Oracle Endeca Commerce

Tools and Frameworks Deployment Template Usage Guide

Version 3.1.0 • July 2012

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

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

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

About this guide

This guide describes how to configure, run, and customize the Deployment Template that is included with Tools and Frameworks.

The Deployment Template is a utility that you run to create a new Endeca application with the complete directory structure required for deployment, including Endeca Application Controller (EAC) control scripts, configuration files, and batch files or shell scripts that wrap common script functionality.

Some scripts created by the Deployment Template are documented in the *Assembler Application Developer's Guide*, rather than this guide, because the scripts are very closely associated with Assembler features. Similarly, some scripts are documented in the *Oracle Endeca Commerce Administrator's Guide* because the scripts are very closely associated with administrative tasks such as backing up or restoring site configuration.

Who should use this guide

This guide is for developers or administrators who create and maintain Oracle applications using the Deployment Template.

Conventions used in this guide

This guide uses the following typographical conventions:

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

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

Contacting Oracle Support

Oracle Support provides registered users with important information regarding Oracle Endeca software, implementation questions, product and solution help, as well as overall news and updates.

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

Chapter 1

Deploying and initializing an EAC Application

This section describes how to deploy and initialize an EAC application using the Deployment Template.

Deployment prerequisites

You must have installed Tools and Frameworks on the machine running the EAC Central Server (part of the Platform Services package) and set environment variables used by the Oracle Endeca software (including `ENDECA_ROOT`).

About deploying EAC applications

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

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

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

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

Deploying and initializing an EAC application

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

To deploy an EAC application on Windows:

1. Start a command prompt (on Windows) or a shell (on UNIX).
2. Navigate to `<installation path>\ToolsAndFrameworks\<version>\deployment_template\bin` or the equivalent path on UNIX.
3. From the `bin` directory, run the `deploy` script.
   For example, on Windows:
   ```
   C:\Endeca\ToolsAndFrameworks\3.1.0\deployment_template\bin>deploy
   ```
4. If the path to the Platform Services installation is correct, press `Enter`.
   (The template identifies the location and version of your Platform Services installation based on the `ENDECA_ROOT` environment variable. If the information presented by the installer does not match the version or location of the software you plan to use for the deployment, stop the installation, reset your `ENDECA_ROOT` environment variable, and start again. Note 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.)
5. Specify a short name for the application.
   The name should consist of lower- or uppercase letters, or digits between zero and nine.
6. Specify the full path into which your application should be deployed.
   This directory must already exist. The `deploy` script creates a folder inside of the deployment directory with the name of your application and the application directory structure.
   For example, if your application name is `MyApp`, and you specify the deployment directory as `C:\Endeca\apps`, the `deploy` script installs the template for your application into `C:\Endeca\apps\MyApp`.
7. Specify the port number of the EAC Central Server.
   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.
8. Specify the port number of Oracle Endeca Workbench, or press **Enter** to accept the default of 8006.
9. Specify the port number of the Live Dgraph, or press **Enter** to accept the default of 15000.
10. Specify the port number of the Authoring Dgraph, or press **Enter** to accept the default of 15002.
11. Specify the port number of the Log Server, or press **Enter** to accept the default of 15010.
    If the application directory already exists, the `deploy` script time stamps and archives the existing directory to avoid accidental loss of data.
12. Navigate to the `control` directory of the newly deployed application.
    This is located under your application directory. For example: `C:\Endeca\apps\<app name>\control`.
13. From the `control` directory, run the `initialize_services` script.
    - On Windows:
      ```
      [appdir]\control\initialize_services.bat
      ```
    - On UNIX:
      ```
      [appdir]/control/initialize_services.sh
      ```

    The script initializes 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 application configuration

file to the EAC. 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`.

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

If no script customization is required, the application is now ready for use. Go on to *Managing Data Operations* on page 53.

However, if you need to configure an EAC application (the scripts and files) to reflect your environment and data processing requirements, go on to *Configuring an EAC Application* on page 19 before *Managing Data Operations* on page 53.

Configuring automated/file-based deployment

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

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

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

```
<install app-name="Discover">
  <deployment-path>/localdisk/endeca/apps</deployment-path>
  <base-module type="dgraph" />
  <options>
    <option name="eac-port">8888</option>
    <option name="workbench-port">8006</option>
    <option name="dgraph1Port">15000</option>
    <option name="authoringDgraphPort">15002</option>
    <option name="logserverPort">15010</option>
  </options>
</install>
```

To configure automated/file-based deployment:

1. Start a text editor, create a new text file, and copy/paste the example above.
2. If necessary, modify the default port values for the EAC Central Server, Workbench, Live Dgraph, Authoring Dgraph, and the Log Server to new values.
3. Save and close the file.
4. Run the `deploy` script and specify the `--install-config` flag and the location of the deployment configuration file.
   The following example specifies the deployment descriptor (`deploy.xml`) for a version of the Discover Electronics reference application, then the `--install-config` flag with an argument to the deployment configuration file (`pci-app-install-config.xml`):

   ```
   ./deploy.sh --app /localdisk/endeca/ToolsAndFrameworks/*/reference/dis¬
   cover-data-pci/deploy.xml --install-config /localdisk/infrontSetup¬
   Scripts/config/pci-app-install-config.xml --no-prompt
   ```

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

Modifying the template files to support custom applications

This section provides information about deploying custom applications.

Custom application descriptors

The Deployment Template deploys new applications based on application descriptor XML documents. The documents describe the directory structure associated with an application as well as the files to distribute during the deployment process.

By default, the Deployment Template ships with application descriptor files named `base_descriptor.xml` located in `<installation path>\ToolsAndFrameworks\<version>\deployment_template\app-templates`.

This document describes the directory structure of the deployment as well as the copying that is done during the deployment to distribute files into the new directories. Additionally, this document describes whether files are associated with a Windows or UNIX deployment, and whether copied files should be updated to replace tokens in the format @@TOKEN_NAME@@ with text strings specified to the installer.

The following tokens are handled by the base descriptor:

- @@WORKBENCH_PORT@@ - Oracle Endeca Workbench port.
- @@DGRAPH_1_PORT @@ - Live Dgraph port.
- @@AUTHORING_DGRAPH_PORT @@ - Authoring Dgraph port.
- @@LOGSERVER_PORT@@ - Log Server port.

