Oracle Commerce Guided Search Platform Services

Application Controller Guide

Version 11.1 • July 2014

ORACLE

COMMERCE

Contents

Copyright and disclaimer

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

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

Preface

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

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

About this guide

This guide describes the tasks involved in managing implementations using the Oracle Commerce Guided Search Application Controller.

Who should use this guide

This guide is intended for developers responsible for provisioning and managing Oracle Commerce Guided Search implementations.

Conventions used in this guide

This guide uses the following typographical conventions:

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

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

Contacting Oracle Support

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

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

Chapter 1

Introduction

This section introduces the Oracle Endeca Application Controller and its architecture.

About the Oracle Endeca Application Controller

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

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

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

EAC architecture

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

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

- EAC Central Server
- EAC Agent

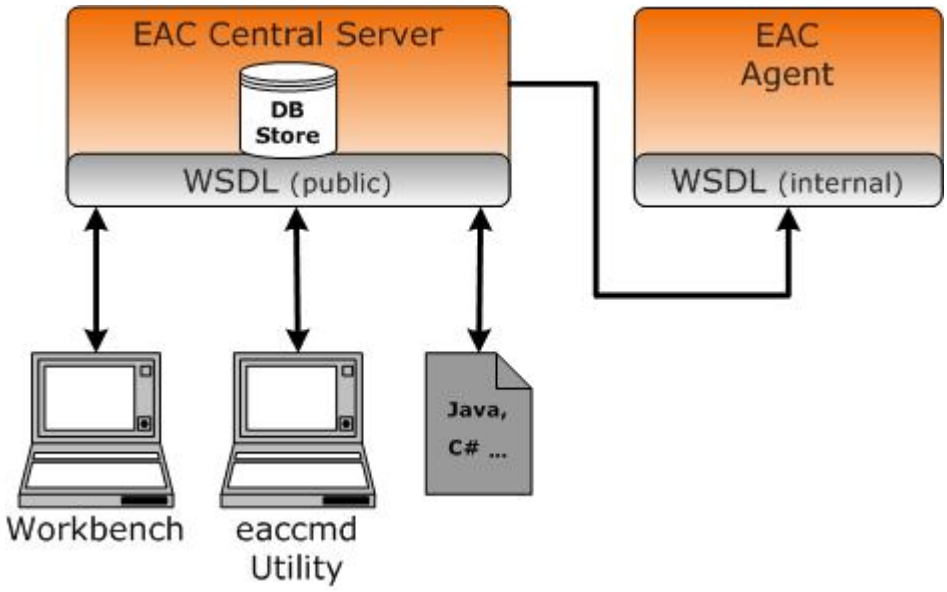
You can communicate with the EAC and provide instance configuration and resource configuration information to the EAC Central Server, using any of the three methods:

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

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

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

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



EAC Central Server

One instance of the EAC serves as the EAC Central Server for your implementation. This instance includes a WSDL-enabled interface, through which you communicate with the EAC. Communication is implemented with the standard Web services protocol, SOAP.

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

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

EAC Agents

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

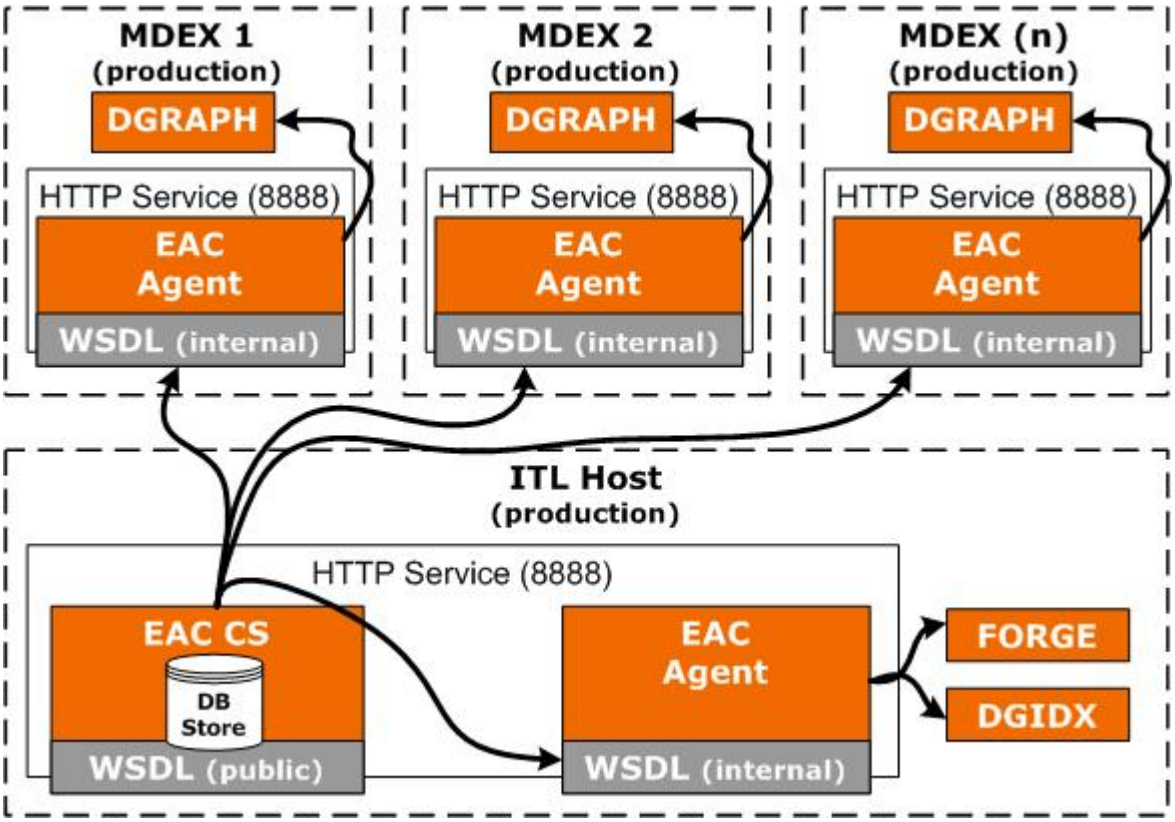
Each Agent also contains a small repository for its own use. The EAC Central Server communicates with its Agents through an internal Web service interface. You do not communicate directly with the Agents—all command, control, and monitoring functions are sent through the EAC Central Server.

EAC architecture example

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

The following diagram shows the architecture of the EAC.

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



Related Links

Using the eaccmd Tool on page 45

This section describes the eaccmd command-line tool, which can be used to provision and run the Endeca Application Controller.

Endeca Application Controller API Interface Reference on page 61

Application Controller interfaces are documented here. However, the exact syntax of a class member depends on the output of the WSDL tool that you are using. Be sure to check the client stub classes that are generated by your WSDL tool for the exact syntax of the Application Controller API class members.

Chapter 2

Using the Application Controller

This section describes how to use the Application Controller.

Installing the Application Controller

This topic describes the ways you can install the Application Controller.

You have the following choices:

- Install the Agent. The Agent controls the workings of a single machine in an Application Controller deployment. There are typically several Agents in a deployment.
- Install the EAC Central Server along with one or more EAC Agents. The Central Server acts as a hub in an Application Controller deployment, relaying commands to each of the Agents in the deployment. As such, there is only a single Central Server per deployment. Alternatively, you can use an SSL-enabled Central Server. Upon configuration, this version encrypts the HTTP channel between the Central Server and the client Web services.

During installation, when you select whether you want to run the Agent and/or the Central Server on a machine, an XML pointer to the appropriate WAR file is copied to its workspace directory. The presence or absence of these files in the workspace directory determines what that machine is running. If you want to run the SSL-enabled version of the Central Server, you must copy the XML pointer to it to your workspace directory manually, as described in the following section.

Enabling SSL security in the Application Controller

SSL in the Application Controller is disabled by default.

To enable SSL security (between the client and the EAC Central Server, between the Central Server and an Agent, or between Agents), you need to do the following:

- Enable the SSL version of the appropriate Application Controller WAR file (`eac-ssl.war` replaces `eac.war` for the Central Server, and `eac-agent-ssl.war` replaces `eac-agent.war` for the Agent).
- Modify the `server.xml` file for the Tomcat that is hosting the Application Controller.

For details on enabling SSL security in the Application Controller, see the *Endeca Security Guide*.

Specifying the EAC Central Server in Oracle Endeca Workbench

You can specify the EAC Central Server from the Endeca Workbench EAC Settings page.

On the EAC Settings page of Endeca Workbench, you specify the host and port for the EAC Central Server. These settings control which machine Endeca Workbench communicates with when making requests to EAC. See the Endeca Workbench help for more information.

Starting and stopping the Application Controller directly on UNIX

Although you typically control the Application Controller through Endeca Workbench, you can also start and stop it independently.

In a UNIX shell, you start the Application Controller (along with any other components using the same port) with the following command:

```
$ENDECA_ROOT/tools/server/bin/startup.sh
```

You stop the Application Controller (along with any other components using the same port) with the following command:

```
$ENDECA_ROOT/tools/server/bin/shutdown.sh
```

Starting the Application Controller from inittab

In a UNIX production environment, the Endeca Application Controller can be started by init from inittab.

In a UNIX development environment, the Endeca HTTP Service can be started from the command line. In a UNIX production environment, however, Oracle recommends that it be started by **init** from **inittab**. If the service crashes or is terminated, **init** automatically restarts it.

The UNIX version of Platform Services contains a file named `endeca_run.sh` that is in the `$ENDECA_ROOT/tools/server/bin` directory. This is a version of `startup.sh` that calls run instead of start and redirects stdout and stderr to `$ENDECA_CONF/logs/catalina.out`.

You can write a script that is referenced in **inittab**. The script sets environment variables and then calls endeca_run.sh. When writing your script, it is recommended as a best practice that you run the Endeca HTTP Service as a user other than root. When running the service as a non-root user, you can set a USER environment variable that will be inherited by other scripts, such as EAC scripts.

This sample script (named `start_endeca_http_service.sh`) sets the ENDECA_USER variable to the “endeca” user, sets the INSTALLER_SH variable to the path of the environment variables script and sources it, and then does an su to change to the “endeca” user:

```
#!/bin/sh
ENDECA_USER=endeca
INSTALLER_SH=/usr/local/endeca/PlatformServices/workspace/setup/installer_sh.ini
# We want to use installer_sh.ini variables in this script,
# so we source it here.
source $INSTALLER_SH
# change to user endeca
su $ENDECA_USER -c "/bin/sh -c \"source $INSTALLER_SH; \
```

```
  cd $ENDECA_CONF/work; exec env USER=$ENDECA_USER \
  $ENDECA_ROOT/tools/server/bin/endeca_run.sh\""
```

On Solaris platforms, replace "source" with "." because source is not a command in the Bourne shell. The `start_endeca_http_service.sh` script is then referenced in inittab with an entry similar to this example.

```
ec:2345:respawn:/usr/local/endeca/PlatformServices/workspace/setup/start_ende¬
ca_http_service.sh
```

When writing your startup script, keep in mind that it is server-specific, and therefore its details (such as paths and user names) depend on the configuration of your server.

Starting and stopping the Application Controller on Windows

Although you typically control the Application Controller through Endeca Workbench, you can also start and stop it independently.

The Endeca HTTP service, which controls the Endeca Application Controller, is created, registered, and configured by the installation, and started when you reboot your computer after installation.

To stop and restart the Application Controller after installation, do the following:

1. Go to **Start** > **Control Panel** > **Administrative Tools** > **Services**.
2. In the Windows Services editor, select the **Endeca HTTP service**.
3. Click **Stop** or **Restart**.

Using the eac.properties file

The `eac.properties` file, which is located in the `$ENDECA_CONF/conf` directory on UNIX, or `%ENDECA_CONF%\conf` on Windows, is the general configuration file for the Endeca Application Controller.

The following section describes the process control-related settings you can specify in `eac.properties`.

Note: SSL-related properties in this file are discussed in the *Endeca Security Guide*.

Setting the MDEX Engine root directory

The attribute `com.endeca.mdexRoot` specifies the root directory of your MDEX Engine installation.

If you did not specify this directory upon installing Platform Services, the value for this setting will be blank. Note that although the EAC will start if this is left blank, If you install the MDEX Engine package later, you should specify the MDEX Engine root directory as an absolute path, including the MDEX Engine version number. For example:

```
com.endeca.mdexRoot=C:\\Endeca\\MDEX\\6.5.0
```

Setting the Copy utility's temporary directory

Directories are copied first to a specified temporary directory on the destination machine before being copied one file at a time to the target location.

You can configure the location of this temporary directory in the `eac.properties` file, using the optional setting `com.endeca.eac.filetransfer.fileTransferTempDir` as follows:

- If this setting is defined as an absolute path, the Copy utility uses it.
- If it is defined as a relative path, the Copy utility considers it to be relative to `%ENDECA_CONF%/state/`
- If it is not defined, the Copy utility uses the directory `%ENDECA_CONF%/state/file_transfer/`

Ensuring clean component shutdown

Server components such as the Dgraph can be cleanly shut down via their HTTP interface.

When stopping a server, the Application Controller first attempts to shut down the server through its HTTP interface. If this does not complete within 30 seconds, it kills the server process. You can modify this default with the `com.endeca.eac.process.shutdownTimeOutSecs` setting in `eac.properties`.

