods that are not displayed using the help command.

Update application definition

By default, the controller will test the application definition in the configuration document against the provisioned definition in the EAC and update EAC provisioning if the definition in the document has changed.

This will happen by default any time any method is invoked on the command line.

System administrators may find it useful to update the definition without invoking a method. To facilitate this, a flag has been provided to perform the described definition update and exit.

```
java Controller --app-config AppConfig.xml --update-definition
```

In addition, there may be a need to invoke a method without testing the application definition. This can be accomplished by using an alternate command line argument:

```
java Controller --app-config AppConfig.xml --skip-definition
  BaselineUpdate
```

Remove an application

The controller provides a convenience method for removing an application from the EAC's central store.

When invoked, this action checks whether the application loaded from the configuration document is defined in the EAC. If it is, all active components are forced to stop and the application's definition is completely removed from the EAC.

```
java Controller --remove-app --app-config AppConfig.xml
```

Display component status

The controller provides a convenience method for displaying the status of all components defined in the configuration document.

When the following method is invoked, the controller iterates over all defined components, querying the EAC for the status of each one and printing it.

```
java Controller --print-status --app-config AppConfig.xml
```

Related Links

Components on page 123

This section describes the component elements available in the custom namespace defined by the `eacToolkit.xsd` XML schema.

Index

F

G

I

L

M

O

P

R

S

U

V

W

X