The following tokens are handled by the Deployment Template:

- @@EAC_PORT@@ - EAC Central Server port.
- @@HOST@@ - Hostname of the server on which the deploy script is invoked.
- @@PROJECT_DIR@@ - Absolute path of the target deployment directory.
- @@PROJECT_NAME@@ - Name of the application to deploy.
- @@ENDECA_ROOT@@ - Absolute path of the ENDECA_ROOT environment variable.
- @@SCRIPT_SUFFIX@@ - ".bat" for Windows, ".sh" for Linux installs.

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

If a project deviates from the Deployment Template directory structure, it may find it useful to create a custom application descriptor document, so that the default Deployment Template can continue to be used for application deployment.

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

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

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

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

```
<!--
  Deployment Template installer configuration file. This file defines the
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 configuration file discussed in previous sections may be used to specify the location of custom application descriptor documents in place of the `--app` command line argument to the installer.

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

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

Communicating with SSL-enabled Oracle Endeca components

The Deployment Template supports enabling SSL to communicate securely with the EAC Central Server and with the Content Acquisition System version 3.0.x and later. (Secure communication between the Deployment Template and CAS 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.

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

    # EAC Application Definition
    #
  -->
  <app appName="test" eacHost="localhost" eacPort="8888"
      dataPrefix="test" sslEnabled="true" lockManager="LockManager">
    <working-dir>${ENDECA_PROJECT_DIR}</working-dir>
    <log-dir>./logs</log-dir>
  </app>

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

    # Lock Manager - Used to set/remove/test flags and obtain/release locks

    #
  -->
  <lock-manager id="LockManager" releaseLocksOnFailure="true" />

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

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

-->
```

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:

```
Deployment Template: 3.1.0
```

Chapter 2

Configuring an EAC Application

This section provides an overview of the elements defined in AppConfig.xml.

About configuring an EAC application

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

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

There are two ways to configure an EAC application:

- Configure `AppConfig.xml` files. The simplest form of configuration consists of editing the `AppConfig.xml` configuration file and its associated configuration files to change the behavior of components or to add or remove components.

  This type of configuration includes the addition of removal of Dgraphs to the main cluster or even the creation of additional clusters. In addition, this category includes adjustment of process arguments (for example, adding a Java classpath for the Forge process in order to enable the use of a Java Manipulator), custom properties and directories (for example, changing the number of index archives that are stored on the indexing server).

- Change behavior of existing BeanShell scripts. Scripts are written in the Java scripting language BeanShell. Scripts are defined in the `AppConfig.xml` document and are interpreted at runtime by the BeanShell interpreter. This allows developers and system administrators to adjust the behavior of the baseline, partial, and configuration update scripts by simply modifying the configuration document.

About the application configuration files

The application configuration file `[appdir]/config/script/AppConfig.xml` and its associated files define the hosts, components, and scripts that make up an EAC application and the that orchestrate updates by executing the defined components.

The Deployment Template provides a single `AppConfig.xml` file that contains pointers to refer to other files that define distinct parts of an application, separate scripts from component provisioning, and are used for other purposes. The full set of application configuration files are as follows:

- `InitialSetup.xml` - Specifies scripts to perform initial setup tasks, such as uploading initial configuration to Workbench.
- `DataIngest.xml` - Specifies data processing scripts, including the baseline update script, partial update script, and the components to perform data processing such as Forge and Dgidx.
- `DgraphDefaults.xml` - Specifies default values that are inherited by all Dgraph components. These values include host IDs, data processing paths, and Dgraph flags.
- `AuthoringDgraphCluster.xml` - Specifies the Dgraphs used in the authoring environment and a script that pushes configuration from Workbench to each Dgraph in the authoring cluster.
- `LiveDgraphCluster.xml` - Specifies the Dgraphs used in the live environment and a script that pushes configuration from Workbench to each Dgraph in the live cluster.
- `WorkbenchConfig.xml` - Specifies the IFCR component, the Workbench Manager component, and a script that promotes content from the authoring environment to the live environment.
- `ReportGeneration.xml` - Specifies the hosts used for logging and report generations, and several scripts that produce log files at different time intervals.

In addition to these files, any number of `--app-config` arguments may be specified to the Controller class in the EAC development toolkit. All of the objects in the files will be read and processed and scripts can refer to components, hosts, or other scripts defined in other files.

Configuring the application configuration files

This topic guides you through the process of configuring an EAC application.

1. Edit the `AppConfig.xml` file in `[appdir]/config/script` to reflect the details of your environment. Specifically, set the following values:
   - Specify the `eacHost` and `eacPort` attributes of the app element with the correct host and port of the EAC Central Server.
   - Specify the `host` elements with the correct host name or names and EAC ports of all EAC Agents in your environment.
   - Specify the `WorkbenchManager` component with the correct host and port for Oracle Endeca Workbench.
2. If necessary, edit the `InitialSetup.xml` file in `[appdir]/config/script`. This file does not usually require any modifications.
3. Edit the `DataIngest.xml` file in `[appdir]/config/script` to reflect your data processing requirements. Specifically, ensure that the baseline update script and partial update script are correct and that the Forge and Dgidx components are correctly configured.
4. Edit the `DgraphDefaults.xml` file in `[appdir]/config/script` with the default values that are inherited by all Dgraph components in both the authoring cluster and live cluster.
5. Edit the `AuthoringDgraphCluster.xml` file in `[appdir]/config/script` to ensure the authoring Dgraph, the authoring cluster and post-startup script is correct for your environment.
6. Edit the `LiveDgraphCluster.xml` file in `[appdir]/config/script` to ensure the live Dgraph, the live cluster and post-startup script is correct for your environment.
7. Edit the `WorkbenchConfig.xml` file in `[appdir]/config/script` to ensure the Workbench Manager and IFCR components are correct for your environment.

8. If necessary, edit the `ReportGeneration.xml` file in `[appdir]/config/script`. This file does not usually require any modifications.

The following topics describe the components that you can define in the application configuration files.

Global application settings

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

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

```
<!–
########################################################################
# 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. This example shows a host defined to run CAS and a host to run the MDEX Engine.

```
<!–
########################################################################
# Servers/hosts
#
-->
  <host id="CASHost" 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.

CAS Server

The Deployment Template provides support for running CAS crawls with the CAS Server Component. A CAS Server component is implemented as a `custom-component`. You configure the component according to the output type of a crawl. The sections below describe the common configuration properties, the output-type configuration properties, and then provide examples for each output type including Record Store output, MDEX-compatible output, and record file output.

Note: The Deployment Template cannot create a new CAS crawl. You create a crawl using CAS and run it using the Deployment Template. For details about creating a crawl, see the *CAS Developer's Guide*.