Managing server restarts

In an effort to make Endeca deployments more fault tolerant, the Application Controller automatically restarts servers that crash.

You can configure the number of times the Application Controller attempts to restart a server within a specified time window. If the server crashes more than the specified number of times in the specified time window, then it is marked as failed.

Both of these variables are set in eac.properties. The `com.endeca.eac.process.maxServerRestartsPerWindow` setting defaults to five, while `com.endeca.eac.process.serverRestartTimeWindowMins` defaults to one.

About the Application Controller log

The Endeca Application Controller log is located in `%ENDECA_CONF%\logs` (on Windows) or `$ENDECA_CONF/logs` (on UNIX).

The EAC log has a default size limit of 1G. The log is named `main.`*`rotation number`*`.log` and is part of a two-log rotation that rolls automatically when the maximum size is reached. When the second log file reaches the maximum size, the first is overwritten. That is, when `main.0.log` reaches the 1G size limit, the system starts to write to `main.1.log`. Once `main.1.log` reaches the 1G size limit, `main.0.log` is overwritten.

Modifying Application Controller logging levels

By default, Application Controller log files log WARNING and SEVERE messages.

If you want to capture INFO level messages as well, you need to modify the `logging.properties` file.

To modify logging levels in the `logging.properties` file:

1. Stop the Endeca HTTP service.
2. Navigate to `%ENDECA_CONF%\conf` (on Windows) or `$ENDECA_CONF/conf` (on UNIX).
3. Open `logging.properties`.
4. Locate the section EAC Log Level.
5. In the line `com.endeca.eac.level`, change WARNING to INFO.

6. Save and close the file.
7. Start the Endeca HTTP service.

For more information about logging options, see the comments in `logging.properties`.

Chapter 3

Provisioning Implementations with Application Controller

You specify Application Controller hosts, components, and scripts, and later reference them in Workbench, eaccmd, or your custom Web services interface. This process is known as provisioning.

Provisioning overview

Provisioning an Endeca implementation with the Application Controller consists of the following steps.

- Creating a provisioning file, in which you define the hosts and components that comprise your implementation, as well as the scripts that it uses.
- Referencing that file when creating an implementation with the eaccmd tool or your custom Web service interface.

Note: This chapter provides examples using the sample wine reference implementation and the eaccmd tool.

Related Links

About eaccmd on page 45
When you manage your Endeca implementation with the Endeca Application Controller, you control and monitor its working through the EAC Central Server.

Endeca Application Controller API Interface Reference on page 61
Application Controller interfaces are documented here. However, the exact syntax of a class member depends on the output of the WSDL tool that you are using. Be sure to check the client stub classes that are generated by your WSDL tool for the exact syntax of the Application Controller API class members.

About the provisioning file and schema

The provisioning file is a file in XML format in which you define the following aspects of your implementation.

- Application (the root element)
- Hosts (and, optionally, directories on hosts)
- Components

- Scripts

The provisioning schema (named `eaccmdProvisioning.xsd`) is located in the `$ENDECA_ROOT/conf/schema` directory on UNIX (`%ENDECA_ROOT%\conf\schema` on Windows).

Note: You can name the provisioning file anything you like. In the remainder of this chapter, we frequently refer to the provisioning file as `app.xml`.

Invalid characters in provisioning

The following characters cannot be used when provisioning applications, components, hosts, scripts, or utility tokens.

Invalid Windows file name characters, including:

- Forward slash (`/`)
- Backslash (`\`)
- Colon (`:`)
- Asterisk (`*`)
- Question mark (`?`)
- Right and left angle brackets (`< >`)
- Double quotation mark (`"`)
- Vertical pipe (`|`)

These additional characters:

- Single quotation mark (`'`)
- Space

Defining the root Application element

The root element in a provisioning file is the application element.

You can also specify an applicationID in the eaccmd tool. If eaccmd specifies a different applicationID for the same application, it overrides the one provided in the provisioning file.

Defining hosts

In the hosts element you list each host by a host ID, a host name, a port number, and (optionally) properties and directories.

The host syntax is as follows:

```
<host host-id="host1" host-name="localhost" port="8888">
 <properties>
   <property name="department" value="engineering" />
   <property name="department" value="prof services" />
   <property name="enforceDiskQuota" />
 </properties>
</host>
```

In this example the port is the HTTP port through which the EAC Central Server communicates with its Agents. The optional use of host-id to alias host definitions is explained in the following section. The optional addition of properties and directories is described later in this document.

Related Links

Aliasing hosts with host-id on page 23
In each host definition, you can create a unique alias called host-id that may be used to refer to the specified host and port. (The host-name and port do not need to be unique.)

Provisioning directories on hosts on page 23
As part of host provisioning, you can also provision directories using a full path and a name.

Adding properties to hosts and components on page 24
You can add properties, consisting of a required name and an optional value, to any host or component element.

Aliasing hosts with host-id

In each host definition, you can create a unique alias called host-id that may be used to refer to the specified host and port. (The host-name and port do not need to be unique.)

For example, say you defined host1 as follows:

```
<host host-id="host1" host-name="localhost" port="8888" />
```

Later, when defining components, you could simply refer to that host-id when specifying the host for a given component.

```
<dgidx name="dgidx-0" host-id="host1">
```

Aliasing hosts in this way has two benefits:

- It allows you to switch staging and production machines easily, by changing the name and port associated with a host-id alias.
- It makes it possible to reference a single physical host through different host-id aliases.

Provisioning directories on hosts

As part of host provisioning, you can also provision directories using a full path and a name.

For example, assuming a host has already been provisioned as defined above, you could add the following element:

```
<host >
...
<directories>
<directory dir-id="input">
   <path>C:\staging_app\working\input</path>
  </directory>
</directories>
</host>
```

Defining components in your provisioning file

The components element contains all of the components in your implementation.

Depending on the component type, the settings vary. The following section provides details about all supported component types.

Note the following:

- The order of elements in a component does not matter.
- Unless otherwise noted, relative paths are supported.

- Required elements are labelled as such. If you attempt to provision a component without a required element, you will receive an error.

Using XML entities in your provisioning file

The Application Controller supports the use of XML entities in provisioning files.

For example, assume you established the following entities in your XML provisioning file:

```
<!DOCTYPE application [
  <!ENTITY W_base "C:\Endeca\PlatformServices\reference\sample_wine_data\data">
      ...
  <!ENTITY H1 "host1">
       ...
]>
```

Subsequently, when defining a Forge component, rather than having to enter the host machine and working directory like this:

```
<forge component-id="forge1" host-id="host1">
 <working-dir>
   C:\Endeca\PlatformServices\reference\sample_wine_data\data\
 </working-dir>
  ...
</forge>
```

you can instead refer to them by their entities, like this:

```
<forge component-id="forge1" host-id="&H1;">
   <working-dir>
     &W_base;\
   </working-dir>
```

Adding properties to hosts and components

You can add properties, consisting of a required name and an optional value, to any host or component element.

Such properties can be used for value mapping as well as for flagging the element in question.

You add properties as part of provisioning your application. After your application is provisioned, any properties that you defined are included in the application definition, which you can retrieve using eaccmd's describe-app command. This feature is only useful in user-provided scripts; it is not an additional place to pass arguments or options to Endeca components.

Related Links

Forge on page 27
A Forge element launches the Forge (Data Foundry) software, which transforms source data into tagged Endeca records.

Dgidx on page 28
A Dgidx component sends the finished data prepared by Forge to the Dgidx program, which generates the proprietary indices for each Dgraph.

Dgraph on page 30
A Dgraph element launches the Dgraph (MDEX Engine) software, which processes queries against the indexed Endeca records.

LogServer on page 32
The LogServer component controls the use of the Endeca Log Server.

ReportGenerator on page 33

The ReportGenerator component runs the Report Generator, which processes Log Server files into HTML-based reports that you can view in your Web browser and XML reports that you can view in Endeca Workbench.

Defining scripts in your provisioning file

A script is a named command that you provision and run within the Application Controller.

In most cases, a script invokes a batch file that runs a process, such as a baseline update or report generation, or otherwise exercises component control. Scripts provide the automation that makes it possible for you to wrap and reuse a sequence of commands, without removing your ability to configure your application.

Although only one instance of each script can run at a time, most scripts are designed to be run repeatedly. For example, rather than start each component separately using Endeca Workbench or eaccmd, you can launch a baseline update script that will execute the start component commands in the proper sequence. You can reuse this script as often as you like.

Scripts live on the EAC Central Server; the EAC runs them from there. You can use scripts with the eaccmd tool, when accessing the Endeca WSDL programmatically, or within Endeca Workbench. Details on starting, stopping, and obtaining status for scripts for each of these environments can be found in the following places:

- Component and script control commands.
- The ScriptControl interface.
- In the *Oracle Endeca Workbench Help*.

Note: EAC scripts are not the same as Control Interpreter control scripts, which are deprecated. EAC scripts are not supported on clusters that are not uniformly one platform.

Related Links

ScriptControl interface on page 72
The ScriptControl interface provides programmatic script management capabilities.

Component and script control commands on page 51
The component and script control commands are used to start and stop components or scripts and retrieve their status.

Developing and maintaining scripts

You can write your own script in Java or .NET to contact the Central Server directly.

Because the EAC does not offer any mechanism for passing arguments to scripts at runtime, you need to provision a separate EAC script for every combination of arguments you plan to use. For example, if you want the Report Generator to generate daily and weekly reports, you must provision the associated script twice, once for each time period argument.

Script environment variables

You can write your own script in Java or .NET to contact the EAC Central Server directly. Script environment variables allow you to look up the host, port, and application name if you want to use them in your script.

These environment variables are set in the script's runtime environment. The EAC Central Server provides values for the following three variables:

- EAC_HOST is the hostname for the EAC Central Server host.
- EAC_PORT is the port number for the EAC Central Server host.
- EAC_APP is the application in which this script is provisioned.

Provisioning scripts

Scripts, like hosts and components, need to be provisioned before they can be used in the Application Controller.

Scripts can be provisioned with the following elements:

Sub-element	Description
script-id	Required. The name of this script.
cmd	Required. The command to launch the script.
log-file	Name of the script log file. If log-file is not specified, the default value is used.
working-dir	Working directory for the process that is launched. If it is specified, it must be an absolute path. If working-dir is not specified, the default value of `$ENDECA_CONF/work/(app_id)/` is used.

Example

This example provisions two scripts:

```
<scripts>
   <script script-id="script1">
      <cmd>runthis.sh</cmd>
   </script>
   <script script-id="script2">
      <cmd>run.sh --this</cmd>
   </script>
</scripts>
```

Using canonical paths in an application

The Application Controller provides a great deal of flexibility in computing directories.

However, if you want to write a generic script that can work with any kind of provisioning, the `getApplication()` method can make it difficult to predict unspecified directory destinations.

In such cases, the `getCanonicalApplication()` method returns the provisioning just as `getApplication()` does, but with all paths canonicalized. This process ensures that all paths are absolute, and that the working directory and log path settings are provided. It also prevents .. from being used in a path name. In eaccmd, you use the optional `--canonical` flag to the describe-app command to enable canonicalization.

Because it has to resolve paths on each Agent, `getCanonicalApplication()` can be slightly slower than `getApplication()`. Therefore, if you know that your script uses full paths, you may prefer to use `getApplication()`.

Application Controller component reference

This section includes details and examples about the following components: Forge, Dgidx, Dgraph, LogServer, and ReportGenerator.

Forge

A Forge element launches the Forge (Data Foundry) software, which transforms source data into tagged Endeca records.

Every Application Controller component contains the following attributes:

Attribute	Description
component-id	Required. The name of this instance of the component.
host-id	Required. The alias of the host upon which the component is running.
properties	An optional list of properties, consisting of a required name and an optional value.

The Forge element contains the following sub-elements:

Sub-element	Description

args	Command-line flags to pass to Forge, expressed as a set of arg sub-elements. If an argument takes a value, the argument and value must be on separate lines in the provisioning file. For example:
	`<args>`
	`  <arg>--threads</arg>`
	`  <arg>3</arg>`
	`</args>`

input-dir	The path to the Forge input.
log-file	Name of the Forge log file. If the log-file is not specified, the default is component working directory plus component name plus ".log".
output-prefix-name	The implementation-specific prefix name, without any associated path information.
output-dir	Directory where the output from the Forge process will be stored.
pipeline-file	Required. Name of the Pipeline.epx file to pass to Forge.
num-partitions	The number of partitions.
working-dir	Working directory for the process that is launched. If it is specified, it must be an absolute path. If any of the other properties of this component contain relative paths, they are interpreted as relative to the working directory. If working-dir is not specified, it defaults to `$ENDECA_CONF/work/<appName>/ <componentName>` on UNIX, or `%ENDECA_CONF%\work\<appName>/ <componentName>` on Windows.
state-dir	The directory where the state file is located.
temp-dir	The temporary directory that Forge uses.
web-service-port	The port on which the Forge metrics Web service listens.
ssl-configuration	Both the parallel Forge and Forge metrics Web service can secure their communications with SSL. The `ssl-configuration` element contains three sub-elements of its own: • `cert-file`: The `cert-file` specifies the path of the eneCert.pem certificate file that is used by Forge processes to present to any client.