The `custom-component` configuration properties

The `custom-component` configuration properties identify the CAS server in the Servers/hosts section of `AppConfig.xml`. The properties are defined as follows:

- `id` - Assigns a unique ID to a specific CAS Server.
- `host-id` - Points back to the `id` attribute of the `host` global configuration element.
- `class` - Specifies the class that implements the `ContentAcquisitionServerComponent`. Specify class="`com.endeca.eac.toolkit.component.cas.ContentAcquisitionServer¬Component`".

Common configuration properties

The common configuration properties describe the host and port running CAS. The properties are defined as follows:

- `casHost` - Host name of the server on which the Content Acquisition System is running.

- `casPort` - Port on which the Endeca CAS Service listens. If the application is running in SSL mode, the `casPort` is the SSL port of the Endeca CAS Service The port number must match the `com.endeca.cas.port` value that is used in the CAS Service configuration script. Or, if the Endeca CAS Service is configured for SSL, then the port number must match `com.ende¬ ca.cas.ssl.port` value. The configuration script is in `<install path>\CAS\workspace\conf\jetty.xml`.

Configuration properties specific to Record Store output

There are no additional configuration properties required for crawls that write to a Record Store instance. Only the `custom-component` and common configuration properties are required.

Example

This example CAS Server component is configured for Record Store output:

```
<!--
########################################################################
# Content Acquisition System Server
#
<custom-component id="CAS" host-id="CASHost" class="com.endeca.eac.toolk¬
it.component.cas.ContentAcquisitionServerComponent">
   <properties>
      <property name="casHost" value="localhost" />
      <property name="casPort" value="8500" />
    </properties>
</custom-component>
-->
```

Configuration properties specific to record file output

The configuration properties are defined as follows:

- `casCrawlFullOutputDestDir` - Indicates the destination directory to which the crawl output file will be copied after a baseline crawl. Note that this is not the directory to which the CAS crawl writes its output; that output directory is set as part of the crawl configuration.
- `casCrawlIncrementalOutputDestDir` - Indicates the destination directory to which the crawl output file will be copied after an incremental crawl. As with the previous property, this is not the directory to which the CAS crawl writes its output. If you run incremental crawls, the default settings assume that the output format will be compressed binary files.
- `casCrawlOutputDestHost` - Indicates the ID of the host on which the destination directories (specified by the previous two properties) reside.

Example

This example CAS Server component is configured for a record file output:

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

    # Content Acquisition System Server
    #
  -->
  <custom-component id="CAS" host-id="CASHost" class="com.ende¬
ca.soleng.eac.toolkit.component.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>
```

Forges

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

If necessary, you can define a Forge cluster component to apply actions to an entire cluster of Forges, rather than manually iterating over a number of Forges. You could use this feature to run several instances of Forge in parallel to process large joins.

In addition, the object contains logic associated with executing Forges in parallel based on Forge groups, which are described below. Multiple Forge clusters can be defined, with no restriction around which Forges belong to each cluster or how many clusters a Forge belongs to.

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

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

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

- `numLogBackups` - Number of log directory backups to store.
- `numStateBackups` - Number of autogen state directory backups to store.
- `numPartialsBackups` - Number of cumulative partials directory backups to store. It is recommended that you increase the default value of 5. The reason is that the files in the updates directory for the Dgraph are automatically deleted after partials are applied to the Dgraph. The number you choose depends on how often you run partial updates and how many copies you want to keep.
- `incomingDataHost` - Host to which source data files are extracted.
- `incomingDataDir` - Directory to which source data files are extracted.
- `incomingDataFileName` - Filename of the source data files that are extracted.
- `configHost` - Host from which configuration files and dimensions are retrieved for Forge to process.
- `configDir` - Directory from which configuration files and dimensions are retrieved for Forge to process.
- `cumulativePartialsDir` - Directory where partial updates are accumulated between baseline updates.

- `wsTempDir` - Temp Oracle 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 necessary, you can define a Dgidx cluser to apply actions to an entire cluster of Dgidxs, rather than manually iterating over a number of Dgidxs. In addition, the object contains logic associated with executing Dgidxs in parallel based on Dgidx groups, which are described below. Multiple indexing clusters can be defined, with no restriction around which Dgidx belongs to each cluster or how many clusters a Dgidx belongs to.

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

```
<!--
########################################################################
# Indexing Cluster
#
-->
<indexing-cluster id="IndexingCluster" getDataInParallel="true">
```

```
  <dgidx ref="Dgidx1" />
  <dgidx ref="Dgidx2" />
</indexing-cluster>
```

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

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

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

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

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>
```

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>
```

IFCR

The IFCR is a custom component that specifies user information for an Endeca Configuration Repository that is running inside Oracle Endeca Workbench. The deployment template scripts use the information to connect to an Endeca Configuration Repository and move configuration used by Authoring and Live Dgraphs, the media MDEX reference application, and the IFCR Backup Utility.

You define an IFCR component in the `WorkbenchConfig.xml` file which is then referenced by `AppConfig.xml`.

The `custom-component` configuration properties

The `custom-component` configuration properties identify the IFCR in the Data Ingest Hosts section of `DataIngest.xml`.

The properties are defined as follows:

- `id` - Assigns a unique ID to a specific IFCR instance.
- `host-id` - Points back to the `id` attribute of the `host` global configuration element.
- `class` - Specifies the class that implements the `IFCRComponent`. Specify class="`com.ende¬ca.soleng.eac.toolkit.component.IFCRComponent`".

IFCR configuration properties

The configuration properties are defined as follows:

- `repositoryUrl` - Specifies host, port, and ifcr directory as `http://<workbench host>:<port>/ifcr`.
- `username` - Name of the user logging in to Oracle Endeca Workbench where the Endeca Configuration Repository is hosted.
- `password` - Corresponding password for the user name.
- `numExportBackups` - Indicates the number of backups to keep for exported configuration of the Endeca Configuration Repository. If this property is not configured, then no backups are retained. The default value is 5.

Example

This example shows a typical configuration:

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

    # IFCR - A component that interfaces with the Workbench repository.
  -->
  <custom-component id="IFCR" host-id="ITLHost" class="com.ende¬
ca.soleng.eac.toolkit.component.IFCRComponent">
    <properties>
      <property name="repositoryUrl" value="http://localhost:8006/ifcr" />
```

```
      <property name="username" value="admin" />
      <property name="password" value="admin" />
      <property name="numExportBackups" value="3" />
    </properties>
  </custom-component>
```

Workbench Manager

The Workbench Manager is a custom component that specifies connection information for Oracle Endeca Workbench and also a configuration directory for Oracle Endeca Workbench. The deployment template scripts use the information to connect to Workbench and update shared configuration contained in the configuration directory.

You define a Workbench Manager component in the `WorkbenchConfig.xml` file which is then referenced by `AppConfig.xml`.

The `custom-component` configuration properties

The `custom-component` configuration properties identify the Workbench Manager in the Data Ingest Hosts section of `DataIngest.xml`.

The properties are defined as follows:

- `id` - Assigns a unique ID to a specific Workbench instance.
- `host-id` - Points back to the `id` attribute of the `host` global configuration element.
- `class` - Specifies the class that implements the `WorkbenchManagerComponent`. Specify class="`com.endeca.soleng.eac.toolkit.component.WorkbenchManagerComponent`".

Workbench configuration properties

The configuration properties are defined as follows:

- `workbenchHost` - Host name of the server on which Oracle Endeca Workbench is running.
- `workbenchPort` - Port on which Workbench listens. This is the port of the Endeca Tools Service on the Oracle Endeca Workbench host. If the application is running in SSL mode, the `workbench¬Port` is the SSL port of Workbench.
- `configDir` - Directory to which Workbench configuration files are uploaded or downloaded by other components in the implementation.
- `workbenchTempDir` - Temporary directory used for Workbench interaction. Post-Forge dimensions are uploaded or downloaded from this directory by other components in the implementation.

Example

This example shows a typical configuration:

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

    # WorkbenchManager - A component that interfaces with the legacy
    # 'web studio' configuration repository. It is used primarily during
    # data ingest to load post-forge dimensions into Workbench.
  -->
  <custom-component id="WorkbenchManager" host-id="ITLHost" class="com.en¬
deca.soleng.eac.toolkit.component.WorkbenchManagerComponent">
    <properties>
      <property name="workbenchHost" value="localhost" />
```

```
      <property name="workbenchPort" value="8006" />
    </properties>
    <directories>
      <directory name="configDir">./config/pipeline</directory>
      <directory name="workbenchTempDir">./data/workbench/temp</directory>

    </directories>
  </custom-component>
```

Reporting

Oracle 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.

Configuration Manager

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

Note: In Deployment Template 3.0, the Configuration Manager was deprecated and replaced by Workbench Manager.

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 Workbench 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 Workbench 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="localhost" />
    <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>
```

Configuring the BeanShell scripts

The following list describes a number of customization approaches that you can implement to extend the existing functionality or add new functionality to the template.

- For example, if a deployment uses JDBC to read data into the Forge pipeline instead of using extracted data files, the following changes would be implemented in the BaselineUpdate script:

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

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

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

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

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

Note that this amended BeanShell script imports two classes from the classpath, references variables that point to elements in the `AppConfig.xml` document (e.g. Forge, Dgidx) and defines new variables without specifying their type (e.g. srcDir, destDir). Details about BeanShell scripting can be found in Appendix A of this guide.

- Write new BeanShell scripts - Some use cases may call for greater flexibility than can easily be achieved by modifying existing BeanShell scripts. In these cases, writing new BeanShell scripts may accomplish the desired goal. For example, the following BeanShell script extends the previous example by pulling the new functionality into a separate script:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

      // get dimensions for processing
      getConfig();