Sub-element	Description
	This is also the certificate that the Application Controller Agent should present to Forge when trying to talk to it. The file name can be a path relative to the component's working directory. • `ca-file`: The `ca-file` specifies the path of the eneCA.pem Certificate Authority file that Forge processes uses to authenticate communications with other Endeca components. The file name can be a path relative to the component's working directory. • `cipher`: Specify one or more cryptographic algorithms, one of which Dgraph will use during the SSL negotiation. If you omit this setting, the Dgraph chooses a cryptographic algorithm from its internal list of algorithms. See the *Endeca Commerce Security Guide* for more information.

Example

The following example provisions a Forge component for use with the sample wine data:

```
<forge component-id="wine_forge" host-id="wine_indexer">
 <args>
  <arg>-vw</arg>
 </args>
 <num-partitions>1</num-partitions>
 <working-dir>
   C:\Endeca\PlatformServices\reference\sample_wine_data
 </working-dir>
 <pipeline-file>.\data\forge_input\pipeline.epx</pipeline-file>
 <input-dir>.\data\forge_input</input-dir>
 <output-dir>.\data\partition0\forge_output</output-dir>
 <state-dir>.\data\partition0\state</state-dir>
 <log-file>.\logs\wine_forge.log</log-file>
 <output-prefix-name>wine</output-prefix-name>
</forge>
```

Dgidx

A Dgidx component sends the finished data prepared by Forge to the Dgidx program, which generates the proprietary indices for each Dgraph.

Every Application Controller element contains the following attributes:

Attribute	Description
component-id	Required. The name of this instance of the component.
host-id	Required. The alias of the host upon which the component is running.
properties	An optional list of properties, consisting of a required name and an optional value.

The Dgidx element contains the following sub-elements:

Sub-element	Description

args	Command-line flags to pass to Dgidx, expressed as a set of arg sub-elements. If an argument takes a value, the argument and value must be on separate lines in the provisioning file. For example:
	`<args>`
	`  <arg>--threads</arg>`
	`  <arg>3</arg>`
	`</args>`

app-config-prefix	Path and file prefix that define the input for Dgidx. For example, in `/endeca/project/files/myProject`, files beginning with myProject in the directory `/endeca/project/files` are the ones to be considered.
output-prefix	Required. Path and prefix name for the Dgidx output. For example, `output_prefix = c:\temp\wine` generates files that start with “wine” in the `c:\temp` directory.
log-file	The path to and name of the Dgidx log files. If the log-file is not specified, the default is component working directory plus component name plus “.log”. Dgidx can generate three distinct log files: the basic component log file, and two files that log the subtasks described in run-aspell, below. • The file dgwordlist logs stdout/stderr for the dgwordlist subtask described below. The name of this file is derived from the Dgidx component’s log-file location, plus the term “dgwordlist”. If an extension exists, “dgwordlist” is added before the extension. For example, if the original log-file is `C:\dir\dgidx-1.log`, then the dgwordlist log would be `C:\dir\dgidx-1.dgwordlist.log`. • The file aspellcopy logs the stdout/stderr for the subtask of uploading the Aspell files to Dgidx’s output directory, where the Dgraph can access them. The name of this file is derived from the Dgidx component’s log-file location, plus the term “aspellcopy”. If an extension exists, “aspellcopy” is added before the extension. For example, if the original log-file is `C:\dir\dgidx-1.txt`, then the aspellcopy log would be `C:\dir\dgidx-1.aspellcopy.txt`.
input-prefix	Required. Path and prefix name for the Forge output that Dgidx indexes.
working-dir	Working directory for the process that is launched. If it is specified, it must be an absolute path. If any of the other properties of this component contain relative paths, they are interpreted as relative to the working directory. If working-dir is not specified, it defaults to `$ENDECA_CONF/work/<appName>/<componentName>` on UNIX, or `%ENDECA_CONF%\work\<appName>/<componentName>` on Windows.
run-aspell	Specifies Aspell as the spelling correction mode for the implementation. This causes the Dgidx component to run dgwordlist and to copy the Aspell files to its output directory, where the Dgraph component can access them. The default is true. See log-file above for details on the logging of these subtasks. For Aspell details, see the *Endeca Advanced Development Guide*.
temp-dir	A temporary directory used by this component.

Example

The following example provisions a Dgidx component to work with the sample wine data:

```
<dgidx component-id="wine_dgidx" host-id="wine_indexer">
 <args>
  <arg>-v</arg>
 </args>
 <working-dir>
   C:\Endeca\PlatformServices\reference\sample_wine_data
 </working-dir>
 <input-prefix>.\data\partition0\forge_output\wine</input-prefix>
 <app-config-prefix>
   .\data\partition0\forge_output\wine
 </app-config-prefix>
 <output-prefix>.\data\partition0\dgidx_output\wine</output-prefix>
 <log-file>.\logs\wine_dgidx.log</log-file>
 <run-aspell>true</run-aspell>
</dgidx>
```

Dgraph

A Dgraph element launches the Dgraph (MDEX Engine) software, which processes queries against the indexed Endeca records.

Every Application Controller component contains the following attributes:

Attribute	Description
component-id	Required. The name of this instance of the component.
host-id	Required. The alias of the host upon which the component is running.
properties	An optional list of properties, consisting of a required name and an optional value.

The Dgraph element contains the following sub-elements:

Sub-element	Description

args	Command-line flags to pass to Dgraph, expressed as a set of arg sub-elements. If an argument takes a value, the argument and value must be on separate lines in the provisioning file. For example:
	`<args>`
	`  <arg>--threads</arg>`
	`  <arg>3</arg>`
	`</args>`

port	Required. The port at which the Dgraph should listen. The default is 8000.
log-file	The path to and name of the Dgraph log file. If the log-file is not specified, the default is component working directory plus component name plus ".log".
input-prefix	Required. Path and prefix name for the Dgidx output that the Dgraph uses as an input.

Sub-element	Description
app-config-prefix	Path and file prefix that define the input for the Dgraph. For example, in `/endeca/project/files/myProject`, files beginning with myProject in the directory `/endeca/project/files` are the ones to be considered.
working-dir	Working directory for the process that is launched. If it is specified, it must be an absolute path. If any of the other properties of this component contain relative paths, they are interpreted as relative to the working directory. If working-dir is not specified, it defaults to `$ENDECA_CONF/work/<appName>/<componentName>` on UNIX, or `%ENDECA_CONF%\work\<appName>/<componentName>` on Windows.
startup-timeout	Specifies the amount of time in seconds that the Application Controller waits while starting the Dgraph. If it cannot determine that the Dgraph is running in this timeframe, it times out. The default is 60.
req-log-file	Path to and name of the request log.
spell-dir	If specified, is the directory in which the Dgraph will look for Aspell files. If it is not specified, the Dgraph will look for Aspell files in the Dgraph's input directory (that is, input-prefix without the prefix). For example, if input-prefix is /dir/prefix and all the Dgraph input files are /dir/prefix.*, the Dgraph will look for the Aspell files in /dir/.
update-dir	Specifies the directory from which the Dgraph reads partial update file. For more information, see the *Endeca Partial Updates Guide*.
update-log-file	Specifies the file for update-related log messages.
temp-dir	A temporary directory used by this component.
ssl-configuration	Contains three sub-elements of its own: • `cert-file`: The `cert-file` specifies the path of the eneCert.pem certificate file that is used by the Dgraph to present to any client. This is also the certificate that the Application Controller Agent should present to the Dgraph when trying to talk to the Dgraph. The file name can be a path relative to the component's working directory. • `ca-file`: The `ca-file` specifies the path of the eneCA.pem Certificate Authority file that the Dgraph uses to authenticate communications with other Endeca components. The file name can be a path relative to the component's working directory. • `cipher`: Specify one or more cryptographic algorithms, one of which Dgraph will use during the SSL negotiation. If you omit this setting, the Dgraph chooses a cryptographic algorithm from its internal list of algorithms. See the *Endeca Commerce Security Guide* for more information..

Example

The following example provisions an SSL-enabled Dgraph component for use with the sample wine data:

```
<dgraph component-id="wine_dgraph" host-id="wine_indexer">
 <args>
   <arg>--spl</arg>
   <arg>--dym</arg>
 </args>
 <port>8000</port>
```

```
 <working-dir>
   C:\Endeca\PlatformServices\reference\sample_wine_data
 </working-dir>
 <input-prefix>.\data\partition0\dgraph_input\wine</input-prefix>
 <app-config-prefix>
   .\data\partition0\dgraph_input\wine
 </app-config-prefix>
 <log-file>.\logs\wine_dgraph.log</log-file>
 <req-log-file>.\logs\wine_dgraph_req_log.out</req-log-file>
 <startup-timeout>120</startup-timeout>
 <ssl-configuration>
   <cert-file>
     C:\Endeca\PlatformServices\workspace\etc\eneCert.pem
   </cert-file>
   <ca-file>
     C:\Endeca\PlatformServices\workspace\etc\eneCA.pem
   </ca-file>
   <cipher>AES128-SHA</cipher>
 </ssl-configuration>
</dgraph>
```

LogServer

The LogServer component controls the use of the Endeca Log Server.

Every Application Controller component contains the following attributes:

Attribute	Description
component-id	Required. The name of this instance of the component.
host-id	Required. The alias of the host upon which the component is running.
properties	An optional list of properties, consisting of a required name and an optional value.

The LogServer component contains the following sub-elements:

Sub-element	Description
ort	Required. Port on which to run the LogServer.
output-prefix	Required. Path and prefix name for the LogServer output. For example, `output_prefix = c:\temp\wine` generates files that start with “wine” in `c:\temp`.
gzip	Required. Controls the archiving of log files. Possible values are true and false.
working-dir	Working directory for the process that is launched. If it is specified, it must be an absolute path. If any of the other properties of this component contain relative paths, they are interpreted as relative to the working directory. If working-dir is not specified, it defaults to `$ENDECA_CONF/work/<appName>/<componentName>` on UNIX, or `%ENDECA_CONF%\work\<appName>/<componentName>` on Windows.
startup-timeout	Specifies the amount of time in seconds that the eaccmd waits while starting the LogServer. If it cannot determine that the LogServer is running in this timeframe, it times out. The default is 60.

Sub-element	Description
log-file	The path to the LogServer log file. If the log-file is not specified, the default is component working directory plus component name plus ".log".

Example

The following example provisions a LogServer component based on the sample wine data.

```
<logserver component-id="wine_logserver" host-id="wine_indexer">
 <port>8002</port>
 <working-dir>
  C:\Endeca\PlatformServices\reference\sample_wine_data
 </working-dir>
 <output-prefix>.\logs\logserver_output\wine</output-prefix>
 <gzip>false</gzip>
 <startup-timeout>120</startup-timeout>
 <log-file>.\logs\wine_logserver.log</log-file>
</logserver>
```

ReportGenerator

The ReportGenerator component runs the Report Generator, which processes Log Server files into HTML-based reports that you can view in your Web browser and XML reports that you can view in Endeca Workbench.

Every Application Controller component contains the following attributes:

Attribute	Description
component-id	Required. The name of this instance of the component.
host-id	Required. The alias of the host upon which the component is running.
properties	An optional list of properties, consisting of a required name and an optional value.

The ReportGenerator component contains the following sub-elements:

Sub-element	Description
working-dir	Working directory for the process that is launched. If it is specified, it must be an absolute path. If any of the other properties of this component contain relative paths, they are interpreted as relative to the working directory. If working-dir is not specified, it defaults to `$ENDECA_CONF/work/<appName>/<componentName>` on UNIX, or `%ENDECA_CONF%\work\<appName>/<componentName>` on Windows.
input-dir-or-file	Required. Path to the file or directory containing the logs to report on. If it is a directory, then all log files in that directory are read. If it is a file, then just that file is read.
output-file	Required. Name the generated report file and path to where it is stored. For example: `C:\Endeca\reports\myreport.html` on Windows `/endeca/reports/myreport.html` on UNIX

Sub-element	Description
stylesheet-file	Required. Filename and path of the XSL stylesheet used to format the generated report. For example: `%ENDECA_CONF%\etc\ report_stylesheet.xsl` on Windows `$ENDECA_CONF/etc/report_stylesheet.xsl` on UNIX
settings-file	Path to the report_settings.xml file. For example: `%ENDECA_CONF%\etc\report_settings.xml` on Windows `$ENDECA_CONF/etc/report_settings.xml` on UNIX
timerange	Sets the time span of interest (or report window). Allowed keywords: • Yesterday • LastWeek • LastMonth • DaySoFar • WeekSoFar • MonthSoFar These keywords assume that days end at midnight, and weeks end on the midnight between Saturday and Sunday.
start-date <date> stop-date <date>	These set the report window to the given date and time. The date format should be either `yyyy_mm_dd` or `yyyy_mm_dd.hh_mm_ss`. For example, `2009_10_23.19_30_57` expresses Oct 23, 2009 at 7:30:57 in the evening.
time-series	Turns on the generation of time-series data and specifies the frequency, Hourly or Daily.
charts	Turns on the generation of report charts. Disabled by default.
log-file	The path to the ReportGenerator log file. If the log-file is not specified, the default is component working directory plus component name plus ".log".
java_binary	Should indicate a JDK 1.5.x or later. Defaults to the JDK that Endeca installs.
java_options	Command-line options for the java_binary setting. This command is primarily used to adjust the ReportGenerator memory, which defaults to 1GB. To set the memory, use the following: `java_options = -Xmx[MemoryInMb]m -Xms[MemoryInMb]m`
args	Command-line flags to pass to the ReportGenerator, expressed as a set of arg sub-elements.