    }
  }
  ```

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

  In order to use the new functionality, the developer will compile the new class and ensure that it is included on the classpath when invoking scripts. The simplest way to do this is to deploy the compiled `.class` file to the `[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.

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 3

Replacing the Default Forge Pipeline

This chapter describes how to modify or create a Forge pipeline that is designed for use within the deployment template operational structure. This includes pipeline naming requirements, common errors encountered, etc.

About the sample pipelines

For testing purposes, the Deployment Template includes a Developer Studio project with two pipelines (a baseline and a partial). The sample pipelines facilitate testing the deployment template; however, the files should be replaced with project-specific files immediately after a deployed application has been properly configured.

The pipelines are located in `[appdir]/config/pipeline`. The pipeline for a baseline update processes 10 records, and the pipeline for a partial update that adds 2 more records.

Sample pipeline overview

This section describes the high-level steps that are necessary to integrate a new/existing pipeline with a deployment template.

Additional detail on each of these steps is provided in later sections.

1. Ensure that the application name and pipeline configuration prefix match the data prefix configured in the deployment template.
2. Place pipeline configuration files in the `[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.

Specifying a pipeline

By default, the Deployment Template checks the `[appdir]/config/pipeline` for the pipeline to run. This includes baseline updates and partial updates. It is simplest to put your pipeline files in this

directory. Alternatively, the `devStudioConfigDir` attribute in the `ConfigManager` custom component specifies the pipeline to run.

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

1. Ensure that your pipeline files are located in `[appdir]/config/pipeline`.
2. Alternatively, modify the `devStudioConfigDir` property in the `ConfigManager` custom component to reference the pipeline directory.
   In this example, the pipeline is stored in the `pipeline` directory:

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

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

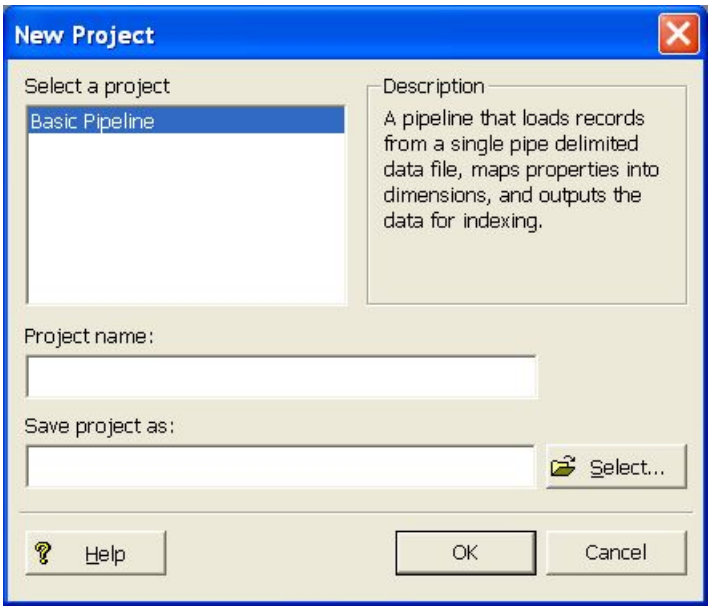
3. If you modified the value in step 2, also modify the value of the `configDir` attribute in the Partial update Forge section to reference the `config/pipeline` directory.
   For example:

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

Creating a new project

Once the reference configuration files have been deleted, a new pipeline configuration project can be created.

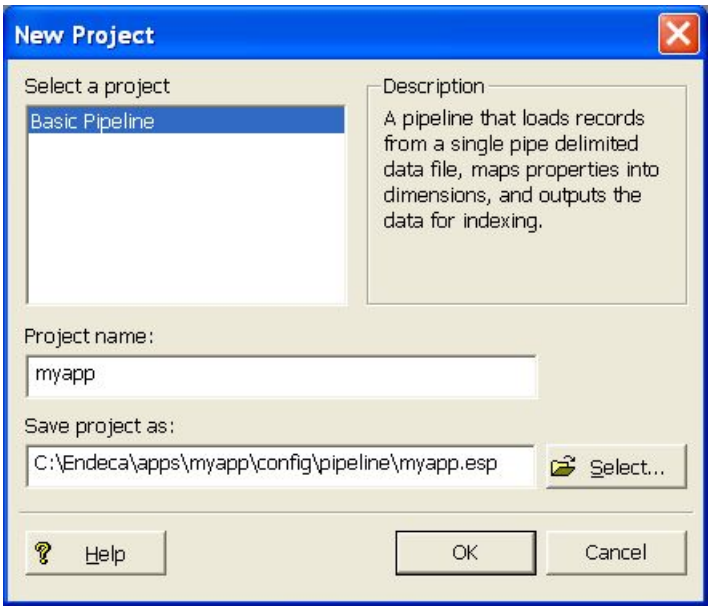
When creating a new project using the Oracle Endeca Developer Studio, you are prompted with the following dialog box:



To create a new project:

1. In order for a new pipeline to be run properly within the deployment template, the following must be properly specified:
   a) The **Project Name** field must be the same as the data prefix specified for the "`app`" element in `[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 are 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 51
This section provides troubleshooting information for commonly received errors.

Configuring a record specifier

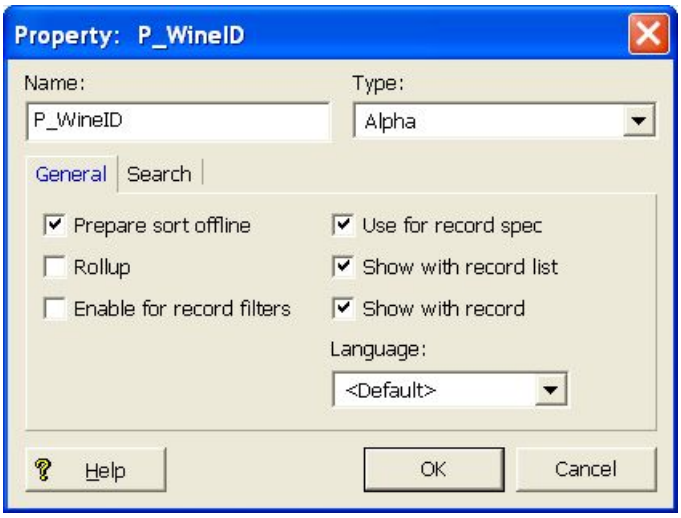
The deployment includes support for both baseline and partial index updates. In order to support partial updates, an application must include a record specifier, which is a property marked as the unique identifier of records in the index.

For details about the record specifier property, refer to the *Platform Services Forge Guide*.

When configuring your application, identify a property for which each record will have a unique assigned value.

To enable the use of that property as a record spec:

1. Open the **Property** dialog box in Developer Studio.
2. Check the box labeled "**Use for record spec**."



Forge flags

In order to reduce the amount of configuration required to integrate a pipeline into a deployment template, a standard deployment template application runs the primary and partial update Forge processes with an abbreviated set of flags.

Since the deployment template already specifies directory structures and file prefixes, the following flags are used to override a pipeline's input and output components, specifying the appropriate directories and prefixes for either reading or writing data.

Primary Forge flags

Flag	Description
`--inputDir`	`[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 application includes a sample dataset in `[appdir]/test_data/baseline` directory. When processing the sample data, the `load_baseline_test_data` script copies the contents of this directory into the `[appdir]/data/incoming/` directory and sets a flag in the EAC.

This flag, named `baseline_data_ready`, indicates to the deployment template scripts that the data extraction process is complete and data is ready for processing. Once that has occurred, the baseline update process copies these files into the `[appdir]/data/processing` directory before running the Forge process.

When using a default deployment template application, it is therefore necessary for all input record adapters to look in the `[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 50
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

Because the `--outputPrefix` and `--outputDir` flags are both included, the deployment template will override any values specified for the Indexer Adapter "URL" and "Output prefix" properties.

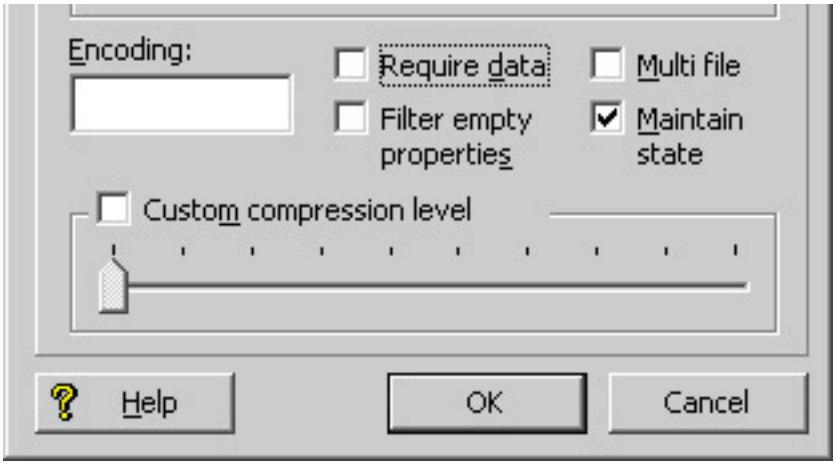
Therefore, it is unnecessary to modify these properties in most cases.

Output record adapters

Output record adapters are often used to generate debug or state information. By default, the location to which this data is written will be overridden by the `--outputDir` flag.

In most cases, however, it is undesirable for these files to be written to the same location as the Forge output files.

In these cases, an output record adapter can be configured to instead respect the `--stateDir` flag by selecting the "**Maintain State**" checkbox.



Now any files generated by this output record adapter will be written to the `[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 49
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 49
The record adapters load the source data.

Output record adapters on page 50
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.

Chapter 4

Managing Data Operations

This section describes how to incorporate test data and production data into an application.

Running a baseline update with test data

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

The `load_baseline_test_data` script loads the test data stored in `[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.

When you are done familiarizing yourself with the data processing steps and the test data, see *Running a baseline update with production data* on page 54. Processing production data requires the following changes to an application' s configuration:

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

To run a baseline update with test data:

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

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

4. 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
  ```