Example

The following example provisions a ReportGenerator component based on the sample wine data.

```
<reportgenerator component-id="wine_gen_html_report" host-id="wine_indexer">
 <working-dir>
  C:\Endeca\PlatformServices\reference\sample_wine_data
 </working-dir>
 <input-dir-or-file>.\logs\logserver_output</input-dir-or-file>
 <output-file>.\reports\daily\daily_report.html</output-file>
```

```
 <stylesheet-file>.\etc\report_stylesheet.xsl</stylesheet-file>
 <settings-file>.\etc\report_settings.xml</settings-file>
 <timerange>day-so-far</timerange>
 <charts>true</charts>
 <log-file>.\logs\wine_gen_html_report.log</log-file>
</reportgenerator>
```

Provisioning your implementation with eaccmd

You can use the `eaccmd` command-line interface to create an implementation based on the provisioning file you created.

To provision your implementation:

1. Create a provisioning document as described above.
2. Run `eaccmd` with the --define-app command, specifying the provisioning document you created in step 1. For example:
   ```
   eaccmd localhost:8888 define-app --app myApp --def app.xml
   ```

Related Links

Using the eaccmd Tool on page 45
This section describes the eaccmd command-line tool, which can be used to provision and run the Endeca Application Controller.

Provisioning the Application Controller to work on multiple machines

Typically, you provision the Application Controller to work in a distributed environment. You do this by defining the implementation appropriately and then starting the components on the provisioned delegate machines.

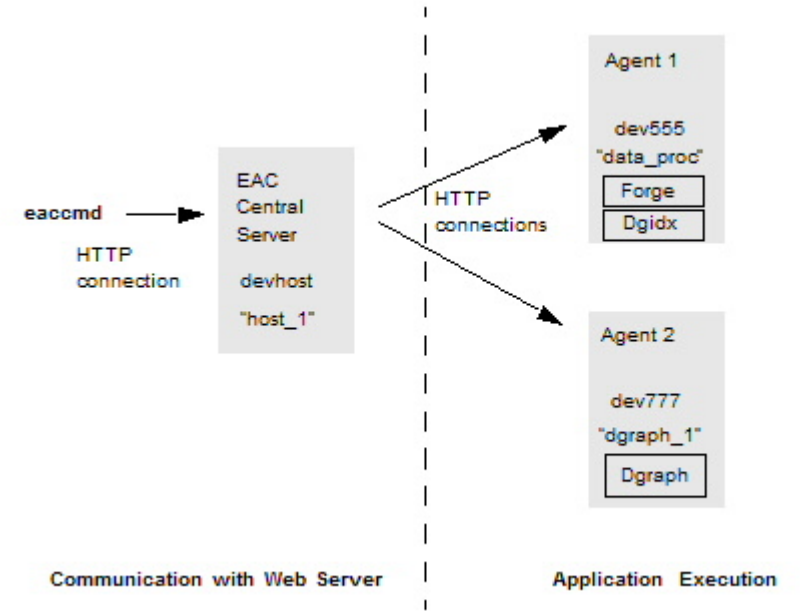
The following steps walk you through multi-machine provisioning and execution using the Application Controller.

1. Write a provisioning document for the EAC Central Server in which you define all of the components and their corresponding host machines. Save this document as app.xml.
2. Run `eaccmd` on the host_1 machine, using the app.xml provisioning document as follows:
   ```
   eaccmd devhost:8888 define-app --app myApp --def app.xml
   ```
3. To start the component Forge on machine data_proc, issue this `eaccmd` command on host_1:
   ```
   eaccmd devhost:8888 start --app myApp --comp forge
   ```
4. To start the component Dgidx on machine data_proc, issue this eaccmd command on host_1:
   ```
   eaccmd devhost:8888 start --app myApp --comp dgidx
   ```
5. To start the component Dgraph on machine dgraph_1, issue this `eaccmd` command on host_1:
   ```
   eaccmd devhost:8888 start --app myApp --comp dgraph
   ```

Multi-machine example

The example below illustrates how provisioning and running the Application Controller work in multi-machine environments. In this scenario, there are three machines: devhost, which serves as the EAC Central Server, and dev555 and dev777, which serve as Agent machines running Forge and Dgraph respectively. The Application Controller is installed identically on each machine. Eaccmd is run on devhost (aliased host_1), using HTTP port 8888.

Eaccmd issues commands to the EAC Central Server, which in turn passes them on to Agent machines dev555 (aliased data_proc) and dev777 (aliased dgraph_1) via HTTP. The EAC Central Server machine, devhost, handles all direct communication with the user, while the Agent machines execute application tasks.



Note: EAC task tokens (names or IDs) must be unique across an application. If two tasks have the same token (such as "copy_index_files_to_dgraph_server"), and exist on separate EAC Agent machines, you cannot run both instances of this task simultaneously.

Forcing the removal of an application

You remove an application in eaccmd with the remove-app command.

If you want to remove an application that is throwing an error (for example, because it contains a host or component that has become unreachable), or one with running utilities or components, you must add the --force flag. The syntax is as follows:

```
remove-app --force --app app_id
```

In a WSDL tool, this behavior is controlled by the forceRemove property on the RemoveApplicationType class.

About incremental provisioning

With incremental provisioning, it is possible to add, remove, or modify one or more hosts, components, or scripts without having to bring down the entire implementation.

You can perform incremental provisioning in eaccmd or your custom Web service tool. We use eaccmd in the examples below.

Related Links

About eaccmd on page 45
When you manage your Endeca implementation with the Endeca Application Controller, you control and monitor its working through the EAC Central Server.

Endeca Application Controller API Interface Reference on page 61
Application Controller interfaces are documented here. However, the exact syntax of a class member depends on the output of the WSDL tool that you are using. Be sure to check the client stub classes that are generated by your WSDL tool for the exact syntax of the Application Controller API class members.

Incrememental provisioning guidelines

The following guidelines apply to incremental provisioning.

- Scripts can be changed at any time, as long as they are not running.
- Properties on either hosts or components can be changed at any time.
- Anything other than a property on a component cannot be changed, nor can a component be removed, if the component is either running or unreachable.
- Anything other than a property or a directory on a host cannot be changed, nor can a host be removed, if any components or utilities on it are running, or if the host is unreachable.

You can attempt to override the constraints mentioned above by using the `--force` flag.

About the def_file setting

The def_file is the provisioning document used to add a component or host to the implementation.

You can use a larger provisioning file for this purpose, or you can use one that specifies exactly one component or host. If you choose to use a larger provisioning file, then you must specify which component or host listed within it that you are adding.

For example, say you want to add a host called new_host to your application. You could add provisioning information for new_host to your existing provisioning file, `myApp.xml`. When you run the add-host command, you would give it the host name as well as the provisioning file name.

In the case of scripts, you have two options: you can use a def_file, as you do with components and hosts, or you can provide the necessary information individually, through the `--cmd` (command), `--wd` (working directory), and `--log-file` settings.

About the --force flag

The --force flag indicates whether or not the Application Controller should attempt to force any running components, utilities, or scripts to stop before attempting an update or a remove operation.

In the case of updates, the update persists in the application provisioning, regardless of whether or not the forced stop was successful, even if this leaves a dangling process somewhere.

Examples

- In the case of a component, the command:

```
update-component --force --app myApp --name forge
```

  would first stop the component forge, if it is running, before updating it.

- In the case of a host, the command:

  ```
  remove-host --force --app myApp --name dev777
  ```

  would first stop any running components or services on host dev777 before removing that host.
- In the case of a script, the command:

  ```
  update-script --force --app myApp --script newbaseline.pl
  --cmd perl
  ```

  would first stop the script newbaseline.pl before updating it.

Adding a component in eaccmd

You can use eaccmd to add components to your application.

To add a component in eaccmd, use the following syntax:

```
add-component --app app_id [--comp comp_id] --def def_file
```

For example:

```
add-component --app myApp --comp new_forge --def myApp.xml
```

Removing a component in eaccmd

You can use eaccmd to remove components from applications.

To remove a component in eaccmd, use the following syntax:

```
remove-component [--force] --app app_name --comp comp_id
```

For example:

```
remove-component --force --app myApp --comp forge
```

Modifying a component in eaccmd

You can use eaccmd to modify components in an application.

To change the attributes of a previously-defined component in eaccmd, use the following syntax:

```
update-component [--force] --app app_id [--comp comp_id]
--def def_file
```

For example:

```
update-component --force --app myApp --def newDgraphProps.xml
```

Adding a host in eaccmd

You can use eaccmd to add hosts to your application.

To add a host in eaccmd, use the following syntax:

```
add-host --app app_id [--host host_id] --def def_file
```

For example:

```
add-host --app myApp --host mktg022 --def myApp.xml
```

Removing a host in eaccmd

You can use eaccmd to remove hosts from an application.

To remove a host in eaccmd, use the following syntax:

```
remove-host [--force] --app app_id --host host_id
```

For example:

```
remove-host --force --app myApp --host dev777
```

Modifying a host in eaccmd

You can use eaccmd to modify hosts in an application.

To change the attributes of a previously-defined host in eaccmd, use the following syntax:

```
update-host [--force] --app app_id [--host host_id]
--def def_file
```

For example:

```
update-host --force --app myApp --host mktg022
--def newMktgHostProps.xml
```

Adding a script in eaccmd

You can use eaccmd to add scripts to your application.

To add a script in eaccmd, use the following syntax:

```
add-script --app app_id --script script_id [--cmd command --wd working_dir --log-
file log_file] | [--def def_file]
```

For example:

```
add-script --app myApp --script newbaseline.pl --cmd perl
```

Removing a script in eaccmd

You can use eaccmd to remove scripts from applications.

To remove a script in eaccmd, use the following syntax:

```
remove-script [--force] --app app_id --script script_id
```

For example:

```
remove-script --app myApp --script testbaseline.pl
```

Modifying a script in eaccmd

You can use eaccmd to modify a script in an application.

To modify an existing script in eaccmd, use the following syntax:

```
update-script [--force]  --app app_id --script script_id [--cmd command --wd
working_dir --log-file log_file] | [--def def_file]
```

For example:

```
update-script --app myApp --script newbaseline.pl --def myApp.xml
```

Provisioning your deployment with the Endeca Deployment Template

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

Representing the best practices of Endeca's Customer Solutions organization, the template includes the complete directory structure required for deployment, including EAC scripts, configuration files, and batch files or shell scripts that wrap common script functionality.

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

Using the Endeca Deployment Template

The Endeca Deployment Template should be installed immediately following the installation of Oracle Endeca Commerce on all servers that will be hosting Oracle Endeca Commerce components, and before any provisioning has been done through Endeca Workbench.

If Endeca Workbench has been used to make any changes to Oracle Endeca Commerce configuration prior to installing the Endeca Deployment Template, they will be overwritten and lost.

Chapter 4

Common System Architectures in an Endeca Implementation

This section describes typical system architectures for each stage of an Endeca implementation.

Overview of system architectures

This topic provides a general description of typical system architectures for each stage of an Endeca implementation.

Endeca implementations typically have three stages:

1. Development
2. Staging and testing
3. Production

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

Development environment

A development environment is one in which developers create or substantially modify an Endeca implementation.

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

Staging and testing environment

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

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

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

Sample production environments

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

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

Descriptions of implementation size

We can roughly divide implementations into small, medium, and large.

A full definition of these terms includes an accounting of record size (number and size of properties and dimension values per record), total data set size, the number of indexing and MDEX Engine servers, and other measurements of scale.

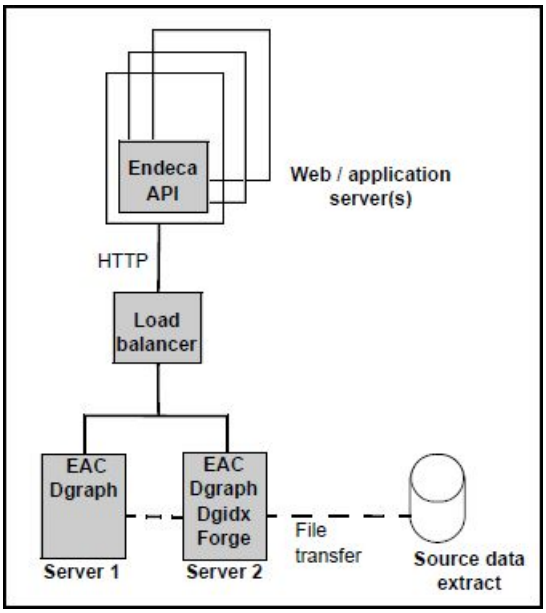
Although that level of detail is necessary for sizing a specific implementation, it is not necessary for the more general discussion of hardware architecture here. For simplicity's sake, this chapter uses the terms small and medium e as follows:

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

Small implementation with lower throughput

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

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



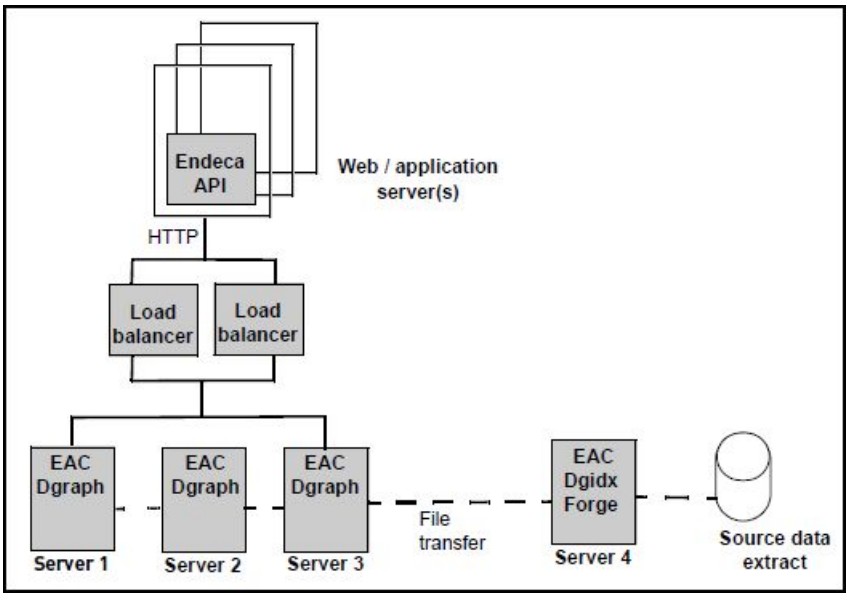
The advantage of this scenario is low cost and MDEX Engine redundancy. If one MDEX Engine is offline for any reason, the load balancer distributes user queries to the other MDEX Engine.