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

Running a baseline update with production data

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

To run a baseline update with production data:

1. Ensure that the Endeca HTTP Service is running on each server in the deployment environment and that you have already deployed and initialized an application.
2. Replace the default Forge pipeline (Developer Studio configuration files) in `[appdir]/config/pipeline` with the Developer Studio configuration files for your application. For details, see *Replacing the Default Forge Pipeline* on page 43.
3. Replace the baseline test data stored in `[appdir]/test_data/baseline` with production data for the application. This step varies depending on your application requirements. It can include any of the following approaches:
   - Add a data extract file to the `[appdir]/test_data/baseline` and delete the test data extract.
   - Set up a CAS crawl to run as part of the `baseline_update` script.
   - Make a direct call to a database via ODBC or JDBC.
4. Start a command prompt (on Windows) or a shell (on UNIX).
5. Navigate to the `control` directory of deployed application.

   This is located under your application directory. For example: `C:\Endeca\apps\<app name>\control`.
6. Set the `baseline_data_ready` flag in the EAC by running the `set_baseline_data_ready_flag` script.

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

7. Run the `baseline_update` script.
   - On Windows:
     ```
     [appdir]\control\baseline_update.bat
     ```
   - On UNIX:
     ```
     [appdir]/control/baseline_update.sh
     ```
8. Examine the indexed data in an Endeca front-end application.
   For example, start a Web browser and open the JSP reference application at `http://localhost:8006/endeca_jspref`.

Running a partial update with production data

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

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

To run a partial update with production data:

1. Ensure that the Endeca HTTP Service is running on each server in the deployment environment and that you have already deployed and initialized an application.
2. Replace the default Forge pipeline (Developer Studio configuration files) in `[appdir]/config/pipeline` with the Developer Studio configuration files for your application. For details, see *Replacing the Default Forge Pipeline* on page 43.
3. Provide the partial data (incremental data changes since the last baseline update). This step varies depending on the application requirements. It can include any of the following approaches:
   - Add a data extract file to the `[appdir]/test_data/partial`.
   - Set up a CAS crawl to run as part of the `baseline_update` script.
   - Make a direct call to a database via ODBC or JDBC.
4. Start a command prompt (on Windows) or a shell (on UNIX).
5. Navigate to the `control` directory of deployed application.

   This is located under your application directory. For example: `C:\Endeca\apps\<app name>\control`.
6. Set the `partial_data_ready` flag in the EAC by running the `set_partial_data_ready_flag` script.
   - On Windows:
     ```
     [appdir]\control\set_partial_data_ready_flag.bat
     ```

- On UNIX:
  ```
  [appdir]/control/set_partial_data_ready_flag.sh
  ```

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

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

Running CAS crawls

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

For details about `ContentAcquisitionServerComponent`, see the *EAC Component API Reference for CAS Server (Javadoc)* installed in `CAS\<version>\doc\cas-dt-javadoc` and see the CAS examples in *Script Reference* on page 57.

Chapter 5

Script Reference

This section describes scripts that are included with the Deployment Template, provides additional sample scripts, and provides information about running and configuring them.

Deployment Template script reference

The Deployment Template includes a set of utility scripts with deployed applications.

The following scripts are available in the `control` directory of a deployed application:

Script	Purpose
`baseline_update`	Runs a baseline update.
`export_site`	Takes a path to an XML file as an argument and exports the content in the Endeca Configuration Repository to the specified XML file. If no file is specified, site data is exported to `<App_Name>-<timestamp>.xml`, where the timestamp format is `YYYY-MM-DD_HH-MM-SS`.
`get_editors_config`	Exports editor configuration to the `<App_Dir>\config\editors\config` directory.
`get_media`	Exports media configuration to the `<App_Dir>\config\media` directory.
`get_templates`	Exports template configuration to the `<App_Dir>\config\cartridge_templates` directory.
`import_site`	Takes a path to an XML file and imports the content to the Endeca Configuration Repository. Optionally, you can use the `--force` flag to override the confirmation prompt for overwriting site content that already exists.

Script	Purpose
load_baseline_test_data	Copies data from the <App_Dir>\test_data\baseline\ directory to <App_Dir>\data\incoming for a baseline update and calls the set_baseline_data_ready_flag script.
load_partial_test_data	Copies data from the <App_Dir>\test_data\partial\ directory to <App_Dir>\data\partials\incoming for a partial update and calls the set_partial_data_ready_flag script.
partial_update	Runs a partial update.
promote_content	Promotes content and configuration in the authoring environment to the live environment.
runcommand	Provides a means of invoking methods in AppConfig.xml against specified instances of objects. You can run runcommand with the --help flag for a list of command line arguments and flags.
set_baseline_data_ready_flag	Sets the baseline_data_ready flag in the EAC.
set_editors_config	Imports editor configuration from <App_Dir>\config\editors\config to the Endeca Configuration Repository.
set_media	Imports media from <App_Dir>\config\media to the Endeca Configuration Repository.
set_partial_data_ready_flag	Sets the partial_extract flag in the EAC.
set_templates	Imports templates from <App_Dir>\config\cartridge_templates to the Endeca Configuration Repository.
initialize_services	This script should be run once after deploying an application. It does the following: • Removes existing application provisioning • Sets new EAC provisioning and performs initial setup • Calls set_editors_config • Calls set_media • Calls set_templates