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

Medium implementation with higher throughput

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

To achieve higher throughput, servers 1, 2, and 3 all run mirror copies of the MDEX Engine. This level of redundancy provides faster throughput by load balancing the incoming queries over a greater number of MDEX Engines. If either load balancer or any MDEX Engine should fail, then the redundant load balancer and remaining MDEX Engines handle all queries. Server 4 runs all the offline processes including Forge and Dgidix.



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

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

Chapter 5

Using the eaccmd Tool

This section describes the eaccmd command-line tool, which can be used to provision and run the Endeca Application Controller.

About eaccmd

When you manage your Endeca implementation with the Endeca Application Controller, you control and monitor its working through the EAC Central Server.

You can communicate with the EAC Central Server in two ways:

- With the eaccmd command-line tool, as described in this chapter.
- Through direct programmatic control with a language that understands Web services.

The Application Controller's WSDL API is described in the "Endeca Application Controller API Interface Reference."

Running eaccmd

This topic describes how to run eaccmd.

The eaccmd tool is installed by default in `%ENDECA_ROOT%\bin` on Windows. On UNIX, it is `$ENDECA_ROOT/bin`. You run eaccmd within a scripting environment such as Bash or Perl. You can run eaccmd on any machine as long as it is pointing at the EAC Central Server.

The eaccmd syntax is platform-independent.

Related Links

eaccmd usage on page 45
This topic describes the usage of eaccmd.

eaccmd usage

This topic describes the usage of eaccmd.

The eaccmd usage is as follows:

```
eaccmd host:eac_port <cmd> [--async] [-verbose]
```

where settings in square brackets ([ ]) are optional and **<cmd>** is one of:

```
[Provisioning commands:]
  define-app [--app app_id] [--def def_file]
  describe-app --app app_id [--canonical]
  remove-app [--force] --app app_id
  list-apps
[Incremental Provisioning commands:]
  add-component --app app_id [--comp comp_id] --def def_file
  add-host --app app_id [--host host_id] --def def_file
  add-script --app app_id --script script_id (--def def_file |
    [--wd working_dir] [--log-file log_file] --cmd command [args...])
  remove-component [--force] --app app_id --comp comp_id
  remove-host [--force] --app app_id --host host_id
  remove-script --app app_id --script script_id
  update-component [--force] --app app_id [--comp comp_id] --def def_file
  update-host [--force] --app app_id [--host host_id] --def def_file
  update-script [--force] --app app_id --script script_id
    (--def def_file | [--wd working_dir] [--log-file log_file]
    --cmd command [args...])
[Synchronization commands:]
  set-flag --app app_id --flag flag
  remove-flag --app app_id --flag flag
  remove-all-flags --app app_id
  list-flags --app app_id
[Component and Script Control commands:]
  start --app app_id [--comp comp_id | --script script_id]
  stop --app app_id [--comp comp_id | --script script_id]
  status --app app_id [--comp comp_id | --script script_id]
[Utility commands:]
  ls --app app_id --host host_id --pattern file_pattern
  start-util --type shell --app app_id [--token token]
    --host host_id [--wd working_dir] --cmd command [args...]
  start-util --type copy --app app_id [--token token] [--recursive]
    --from host_id --to host_id --src src_path --dest dest_path
  start-util --type backup --app app_id [--token token] --host host_id
    --dir ls [--method <copy|move>] [--backups num_backups]
  start-util --type rollback --app app_id [--token token] --host host_id
    --dir ls
  stop-util --app app_id --token token
  status-util --app app_id --token token
```

eaccmd feedback

Eaccmd gives no feedback in cases of success (that is, if a component is running or completed or a service is completed).

If an operation fails, a FAILED message is printed to the screen.

If instead you want eaccmd to run asynchronously, you must use the `--async` flag on the command line after the command, as follows:

```
eaccmd host:port <cmd> [--async]
```

Component and utility status verbosity

By default, eaccmd provides single-word component and utility status messages, such as Running. To receive more detailed feedback, you can run eaccmd with the --verbose flag.

This flag provides useful information beyond simply the state.

Server component status verbosity

The following is an example of a verbose status message for a server component. Server components include the Dgraph and LogServer.

```
State: NotRunning
Start time: 10/11/08 3:58 PM
Failure Message:
```

Batch component status verbosity

The following is an example of a verbose status message for a batch component. Batch components include Forge, Dgidx, and ReportGenerator.

```
State: NotRunning
Start time: 10/11/08 3:58 PM
Duration: 0 days 0 hours 0 minutes 6.96 seconds
Failure Message:
```

Using the default host and port

The `eaccmd.properties` file supplies host and port information to eaccmd.

In the `eaccmd.properties` file, which is located in the `$ENDECA_CONF/conf` directory on UNIX and `%ENDECA_CONF%\conf` on Windows, you can specify a host and port for eaccmd to use. (The default values are host=localhost and port=8888.) With this file in place, you do not have to specify the host and port on the command line.

If your EAC Central Server is not on localhost:8888, you must either edit the file to point to the correct host and port or continue to specify host:port on the command line. Any host:port specified on the command line overrides the settings in the `eaccmd.properties` file.

eaccmd command reference

The eaccmd tool contains commands for provisioning, resource configuration, and component use.

Provisioning commands

The provisioning commands make it possible for you to define and manage your applications from the command line.

Command	Description
define-app [--app app_id] [--def def_file]	Defines an application. Def_file takes an XML provisioning file, a sample of which,

Command	Description
	sample_wine_definition.xml, is located in the %ENDECA_REFERENCE_DIR%\sample_wine_data\etc directory on Windows, or the $ENDECA_REFERENCE_DIR\sample_wine_data\etc directory on UNIX. The provisioning file typically contains an application ID. If eaccmd specifies a different app_id for the same application, the eaccmd version overrides the one in provided in the provisioning file.
describe-app --app app_id [--canonical]	Describes an application. Returns an XML file in the format used by the def_file setting of define-app. If --canonical is specified, all paths are canonicalized.
remove-app [--force] --app app_id	Removes the named application. The optional --force flag indicates whether or not this remove operation should force any running components or services to stop before attempting the remove. Remove fails if any components or services are still running (that is, not forced to stop).
list-apps	Lists all defined applications.

Provisioning example

The following example defines an application called my_wine. (In this and all examples that follow we assume that the host and port are set in the eaccmd.properties file and so do not need to be included on the command line.)

```
eaccmd define-app --app my_wine --def sample_wine_definition.xml
```

Incremental provisioning commands

The incremental provisioning commands make it possible for you to add, remove, or update a host, component, or script without having to bring down the entire application.

Command	Description
add-component --app app_id [--comp comp_id] --def def_file	Adds a single component to an application. Def_file is a provisioning document. You can use a larger provisioning file for this purpose, or you can use one that specifies exactly one component or host. If you choose to use a larger provisioning file, then you must specify which component listed within it that you are adding, using the --comp flag.
add-host --app app_id [--host host_id] --def def_file	Adds a single host to an application. Def_file is a provisioning document. You can use a larger provisioning file for this purpose, or you can use one that specifies exactly one component or host. If you choose to use a larger provisioning file, then you must specify which host listed within it that you are adding, using the --host flag.

Command	Description
add-script --app app_id --script script_id (--def def_file | [--wd working_dir] [--log-file log_file] --cmd command [args...])	Adds a script to an application. Scripts can be added at any time. You can use `--def` to specify a definition file to start the script, or use the following settings: `--log-file` is the file for appended stdout/stderr output. If it is not specified, it defaults to `$ENDECA_CONF/logs/script/(app_id).(script_id).log` `--wd` is the working directory. If it is not specified, it defaults to `$ENDECA_CONF/working/(app_id)/` `--cmd` is the command that is used to start the script. If `--cmd` is omitted, the first unrecognized argument is taken as the start of your command. The `--log-file` and `--wd`, if used, should come before `--cmd`.
remove-component [--force] --app app_id --comp comp_id	Removes a single component from an application. The optional --force flag indicates whether or not this remove operation should force any running components or services to stop before attempting the remove. Remove fails if any components or services are still running (that is, not forced to stop).
remove-host [--force] --app app_id --host host_id	Removes a single host from an application. The optional `--force` flag indicates whether or not this remove operation should force any running components or services to stop before attempting the remove. Remove fails if any components or services are still running (that is, not forced to stop).
remove-script [--force] --app app_id --script script_id	Removes a script from an application. The optional `--force` flag indicates whether or not this remove operation should force a running script to stop before attempting the remove.
update-component [--force] --app app_id [--comp comp_id] --def def_file	Updates a component. Component properties can be updated at any time. Other changes cannot be made if the component is running or unreachable. The optional `--force` flag indicates that the Application Controller will attempt to force the conditions under which the specified updates can be made (by stopping stop a running component or utility invocation, for example). Regardless of whether or not the forced stop is successful, however, the update persists in the application provisioning, even if this leaves a dangling process somewhere.
update-host [--force] --app app_id [--host host_id] --def def_file	Updates a host. Host properties can be updated at any time. Other changes cannot be made if any components or services are running on the host, or if the host is unreachable. The optional `--force` flag indicates that the Application Controller will attempt to force the conditions under which the specified updates

Command	Description
	can be made (by stopping stop a running component or utility invocation, for example). Regardless of whether or not the forced stop is successful, however, the update persists in the application provisioning, even if this leaves a dangling process somewhere.

update-script [--force] --app app_id --script script_id (--def def_file | [--wd working_dir] [--log-file log_file] --cmd command [args...])	Updates a script. The optional `--force` flag indicates whether or not this update operation should force a running script to stop before attempting the update. You can use --def to specify a definition file to update the script, or use the following settings:
	`--wd` is the working directory. If it is not specified, it defaults to `$ENDECA_CONF/working/(app_id)/`
	`--log-file` is the file for appended stdout/stderr output. If it is not specified, it defaults to `$ENDECA_CONF/logs/script/(app_id).(script_id).log`
	`--cmd` is the command that is used to start the script. If `--cmd` is omitted, the first unrecognized argument is taken as the start of your command. The `--log-file` and `--wd`, if used, should come before `--cmd`.

Incremental provisioning example

The following example adds a Forge component to the my_wine application. Because this provisioning file contains only a single component, it is not necessary to use the `--comp` flag.

```
eaccmd add-component --app my_wine --def update_forge.xml
```

Synchronization commands

Synchronization commands are used by the Synchronization service (described below) to manage application-level flags that let users know when processes are in use.

Command	Description
set-flag --app app_id --flag flag	Sets a flag that demonstrates that a group of processes are in use. You specify the flag with the application name and a flag name, which may be arbitrary but should be well-known.
remove-flag --app app_id --flag flag	Removes the named flag and releases the reserved processes.
remove-all-flags --app app_id	Removes all flags in an application and releases all reserved processes.
list-flags --app app_id	Lists all flags in an application.

About the Synchronization service

The Synchronization service lets you create, query, and delete application-level flags on a series of processes. These flags indicate that the flagged processes are in use. The service creates flags on the fly at the user's request and deletes them when they are released. Using this service, multiple users can synchronize their activities by obtaining and querying the flags. If two users attempt to flag the same processes at the same time an error occurs.

Synchronization service flags are identified by an application name/flag name pair. Because flag names are user-created and arbitrary, all users must be aware of flag names and consistent in their use. If a set of processes needs to be reserved, then everyone concerned needs to know the name of the flag.

Synchronization examples

The following example adds a flag called mkt1010 to the my_wine application:

```
eaccmd set-flag --app my_wine --flag mkt1010
```

The following example removes all flags in the my_wine application:

```
eaccmd remove-all-flags --app my_wine
```

Component and script control commands

The component and script control commands are used to start and stop components or scripts and retrieve their status.

Command	Description
start --app app_id [--comp comp_id | --script script_id]	Starts a component or a script.
stop --app app_id [--comp comp_id | --script script_id]	Stops a component or a script.
status --app app_id [--comp comp_id | --script script_id]	Gets the status of a component (one of Starting, Running, NotRunning, or Failed) or a script (one of Running, NotRunning, or Failed).

Component control example

The following example starts a Dgraph named wine_dgraph in the my_wine application.

```
eaccmd start --app my_wine --comp wine_dgraph
```

Utility commands

The utility commands allow you to run and monitor Application Controller utilities through the eaccmd tool.

There are three kinds of Utility commands: Shell, Copy, and Archive.

General notes on Application Controller utilities

Keep in mind the following general points about Application Controller utilities.

- Utility naming: Be sure to name your utilities carefully. If you create a new utility that has the same name as a running utility, an error is issued. However, if there is an existing utility with the same name that is not running, the new utility overwrites it.
- System cleanup of utility output: Each instance of the Shell and Copy utilities stores status information and output logs. The Application Controller clears this information for non-running utilities instances every seven

days (that is, 10,080 minutes) to save system resources. This setting can be modified in the eac.properties file.

The List Directory Contents (ls) command

The List Directory Contents command lets you see the contents of directories on remote machines. Its behavior is similar to that of ls on UNIX, although some non-ls restrictions, noted below, apply.

Command	Description
ls --app app_id --host host_id --pattern file_pattern	Returns a list of files matching the pattern input in file_pattern. Note the following: A file_pattern must start with an absolute path, such as C:\ or /. A file_pattern can contain . or .. as directory names, and expands * and ? wildcards. A file_pattern cannot contain the wildcard expressions .*, .?, or ..* as directory or file names. Bracketed wildcards, such as file[123].txt, are not supported. Wildcards cannot be applied to drive names. You cannot use .. to create paths that do not exist. For example, the path `/temp/../../a.txt` refers to a path that is above the root directory. This is an invalid path that causes the operation to fail.

Wildcard behavior

The List Directory Contents command expands the wildcards in a pattern. If the expansion results in a file, it returns a file. If the expansion results in a directory, it returns the directory non-recursively. Wildcard expansion can result in any combination of files and directories.

For example, assume that the following directories and files exist:

```
/home/endeca/reference/...
/home/endeca/install.log
/home/e.txt
```

The following command:

```
eaccmd ls --app my_wine --host my_host --pattern /home/e\\*
```

would list all of these files and directories, because they match the file_pattern.

Delimiting wildcard arguments

To prevent inappropriate expansion, any wildcard arguments you use with the List Directory Contents utility in eaccmd need to be delimited with double quotation marks. For example: On Windows, "C:*.txt". On UNIX, "/home/endeca/test/*.txt".

The Shell utility

The Shell utility allows you to run arbitrary commands in a host system shell.

Command	Description
start-util --type shell --app app_id [--token token] --host host_id [--wd working_dir] --cmd command [args...]	Starts a Shell utility with the specified command string. The token is a string. If you do not specify a token, one is generated and returned when you start the utility. The token is used to stop the utility or to get its status.

Command	Description
	--wd, which is optional, sets the working directory for the process that gets launched. If specified, it must be an absolute path. If wd is not specified, the setting defaults to `%ENDECA_CONF%\working\ <appName>\shell` on Windows or `$ENDECA_CONF/working/ <appName>/shell` on UNIX. The --cmd arguments are passed in a single string. If --cmd is omitted, the first unrecognized argument is taken as the start of your command.
stop-util --app app_id --token token	Stops a Shell utility. The token is a string, either user-created or generated and returned when you start the utility, that eaccmd prints to screen. The token can be used to stop the utility or to get its status.
status-util --app app_id --token token	Gets the status of a Shell utility. The token is a string, either user-created or generated and returned when you start the utility, that eaccmd prints to screen. The token can be used to stop the utility or to get its status.

Shell utility examples

The first example deletes the Dgidx output after it has been copied in a separate action over to the Dgraph:

```
eaccmd start-util --type shell --app my_wine --host mkt1010
--cmd rm <dgidx-output-dir>/*.*
```

The second example performs a recursive directory copy:

```
eaccmd start-util --type shell --app myapp --host hosttorunon
--cmd cp–r /mysourcedir /mydestdir
```

Troubleshooting the Shell utility

In many cases, particularly cross-platform scenarios, the Shell command must be wrapped in double quotation marks. The error message returned, which occurs at the console level, is usually something similar to the following:

```
The system cannot find the path specified.
```

The Copy utility

The Copy utility uses an internal Web services interface to copy files or directories, either locally or between machines.

Commands	Description
start-util --type copy --app app_id [--token token] [--recursive] --from host_id --to host_id --src file_pattern --dest dest_path	As part of the Copy utility, starts a copy. You identify the hostname, port, and path for both the source and destination directories. If the copy is local, you do not need to specify the host_id. Keep in mind that you are not necessarily copying to the machine you are running eaccmd on. The hosts you are copying to and from are those you specified in your provisioning file.

Commands	Description
	`--token` is a string used to stop the utility or get its status. If you do not specify a token, one is generated and returned when you start the utility. If `--recursive` is specified, it indicates that the Copy utility recursively copies any directories that match the wildcard. If `--recursive` is not specified, the Copy utility does not copy directories, even if they match the wildcard. Instead, it creates intermediate directories required to place the copied files at the destination path. `--src` is a string representing the file, wildcard, or directory to be copied. A `--src` must start with an absolute path, such as C:\ or /. A --src can contain . or .. as directory names, and expands * and ? wildcards. Note the following: • You cannot use the wildcard expressions .*, .?, or ..* as directory or file names. • Bracket wildcards, such as file[123].txt, are not supported. • Wildcards cannot be applied to drive names. `--dest` is the full path to the destination file or directory. `--dest` must be an absolute path, and no wildcards are allowed. If `--dest` is a directory, that directory must exist, unless the following conditions are met: • The parent of the destination already exists. • You are copying only one thing.
stop-util --app app_id --token token	Stops a Copy utility. The token is a string, either user-created or generated and returned when you start the utility, that eaccmd prints to screen. The token can be used to stop the utility or to get its status.
status-util --app app_id --token token	Gets the status of a Copy utility. The token is a string, either user-created or generated and returned when you start the utility, that eaccmd prints to screen. The token can be used to stop the utility or to get its status.

Copy utility examples

This section illustrates several different Copy actions. For simplicity, the majority of the Copy actions are done on a single machine. The final example shows how to copy across machines.

First, assume the following directory structure exists on the source:

```
/
 endeca1/
  work/
   dgraphlogs/
```

```
    a.log
   forgelogs/
    b.log
 endeca2/
  work/
   dgraphlogs/
    c.log
   forgelogs/
    d.log
    e.log
 destination/
```

The following command copies one file to a new name:

```
eaccmd start-util --type copy --app myApp
   --src "/endeca1/work/dgraphlogs/a.log" --dest "/destination/out.log"
```

The resulting directory change would look like this:

```
 destination/
  out.log
```

The following command copies one file into an existing directory:

```
eaccmd start-util --type copy --app myApp
   --src "/endeca1/work/dgraphlogs/a.log" --dest "/destination"
```

The resulting directory change would look like this:

```
destination/
 a.log
```

The following command recursively copies a directory to a new name:

```
eaccmd start-util --type copy --app myApp
   --src "/endeca1/work/dgraphlogs" --dest "/destination/outlogs" --recursive
```

The resulting directory change would look like this:

```
destination/
 outlogs/
  a.log
```

The following command recursively copies a directory into an existing directory:

```
eaccmd start-util --type copy --app myApp
   --src "/endeca1/work/dgraphlogs" --dest "/destination"
   --recursive
```

The resulting directory change would look like this:

```
destination/
 dgraphlogs/
  a.log
```

The following command copies all files in a directory.

```
eaccmd start-util --type copy --app myApp
   --src "/endeca2/work/forgelogs/*" --dest "/destination"
```

The resulting directory change would look like this:

```
destination/
  d.log
  e.log
```

The following copy command demonstrates the use of multiple wildcards:

```
eaccmd start-util --type copy --app myApp
   --src "/e*/work/*logs/*.log" --dest "/destination"
```

The resulting directory change would look like this:

```
destination/
  a.log
  b.log
  c.log
  d.log
  e.log
```

The following copy demonstrates a recursive copy with wildcards:

```
eaccmd start-util --type copy --app myApp
   --src "/e*/work" --dest "/destination" --recursive
```

The resulting directory change would look like this:

```
destination/
 work/
  dgraphlogs/
   a.log
   c.log
  forgelogs/
   b.log
   d.log
   e.log
```

When copying to another machine, the syntax is as follows:

```
eaccmd start-util --type copy --app myApp --from ITLHost --to MDEXHost
   --src /full/path/to/file/src.txt --dest /full/path/to/file/dest.txt
```

Keep in mind that the hostnames are not IP addresses or DNS names, but rather are the hosts that are defined within the EAC. If you are using the Deployment Template, these are the hosts defined in the `AppConfig.xml` file with tags similar to this example:

```
<host id="ITLHost" hostName="itl.example.com" port="8888" />
<host id="MDEXHost" hostName="mdex.example.com" port="8888" />
```

Also make sure that you have a clear network path between hosts (if necessary, make the appropriate modifications in any firewall to allow traffic).

About the Copy utility

This topic provides details about how the Copy utility works.

The Copy utility supports wildcards (* and ?) and recursive copying. In some cases, the destination directory must already exist; in others, the utility automatically creates both the destination directory and any empty directories in the transfer.

Directories are copied first to a temporary directory on the destination machine before being copied one file at a time to the target location. You can configure the location of this temporary directory in the eac.properties file, using the optional `setting com.endeca.eac.filetransfer.fileTransferTempDir` as follows:

- If this setting is defined as an absolute path, the Copy utility uses it.
- If it is defined as a relative path, the Copy utility considers it to be relative to `%ENDECA_CONF%/state/`
- If it is not defined, the Copy utility uses the directory `%ENDECA_CONF%/state/file_transfer/`

If the Copy utility tries to copy a file to a location where another file already exists, the utility overwrites the preexisting file.

Note: The Copy utility supports both SSL and non-SSL communication, with SSL being off by default. For details on enabling SSL, see the *Endeca Security Guide*.

Destination directories

In most cases, the destination directory where the copied files are placed has to exist already. However, there are a few exceptions where the destination directory does not have to exist prior to the copy:

- Copying just one file to the location of an existing file.
- Copying just one file to a new file name in an existing directory.
- Copying just one directory to a new directory name in an existing parent directory.

Failure and recovery

The following situations result in a failure of the Copy utility:

- The Copy utility tries to write to a directory it doesn't have permissions to.
- There is not enough disk space.
- There is no file at the source location.
- The wildcard expression matches no files.
- When there are mismatches between directories and files (for example, the Copy utility tries to copy a file to path where a directory with that name already exists, or tries to create a directory in the destination and a file with that name already exists).
- You cannot use .. to create paths that do not exist. For example, the path `/temp/../../a.txt` refers to a path that is above the root directory. This is an invalid path that causes the utility to fail.
- Asking for a copy that results in multiple files being written to the same location. For example, given the following directory structure on the source:

  ```
  /trunk/src/a.txt
  /testbranch/src/a.txt
  ```

  a copy from /t*/src/* to /temp would result in the Copy utility trying to write both a.txt files to the same location in the temp directory.

There is no recovery for copies. Therefore, if the transfer of a large file fails, the entire file must be transferred again. Likewise, if a multi-file transfer fails before completion, you must either re-run the entire transfer or request only those parts that did not transfer.

Explicit machine naming

Keep in mind that when you are using the Copy utility, you are potentially working with three machines: the EAC Central Server, from which you issue eaccmd commands, the Agent machine you are copying data from, and the one you are copying data to. In such cases, the name localhost can be confusing. Unless you are using the Copy utility to move files on a single machine, you should use explicit machine names rather than simply localhost.

Delimiting wildcard elements

To prevent inappropriate expansion, any wildcard arguments you use with the Copy utility in eaccmd need to be delimited with double quotation marks. For example:

On Windows, "C:*.txt".

On UNIX, "/home/endeca/test/*.txt".

Copying across platforms

If you are copying files or directories between machines on different platforms, you have to wrap any Window paths on a Linux or Solaris shell in double quotation marks (for example, "C:*.txt").

The Archive utility

The Archive utility allows you to archive and roll back directories.

Using the Archive utility, you can save off and back up a set of component outputs, which later can be rolled back on demand. With the backup operation, you create back up copies of directories distinguished by time stamps. With the rollback operation, you replace the current version of a directory with the most recently backed-up version. The current version is then renamed with an .unwanted suffix.

Note: Do not start a backup or rollback operation while another such operation is in progress on the same directory. Unexpected behavior may occur if you do so.

Related Links

Backup operations on page 58
Backup operations create an archive directory from an existing directory.

Rollback operations on page 59
Rollback operations roll back the directory to the most recent backed up version.

Backup operations

Backup operations create an archive directory from an existing directory.

Backup operations create an archive directory from an existing directory. The archive directory has the same name as the original directory, but with a timestamp appended to the end. The timestamp reflects the time when the backup operation was performed.

For example, if the original directory is called logs and was backed up on October 11, 2008 at 8:00 AM, the backup operation creates a directory called logs.2008_10_11.08_00_00.

Command	Description
start-util --type backup --app app_id [--token token] --host host-id --dir dir [--method] <copy|move> [--backups num_backups]	Starts the backup operation. The token is a string. If you do not specify a token, one is generated and returned when you start the utility. The token is used to stop the utility or to get its status. The host and dir settings specify the path to the directory that will be archived. The method is either copy or move (the default). The optional backups setting specifies the maximum number of archives to store. This number does not include the original directory itself, so if backups is set to 3, you would have the original directory plus up to three archive directories, for a total of as many as four directories. The default num_backups is 5.
stop-util --app app_id --token token	Stops a backup operation. The token is a string, either user-created or system-generated when you start the utility. The token can be used to stop the utility or to get its status.