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]
```

Forge-based data processing

The Deployment Template supports running baseline and partial updates using Forge. In this processing model, an update essentially runs a CAS crawl (if applicable), Forge, Dgidx, and then updates the Dgraphs in an application.

Dgraph baseline update script using Forge

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

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

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

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

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

as soon as the script copies the data out of the `data/incoming` directory, indicating that new data may be extracted.

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

3. If Workbench integration is enabled, download and merge 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 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 Workbench 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 using Forge on page 62
The partial update script defined in the `DataIngest.xml` document for a Dgraph deployment is included in this section, with numbered steps indicating the actions performed at each point in the script.

Dgraph partial update script using Forge

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

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

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

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

2. Validate data readiness. Test that the EAC contains at least one flag with the prefix "`partial_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.

Related Links

Dgraph baseline update script using Forge on page 59

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

Dgraph baseline update script using Forge and a CAS full crawl script

After running a full CAS crawl, you can run a baseline update using Forge to incorporate the records from a Record Store instance.

This example runs a *baseline* update that includes a *full* CAS crawl. The crawl writes output to a Record Store instance and then Forge incorporates the records from the crawl. To create this sequential workflow of CAS crawl and then baseline update, you can do the following:

- Remove the default `Forge.isDataReady` check from the baseline update script. This call handles concurrency control around Forge input files. The Record Store has built-in logic to handle concurrency between read and write operations, so no external concurrency control is required. Removing this call means that the lock manager does not check the flag or wait on the flag to be cleared before running a CAS crawl.
- Add a call to `runBaselineCasCrawl()` to run the full CAS crawl.
- Remove the call to `Forge.getIncomingData()` that fetches extracted data files.

For example, this baseline update script calls `CAS.runBaselineCasCrawl("MyCrawl")` which runs a full CAS crawl that writes output to a Record Store instance. Then the script continues with baseline update processing.

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

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

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

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

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

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

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

        // distributed index, update Dgraphs
```

```
        DistributeIndexAndApply.run();

        WorkbenchManager.cleanDirs();
        Forge.getPostForgeDimensions();
        WorkbenchManager.updateWsDimensions();

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

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

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

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

For example:

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

Dgraph partial update script using Forge and a CAS incremental crawl script

After running an *incremental* CAS crawl, you can run a *partial* update that incorporates the records from a Record Store instance.

To create this sequential workflow of incremental CAS crawl and then partial update, you can do the following:

- Remove the default `PartialForge.isPartialDataReady` check from the partial update script. This call handles concurrency control around Forge input files. The Record Store has built-in logic to handle concurrency between read and write operations, so no external concurrency control is required. Removing this call means that the lock manager does not check the flag or wait on the flag to be cleared before running a CAS crawl.
- Add a call `runIncrementalCasCrawl()` to run the incremental CAS crawl.
- If the pipeline does not read from sources in the Forge incoming directory, remove the call to `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 Forge updates

There are more complicated cases where multiple CAS crawls are running on their own schedules, and Forge updates are running on their own schedules. To coordinate this asynchronous workflow of CAS crawls and baseline or partial updates, you add code that calls methods in `ContentAcquisi¬tionServerComponent`.

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

CAS-based data processing

The Deployment Template supports running baseline and partial updates using CAS as a replacement for Forge. In this processing model, the update runs a CAS crawl to produce MDEX-compatible output. This is the step that removes the need for Forge. Then the update runs Dgidx and updates the Dgraphs in an application.

Dgraph baseline update script using CAS

You do not need to run Forge if you run a CAS crawl that is configured to produce MDEX-compatible output as part of your update process.

This example runs a *baseline* update that includes a*full* CAS crawl. The crawl writes MDEX compatible output and then the update invokes Dgidx to process the records, dimensions, and index configuration produced by the crawl. To create this sequential workflow of CAS crawl and then baseline update, you add a call to `runBaselineCasCrawl()` to run the CAS crawl.

For example, this baseline update script calls `CAS.runBaselineCasCrawl("${lastMileCrawl¬Name}")` which runs a CAS crawl that writes MDEX-compatible output to instance. Then the script continues with baseline update processing.

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

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

      // archive logs and run the crawl.
      CAS.runBaselineCasCrawl("${lastMileCrawlName}");
      Dgidx.archiveLogDir();
      Dgidx.run();

      // distributed index, update Dgraphs
```

```
      DistributeIndexAndApply.run();

      WorkbenchManager.cleanDirs();
      CAS.copyOutputDimensionsFile("${lastMileCrawlName}", WorkbenchManag¬
er.getWorkbenchTempDir());
      WorkbenchManager.updateWsDimensions();

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

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

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

For example:

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

Dgraph partial update script using CAS

You do not need to run Forge if you run a CAS crawl that is configured to produce MDEX-compatible output as part of your update process.

This example runs an *incremental* CAS crawl that writes MDEX compatible output and then runs a *partial* update that to process data records (not index configuration or dimension value records) and then distributes the data records to the Dgraph. To create this sequential workflow of CAS crawl and then partial update, you add a call to `runIncrementalCasCrawl()` to run the CAS crawl.

For example, this partial update script calls `CAS.runIncrementalCasCrawl("${lastMileCrawl¬Name}")` which runs a CAS crawl that writes MDEX-compatible output to instance. Then the script continues with 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")) {
```

```
      // run crawl and rename in data/cas_output w/timestamp.
      CAS.runIncrementalCasCrawl("${lastMileCrawlName}");

      // Copy the partial to the master cumulative directory
      CAS.fetchPartialsToCumulativeDir("${lastMileCrawlName}");

      // copy from srcPartials to localCumulative for authoring
      AuthoringDgraphCluster.copyPartialUpdateToDgraphServers();

      // copy from local to mdex's update-dir and trigger the update for
authoring
      AuthoringDgraphCluster.applyPartialUpdates();

      // copy from srcPartials to localCumulative for live
      LiveDgraphCluster.copyPartialUpdateToDgraphServers();

      // copy from localCumulative to mdex's update-dir and trigger the
update
      LiveDgraphCluster.applyPartialUpdates();

      // Archive accumulated partials
      CAS.archiveCumulativePartials();

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

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

For example:

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

CAS crawl scripts for Record Store output

This topic provides an example CAS crawl script with a crawl that is configured to write to Record Store output. To create a similar CAS crawl script in your application, add code to `AppConfig.xml` that specifies the CAS crawl to run locks the crawl (to wait for any running crawls to complete), runs the crawl, and releases the lock. Depending on your environment, you may need a script that runs a full CAS crawl and a script that runs an incremental CAS crawl.

This example `AppConfig.xml` code runs a *full* crawl that writes to a Record Store instance:

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

    # full crawl script
    #
 -->
```

```
  <script id="MyCrawl_fullCrawl">
    <log-dir>./logs/provisioned_scripts</log-dir>
    <provisioned-script-command>./control/runcommand.bat MyCrawl_fullCrawl
 run</provisioned-script-command>
    <bean-shell-script>
      <![CDATA[
    crawlName = "MyCrawl";

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

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

      CAS.runBaselineCasCrawl(crawlName);

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

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

This example runs an *incremental* crawl that writes to a Record Store instance:

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

    # incremental crawl script
    #
  -->
  <script id="MyCrawl_IncrementalCrawl">
    <log-dir>./logs/provisioned_scripts</log-dir>
    <provisioned-script-command>./control/runcommand.bat MyCrawl_Incremen¬
talCrawl run</provisioned-script-command>
    <bean-shell-script>
      <![CDATA[
    crawlName = "MyCrawl";

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

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

      CAS.runIncrementalCasCrawl(crawlName);

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

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

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

CAS crawl scripts for record file output

This topic provides an example CAS crawl script with a crawl that is configured to write to record file output. To create a similar CAS crawl script in your application, add code to `DataIngest.xml` that specifies the CAS crawl to run locks the crawl (to wait for any running crawls to complete), runs the crawl, and releases the lock. Depending on your environment, you may need a script that runs a full CAS crawl and a script that runs an incremental CAS crawl.

This example `DataIngest.xml` code runs a *full* crawl that writes to record file output:

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

    # full crawl script
    #
 -->

  <script id="MyCrawl_fullCrawl">
    <log-dir>./logs/provisioned_scripts</log-dir>
    <provisioned-script-command>./control/runcommand.bat MyCrawl_fullCrawl
 run</provisioned-script-command>
    <bean-shell-script>
      <![CDATA[
    crawlName = "MyCrawl";

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

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

        if (!CAS.isCrawlFileOutput(crawlName)) {
          throw new UnsupportedOperationException("The crawl " + crawlName
 +
              " does not have a File System output type. The only supported
 output type for this script is File System.");
        }

        log.info("Starting full CAS crawl '" + crawlName + "'.");
        // Remove all files from the crawl's output directory
        CAS.cleanOutputDir(crawlName);
        CAS.runBaselineCasCrawl(crawlName);
        // Rename the output to files to include the crawl name
        // so they do not collide with the output from other crawls
        CAS.renameBaselineCrawlOutput(crawlName);

        destDir = PathUtils.getAbsolutePath(CAS.getWorkingDir(),
           CAS.getCasCrawlFullOutputDestDir());

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