Command	Description
status-util --app app_id --token token	Gets the status of a backup operation. The token is a string, either user-created or system-generated when you start the utility. The token can be used to stop the utility or to get its status.

Backup operation example

In the following example, an archive version of the logs directory is created.

```
eaccmd start-util --type backup --app my_wine --host mkt1010
--dir c:\my_wine\data\logs --backups 2
```

Rollback operations

Rollback operations roll back the directory to the most recent backed up version.

For example, say you have a directory called logs, one called logs.2008_10_11.08_00_00, and other, older versions. When you roll back, the following things happen:

- logs is renamed logs.unwanted.
- logs.2008_10_11.08_00_00 is renamed logs.
- The older versions are left alone.

Note: There can only be a single .unwanted directory at a time. If you roll back twice, the .unwanted directory from the first rollback is deleted.

Command	Description
start-util --type rollback --app app_id [--token token] --host host_id --dir dir	Starts the rollback operation. The token is a string. If you do not specify a token, one is generated and returned when you start the utility. The token is used to stop the utility or to get its status. The host and dir settings specify the path to the directory that will be rolled back.
stop-util --app app_id --token token	Stops a rollback operation. The token is a string, either user-created or generated and returned when you start the utility, that eaccmd prints to screen. The token can be used to stop the utility or to get its status.
status-util --app app_id --token token	Gets the status of a rollback operation. The token is a string, either user-created or generated and returned when you start the utility, that eaccmd prints to screen. The token can be used to stop the utility or to get its status.

Rollback operation example

In the following example, the archived logs directory is rolled back.

```
eaccmd start-util --type rollback --app my_wine --host mkt1010
--dir c:\my_wine\data\logs
```

Chapter 6

Endeca Application Controller API Interface Reference

Application Controller interfaces are documented here. However, the exact syntax of a class member depends on the output of the WSDL tool that you are using. Be sure to check the client stub classes that are generated by your WSDL tool for the exact syntax of the Application Controller API class members.

Using the Application Controller WSDL

You can use the Endeca Application Controller WSDL API to write your application in the language of your choice (such as Java, C#, or Perl).

Using the Web Services tool of your choice (such as Axis for Java), do the following:

1. Run the WSDL through your tool to generate the stubs (that is, an API that your code can call).
2. Write your application, using that code to control the Application Controller.

Note:

- The Application Controller schema is defined in eac.wsdl, which is located in the `$ENDECA_ROOT/lib/services` directory on UNIX and `%ENDECA_ROOT%\lib\services` on Windows.
- You generate clent stubs (or proxies) using the eac.wsdl file located in the file system provided by the Endeca installation. You cannot generate client stubs using the SOAP Web services addresses associated with each service within the WSDL file.

Simple types in the Application Controller WSDL

The Application Controller WSDL defines several data types that can be treated as simple data types.

- IDType, TokenType, BackupMethodType, TimeRangeType, and TimeSeriesType can be treated as Strings.
- PortNumber can be treated as an Integer.
- TimeOut can be treated as a Long.

ComponentControl interface

The ComponentControl interface provides component management capabilities.

It consists of the following methods:

startComponent(FullyQualifiedComponentIDType startComponentInput)

Starts the named component.

FullyQualifiedComponentIDType parameters:

- applicationID identifies the application to use.
- componentID identifies the component to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

stopComponent(FullyQualifiedComponentIDType stopComponentInput)

Stops the named component.

FullyQualifiedComponentIDType parameters:

- applicationID identifies the application to use.
- componentID identifies the component to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Synchronization interface

The Synchronization interface manages application-level flags that let users know when processes are in use.

For example, your code could create a flag named update-running to ensure that a new baseline update does not start while another update is already in progress.

Typical usage is as follows:

```
if (setFlag(MY_FLAG_ID) == true)
 [perform action, such as a baseline update]
 removeFlag(MY_FLAG_ID)
else
 [signal error such as "an update is already in progress"]
```

setFlag(FullyQualifiedFlagIDType setFlagInput)

Creates a new flag, identified by flagID, that is associated with the named application.

FullyQualifiedFlagIDType parameters:

- applicationID identifies the application to use.
- flagID is a unique string identifier for this flag.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- A Boolean, false if the flag was already set, or true if it was not set meaning the method succeeded).

removeFlag(FullyQualifiedFlagIDType removeFlagInput)

Removes the named flag.

FullyQualifiedFlagIDType parameters:

- applicationID identifies the application to use.
- flagID is a unique string identifier for this flag.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

removeAllFlags(IDType removeAllFlagsInput)

Removes all flags in an application.

IDType parameter:

- applicationID identifies the application to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

listFlags(IDType listFlagsInput)

Lists the collection of flags in an application.

IDType parameter:

- applicationID identifies the application to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- flagIDList, a string collection of flagIDs.

Utility interface

The Utility interface allows you to manage the Application Controller utilities (Shell, Copy, and Archive) programmatically.

Note: Be sure to name your utilities carefully. If you create a new utility that has the same name as a running utility, an error is issued. However, if there is an existing utility with the same name that is not running, the new utility overwrites it.

The Utility interface consists of the following methods:

startBackup(RunBackupType startBackupInput)

Starts the backup operation of the Archive utility.

Backup operations create an archive directory from an existing directory. The archive directory has the same name as the original directory, but with a timestamp appended to the end. The timestamp reflects the time when the backup operation was performed.

For example, if the original directory is called logs and was backed up on October 11, 2008 at 8:00 AM, the backup operation creates a directory called logs.2008_10_11.08_00_00.

Note: Do not start a backup or rollback operation while another such operation is in progress on the same directory. Unexpected behavior may occur if you do so.

RunBackupType parameters:

- applicationID identifies the application to use.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- hostID is a unique identifier for the host. The hostID and dirName parameters specify the path to the directory that will be archived.
- dirName is the full path of the directory. The hostID and dirName parameters specify the path to the directory that will be archived.
- backupMethod is either copy or move.
- numBackups specifies the maximum number of archives to store. This number does not include the original directory itself, so if numBackups is set to 3, you would have the original directory plus up to three archive directories, for a total of as many as four directories. The default numBackups is 5.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- The string token assigned to this invocation.

startFileCopy(RunFileCopyType startFileCopyInput)

Launches the Copy utility, which copies files either on a single machine or between machines.

RunFileCopyType parameters:

- applicationID identifies the application to use.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- fromHostID is a unique identifier for the host from which you are copying.
- toHostID is a unique identifier for the host to which you are copying.
- sourcePath is a string representing the file, wildcard, or directory to be copied. A sourcePath must start with an absolute path, such as C:\ or /. A sourcePath can contain . or .. as directory names, and expands * and ? wildcards. Note the following:
- You cannot use the wildcard expressions .*, .?, or ..* as directory or file names.
- Bracket wildcards, such as file[123].txt, are not supported.
- Wildcards cannot be applied to drive names.
- destinationPath is the full path to the destination file or directory. destinationPath must be an absolute path, and no wildcards are allowed.

The destination directory must exist, unless the parent of the destination already exists and you are copying only one thing.

- recursive, when true, indicates that the Copy utility recursively copies any directories that match the wildcard.

  If recursive is false, the Copy utility does not copy directories, even if they match the wildcard. Instead, it creates intermediate directories required to place the copied files at the destination path.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- The string token assigned to this invocation.

startRollback(RunRollbackType startRollbackInput)

Rollback operations roll back the directory to the most recent backed up version.

For example, say you have a directory called logs, one called logs.2008_10_11.08_00_00, and other, older versions. When you roll back, the following things happen:

- logs is renamed logs.unwanted.
- logs.2008_10_11.08_00_00 is renamed logs.
- The older versions are left alone.

Note: There can only be a single .unwanted directory at a time. If you roll back twice, the .unwanted directory from the first rollback is deleted.

Note: Do not start a backup or rollback operation while another such operation is in progress on the same directory. Unexpected behavior may occur if you do so.

RunRollbackType parameters:

- applicationID identifies the application to use.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- hostID is a unique identifier for the host. The hostID and dirName parameters specify the path to the directory that will be archived.
- dirName is the full path of the directory. The hostID and dirName parameters specify the path to the directory that will be archived.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- The string token assigned to this invocation.

startShell(RunShellType startShellInput)

The startShell() method launches the Shell utility, which allows you to run arbitrary commands in a host system shell.

RunShellType parameters:

- applicationID identifies the application to use.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- hostID is a unique identifier for the host.
- cmd is the command line to execute.
- workingDir is the full path to the working directory.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- The string token assigned to this invocation.

stop(FullyQualifiedUtilityTokenType)

Takes a token returned by any of the start methods, and stops that invocation by terminating the process that is running it.

FullyQualifiedUtilityTokenType parameters:

- applicationID identifies the application to use.
- token identifies the token used to stop the utility.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

getStatus(String applicationID, String token)

Takes a token returned by any of the Utility start methods (startBackup(), startFileCopy(), startRollback(), or startShell()), and returns the current status of that utility.

Parameters:

- applicationID identifies the application to use.
- token identifies the token used to get the utility's status.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- A BatchStatusType object.

listDirectoryContents(ListDirectoryContentsInputType listDirectoryContentsInput)

Performs a list operation similar to UNIX ls on a remote host, with the following restrictions on the input file pattern.

- A filePattern must start with an absolute path, such as C:\ or /.
- A filePattern can contain . or .. as directory names, and expands * and ? wildcards.
- A filePattern cannot contain the wildcard expressions .*, .?, or ..* as directory or file names.
- Bracketed wildcards, such as file[123].txt, are not supported.
- Wildcards cannot be applied to drive names.
- You cannot use .. to create paths that do not exist. For example, the path /temp/../../a.txt refers to a path that is above the root directory. This is an invalid path that causes the operation to fail.

ListDirectoryContentsInputType parameters:

- applicationID (required) identifies the application to use.
- hostID (required) is a unique identifier for the host.
- filePattern (required) is the name of the directory, file, or wildcard combination of directory and file whose contents are to be listed.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails. Failure conditions correspond to bad input cases.

Returns:

- A FilePathListType object representing the contents of the requested directory.

Provisioning interface

The Provisioning interface allows you to define and manage your Endeca applications programmatically.

It contains the following methods:

defineApplication(ApplicationType application)

Defines an application.

ApplicationType parameters:

- applicationID identifies the application to use.
- hosts is a collection of HostType objects, representing the hosts to define.
- components is a collection of ComponentType objects (such as ForgeComponentType, DgraphComponentType, and so on) representing the components to define.
- scripts is a collection of ScriptType objects.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

Related Links

ComponentType class on page 77
A class that describes the base type for all components within an application.

ScriptType class on page 88
A class that describes the base type for all scripts within an application.

getApplication(IDType getApplicationInput)

Gets an application, which is composed of hosts, components, and scripts and identified by an application ID.

IDType parameter:

- applicationID identifies the application to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- An ApplicationType object.

Related Links

ApplicationType class on page 76
A class that describes an application to be deployed by the Application Controller. An application is composed of a set of components residing on a set of hosts.

getCanonicalApplication(IDType getCanonicalApplicationInput)

The getCanonicalApplication() method returns the provisioning just as getApplication() does, but with all paths canonicalized.

This process ensures that all paths are absolute, and that the working directory and log path settings are provided. It also prevents .. from being used in a path name.

IDType parameter:

- applicationID identifies the application to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- An ApplicationType object, as described on page 248.

Related Links

ApplicationType class on page 76
A class that describes an application to be deployed by the Application Controller. An application is composed of a set of components residing on a set of hosts.

listApplicationIDs(listApplicationIDsInput)

Lists the applications that are defined.

Returns:

- An ApplicationIDListType object.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Related Links

ApplicationIDListType class on page 76
A class that describes a returned value of a list application call to the Provisioning service.

removeApplication(RemoveApplicationType removeApplicationInput)

Removes the named application.

RemoveApplicationType parameter:

- applicationID identifies the application to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

addComponent(AddComponentType addComponentInput)

Adds a single component to an application.

AddComponentType parameters:

- applicationID identifies the application to use.
- component is one of the following: Forge, Dgidx, Dgraph, LogServer, ReportGenerator

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

removeComponent(RemoveComponentType removeComponentInput)

Removes a single component from an application.

RemoveComponentType parameters:

- applicationID identifies the application to use.

- componentID identifies the component to use.
- forceRemove indicates whether or not a remove operation should force the component to stop before attempting the remove. If the component is running, and forceRemove is not set to true, then the remove call will fail.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

updateComponent(UpdateComponentType updateComponentInput)

Updates a running component.

UpdateComponentType parameters:

- applicationID identifies the application to use.
- component is one of the following: Forge, Dgidx, Dgraph, LogServer, ReportGenerator.
- forceUpdate indicates that the Application Controller will attempt to force the conditions under which the update can take place, by stopping running components.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

addHost(AddHostType addHostInput)

Adds a host to an application.

AddHostType parameters:

- applicationID identifies the application to use.
- host is a HostType object specifying the host to add.
- directories allows you to specify directories using a full path and a name. These directories are associated with hosts and created when the host is provisioned.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

updateScript(UpdateScriptType updateScriptInput)

Updates a running script.