```
        // clear the destination dir of full crawl from previous crawls
        CAS.clearFullCrawlOutputFromDestinationDir(crawlName);

        // remove previously collected incremental crawl files,
        // which are expected to be incorporated in this full crawl
        CAS.clearIncrementalCrawlOutputFromDestinationDir(crawlName);

        // copy the full crawl output to destination directory
        CAS.copyBaselineCrawlOutputToDestinationDir(crawlName);
        LockManager.releaseLock("crawl_lock_" + crawlName);
     }

    else {
      log.warning("Failed to obtain lock.");
    }

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

This example `DataIngest.xml` code runs an *incremental* crawl that writes to record file output:

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

    # incremental crawl script
    #
  -->
  <script id="MyCrawl_IncrementalCrawl">
    <log-dir>./logs/provisioned_scripts</log-dir>
    <provisioned-script-command>./control/runcommand.bat MyCrawl_Incremen¬
talCrawl run</provisioned-script-command>
    <bean-shell-script>
      <![CDATA[
    crawlName = "MyCrawl";

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

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

       if (!CAS.isCrawlFileOutput(crawlName)) {
          throw new UnsupportedOperationException("The crawl " + crawlName
 +
            " does not have a File System output type. The only supported
output type for this script is File System.");
       }

       log.info("Starting incremental CAS crawl '" + crawlName + "'.");
       // Remove all files from the crawl's output directory
       CAS.cleanOutputDir(crawlName);
       CAS.runIncrementalCasCrawl(crawlName);
       // Timestamp and rename the output to files to include the
       // crawl name so they do not collide with the output from
       // previous incremental output from this crawl or incremental
       // output from other crawls
       CAS.renameIncrementalCrawlOutput(crawlName);

       destDir = PathUtils.getAbsolutePath(CAS.getWorkingDir(),
          CAS.getCasCrawlIncrementalOutputDestDir());
```

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

       // copy crawl output to destination directory
       // Note: We assume a downstream process removes incremental crawl
output
       // from this directory that has already been processed.
       CAS.copyIncrementalCrawlOutputToDestinationDir(crawlName);

       LockManager.releaseLock("crawl_lock_" + crawlName);
     }

     else {
        log.warning("Failed to obtain lock.");
     }

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

Configuration update script

The configuration update script defined in the `DataIngest.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 `DataIngest.xml` document.

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

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

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

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

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

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

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

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

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

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

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

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

6. Update Report Generator to the appropriate time range and output file name. Oracle 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
        // defined in "webstudio" host and its "webstudio-report-dir"
        // directory
        reportHost = "webstudio";
       absDestDir = PathUtils.getAbsolutePath(webstudio.getWorkingDir(),

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

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

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

Appendix 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 82
This section describes the application elements available in the custom namespace defined by the `eacToolkit.xsd` XML schema.

Hosts on page 82
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 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">`
	`  <dgidx ref="Dgidx1" />`
	`  <dgidx ref="Dgidx2" />`
	`</indexing-cluster>`

dgraph	This element defines a Dgraph component, including attributes that define the functionality of the Dgraph process as well as custom properties and directories used to configure the functionality of the Dgraph object's methods. Multiple dgraph elements can be defined. Each dgraph element inherits, and potentially overrides, configuration specified in the dgraph-defaults element (see below).
	`<dgraph id="Dgraph1" host-id="MDEXHost" port="15000">`
	`  <properties>`
	`  <property name="restartGroup" value="A" />`
	`  <property name="updateGroup" value="a" />`
	`  </properties>`
	`  <log-dir>./logs/dgraphs/Dgraph1</log-dir>`
	`  <input-dir>./data/dgraphs/Dgraph1/dgraph_input</input-dir>`
	`  <update-dir>./data/dgraphs/Dgraph1/dgraph_input/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" />`

Element	Description
	<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>
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">

Element	Description
	<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 must specify the name of the implemented class that extends com.Endeca.soleng.eac.toolkit.component.CustomComponent. The custom component is also used to implement the Configuration Manager, Workbench Manager, and IFCR components. <custom-component id="IFCR" host-id="ITLHost" class="com.endeca.soleng.eac.toolkit.component.IFCRCompo¬ nent"> <properties> <property name="repositoryUrl" value="http://local¬ host:8006/ifcr" /> <property name="username" value="admin" /> <property name="password" value="admin" /> <property name="numExportBackups" value="3" /> </properties> </custom-component>

Related Links

Display component status on page 97
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"` `  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 83

This section describes the component elements available in the custom namespace defined by the `eacToolkit.xsd` XML schema.

Index

O

P

R

S

U

V

W

X