UpdateScriptType parameters:

- applicationID identifies the application to use.
- script is a ScriptType object specifying the script to be updated.
- forceUpdate is a Boolean that indicates whether the Application Controller should force a running script to stop before attempting the update.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

removeHost(RemoveHostType removeHostInput)

Removes a single host from an application.

RemoveHostType parameters:

- applicationID identifies the application to use.
- hostID is a unique string identifier for this host.
- forceRemove indicates whether or not the Application Controller should force any running components or services to stop before attempting the remove. If a component or service is running, and forceRemove is not set to true, then the remove call will fail.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

updateHost(UpdateHostType updateHostInput)

Updates a running host.

UpdateHostType parameters:

- applicationID identifies the application to use.
- host is a HostType object specifying the host to add.
- directories allows you to specify directories using a full path and a name. These directories are associated with hosts and created when the host is provisioned.
- forceUpdate indicates that the Application Controller will attempt to force the conditions under which the update can take place, by stopping running components or services.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

addScript(AddScriptType addScriptInput)

Adds a script to an application.

AddScriptType parameters:

- applicationID identifies the application to use.
- script is a ScriptType object (see page 269) specifying the script to add.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

Related Links

ScriptType class on page 88
A class that describes the base type for all scripts within an application.

removeScript(RemoveScriptType removeScriptInput)

Removes a script from an application.

RemoveScriptType parameters:

- applicationID identifies the application to use.
- scriptID is a unique string identifier for this host.
- forceRemove indicates that the Application Controller will attempt to force the conditions under which the remove can take place.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.
- ProvisioningFault is a list of provisioning errors and a list of provisioning warnings thrown when there are fatal errors during provisioning.

Returns:

- A ProvisioningWarningListType object, containing minor warnings about non-fatal provisioning problems.

ScriptControl interface

The ScriptControl interface provides programmatic script management capabilities.

It contains the following methods:

startScript(FullyQualifiedScriptIDType startScriptInput)

Starts the named script.

FullyQualifiedScriptIDType parameters:

- applicationID identifies the application to use.
- scriptID identifies the script to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

stopScript(FullyQualifiedScriptIDType stopScriptInput)

Stops the named script.

FullyQualifiedScriptIDType parameters:

- applicationID identifies the application to use.
- scriptID identifies the script to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

getScriptStatus(FullyQualifiedScriptIDType getScriptStatusInput)

Returns the status of a script.

FullyQualifiedScriptIDType parameters:

- applicationID identifies the application to use.
- scriptID identifies the script to use.

Throws:

- EACFault is the error message returned by the Application Controller when the method fails.

Returns:

- A ScriptStatus object (a sub-class of the StatusType class). This status may be Running, NotRunning, or Failed. (Failure results from a failure error code or internal EAC errors).

Related Links

StatusType class on page 89
Describes the status of a server component in the Application Controller.

Chapter 7

Endeca Application Controller API Class Reference

This section describes the Endeca Application Controller API classes.

About Endeca Application Controller API Classes

The Endeca Application Controller API classes and their properties are documented here. However, the exact syntax of a class member depends on the output of the WSDL tool that you are using.

Typically, a Java WSDL tool translates these classes into get and set methods. For example, the ApplicationIDType class would generate getApplicationID() and setApplicationID(String[] applicationID) methods.

The Microsoft .NET WSDL tool translates these classes into .NET properties. Be sure to check the client stub classes that are generated by your WSDL tool for the exact syntax of the Application Controller API class members.

AddComponentType class

A class that describes a component to be added to a named application during incremental provisioning.

AddComponentType properties

- applicationID (required) identifies the application to use.
- component (required) is one of the following: Forge, Dgidx, Dgraph, LogServer, or ReportGenerator.

AddHostType class

A class that describes a host to be added to a named application during incremental provisioning.

AddHostType properties

- applicationID (required) identifies the application to use.
- host (required) is a description of the host to add.
- directories allows you to specify directories using a full path and a name.

AddScriptType class

A class that describes a script to be added to a named application during incremental provisioning.

AddScriptType properties

- applicationID (required) identifies the application to use.
- script (required) is a description of the script to add.

ApplicationIDListType class

A class that describes a returned value of a list application call to the Provisioning service.

ApplicationIDListType encapsulates the list of applications running on this EAC Central Server.

ApplicationIDListType properties

- applicationID identifies the application to use.

ApplicationType class

A class that describes an application to be deployed by the Application Controller. An application is composed of a set of components residing on a set of hosts.

You can construct an ApplicationType object as a full specification of the application, including all hosts and components. Alternatively, you can start with an empty an ApplicationType object and incrementally fill in the hosts, components, and scripts. In the latter case, order matters, because a host must be added before you add a component that lives on that host.

ApplicationType properties

- applicationID identifies the application to use.
- hosts is a list of hosts.
- components is a list of components.
- scripts is a list of scripts.

BackupMethodType class

In relation to the Archive utility, this class serves as an identifier for the type of backup you want the utility to perform, Copy or Move.

BackupMethodType fields

The enumeration of possible values is as follows:

- Copy.
- Move.

BatchStatusType class

Based on the StatusType class, a BatchStatusType object describes the status of a batch component.

Batch components include Forge, Dgidx and ReportGenerator..

BatchStatusType properties

- StateType – (required) An enumeration of the following fields:

  Starting, which only applies to server components (Dgraph or LogServer)

  Running

  NotRunning

  Failed

- startTime – (required) The time the batch component started; for example, 10/11/08 3:58 PM.
- failureMessage – The failure message, which tells you that a failure has occurred in the execution of the component. failureMessage is empty unless state is FAILED. (This is different from EACFault, which tells you that a problem has occurred while processing the Web Service request to get the status.)
- duration – (required) The length of time the batch component has been running; for example, 0 days 0 hours 0 minutes 6.96 seconds.

Related Links

StatusType class on page 89
Describes the status of a server component in the Application Controller.

ComponentListType class

A class that describes a list of components, such as ForgeComponentType and DgraphComponentType.

ComponentListType properties

- component (required) A collection of components comprising this ComponentListType object.

ComponentType class

A class that describes the base type for all components within an application.

ComponentType properties

Each component contains these properties, as well as some others.

- componentID (required) identifies the component to use.
- hostID (required) is a unique string identifier for this host.
- workingDir is a string identifying the working directory for this component.
- logFile is a string identifying the log file for this component.
- properties is a string identifying any properties associated with this component.

DgidxComponentType class

A class that describes a Dgidx component within an application.

A Dgidx component sends the finished data prepared by Forge to the Dgidx program, which generates the proprietary indices for each Dgraph.

DgidxComponentType properties

- componentID (required) identifies the component to use.
- hostID (required) is a unique string identifier for this host.
- workingDir is a string identifying the working directory for this component. Any relative paths in component properties are be interpreted as relative to the component's workingDir. The workingDir property, if specified, must be an absolute path.
- logFile is a string identifying the log file for this component.
- args is a list of command-line flags to pass to Dgidx.
- appConfigPrefix is the path and file prefix that define the input for Dgidx.
- inputPrefix (required) is the path and prefix name for the Forge output that Dgidx indexes.
- outputPrefix (required) is the path and prefix name for the Dgidx output.
- runAspell prepares the Aspell files for the Dgraph. The default is true, which causes the Dgidx component to run dgwordlist and to copy the Aspell files to its output directory, where the Dgraph component can access them. Note that your stub generation tool may generate a Boolean property (for example, runAspellSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.
- tempDir is the path to the temporary directory that Dgidx uses.

DgraphComponentType class

A class that describes a Dgraph component within an application.

A Dgraph element launches the Dgraph (MDEX Engine) software, which processes queries against the indexed Endeca records.

DgraphComponentType properties

- componentID (required) identifies the component to use.
- hostID (required) is a unique string identifier for this host.
- workingDir is a string identifying the working directory for this component. Any relative paths in component properties are be interpreted as relative to the component's workingDir. The workingDir property, if specified, must be an absolute path.
- logFile is a string identifying the log file for this component.
- args is a list of command-line flags to pass to the Dgraph.
- port (required) is the port the Dgraph listens at. The default is 8000.

- appConfigPrefix is the path and file prefix that define the input for Dgraph.
- inputPrefix (required) is the path and prefix name for the Dgidx output that the Dgraph uses as an input.
- reqLogFile is the path to and name of the request log.
- spellDir, if specified, is the directory in which the Dgraph will look for Aspell files. If it is not specified, the Dgraph will look for Aspell files in the Dgraph's input directory (that is, inputPrefix without the prefix). For example, if inputPrefix is /dir/prefix and all the Dgraph input files are /dir/prefix.*, the Dgraph will look for the Aspell files in /dir/).
- startupTimeout specifies the amount of time in seconds that the Application Controller will wait while starting the Dgraph. Note that your stub generation tool may generate a Boolean property (for example, startupTimeoutSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.
- sslConfiguration sets SSL usage for this Dgraph.
- updateDir is the directory from which Dgraph reads partial update files. For more information, see the *Endeca Partial Updates Guide*.
- updateLogFile specifies the file for update-related log messages.
- tempDir is the path to the temporary directory that the Dgraph uses.

DirectoryListType class

A class that represents a collection of DirectoryType objects.

DirectoryListType property

- directory (required) is a collection of DirectoryType objects.

DirectoryType class

A class used by the HostType class to define a directory while provisioning a host.

DirectoryType properties

- dirID (required) is a unique identifier for this directory.
- dir (required) is a full path for this directory.

EACFault class

The class that creates the EACFault. EACFault is the error message returned by the Application Controller when the method fails.

EAC Fault property

- error is the error message.

FilePathListType class

An array of FilePathTypes that describes a returned value of a listDirectoryContents call.

FilePathListType operates on the application level.

FilePathListType property

- filePaths (required) describe a file on a remote host.

FilePathType class

A class that describes a file on a remote host.

FilePathType properties

- path (required) is the full path to the file.
- directory (required) indicates whether the path is a directory.

FlagIDListType class

A class that describes a returned value of a list flags call. FlagIDListType operates on the application level.

FlagIDListType property

- flagID is a unique string identifier for this flag.

ForgeComponentType class

A class that describes a Forge component within an application.

A Forge element launches the Forge (Data Foundry) software, which transforms source data into tagged Endeca records.

ForgeComponentType properties

- componentID (required) identifies the component to use.
- hostID (required) is a unique string identifier for this host.
- workingDir is a string identifying the working directory for this component. Any relative paths in component properties are be interpreted as relative to the component's workingDir. The workingDir property, if specified, must be an absolute path.
- logFile is a string identifying the log file for this component.
- args is a list of command-line flags to pass to Forge.
- stateDir is the directory where the state file is located.

- inputDir is the path to the Forge input.
- outputDir is the directory where the output from the Forge process will be stored.
- outputPrefixName is the prefix, without any associated path information, that Forge uses to save its output files. These files are located in the directory specified by outputDir.
- numPartitions is the number of partitions. Note that your stub generation tool may generate a Boolean property (for example, numPartitionsSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.
- pipelineFile (required) is the name of the Pipeline.epx file to pass to Forge.
- tempDir is the temporary directory that Forge uses.
- webServicePort is the port used by the Forge metrics Web service, which provides progress and performance metrics for Forge. For details, see "The Forge Metrics Web Service" in the *Endeca Forge Guide*. Note that your stub generation tool may generate a Boolean property (for example, webServicePortSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

FullyQualifiedComponentIDType class

A class that serves as an input to the start, stop, get status, and remove component commands.

FullyQualifiedComponentIDType properties

- applicationID (required) identifies the application to use.
- componentID (required) identifies the component to use.

FullyQualifiedFlagIDType class

In relation to the Synchronization service, this class serves as an input to an acquire or release flag method.

FullyQualifiedFlagIDType properties

- applicationID (required) identifies the application to use.
- flagID (required) is a unique string identifier for this flag.

FullyQualifiedHostIDType class

A class that identifies a host so that it can be used as an input to another command, such as remove host.

FullyQualifiedHostIDType properties

- applicationID (required) identifies the application to use.

- hostID (required) is a unique string identifier for this host.

FullyQualifiedScriptIDType class

A class that identifies a script so that it can be used as an input to another command, such as startScript().

FullyQualifiedScriptIDType properties

- applicationID (required) identifies the application to use.
- scriptID (required) is a unique string identifier for this script.

FullyQualifiedUtilityTokenType class

In relation to the Utility service, this object represents the token.

FullyQualifiedUtilityTokenType properties

- applicationID (required) identifies the application to use.
- token (required) identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.

HostListType class

A class that represents a collection of HostType objects.

HostListType property

- host (required) is a unique identifier comprising a hostname, port, and hostID.

HostType class

A class that describes a host within an application.

Along with components, a collection of HostType objects define an application.

HostType properties

- hostname (required) is the name of the host.
- port (required) is the connection port.
- hostID is a unique string identifier for this host.
- directories allows you to specify directories using a full path and a name.

ListApplicationIDsInput class

An empty object you pass into the Web services interface to get back a list of applications.

ListDirectoryContentsInputType class

An object that serves as an input to the listDirectoryContents object.

ListDirectoryContentsInputType properties

- applicationID (required) identifies the application to use to look up the host.
- hostID (required) is a unique identifier for the host within that application.
- filePattern (required) is the pattern that listDirectoryContents() expands the wildcards in a pattern. If the expansion results in a file, it returns a file. If the expansion results in a directory, it returns the directory non-recursively. Wildcard expansion can result in any combination of files and directories.

LogServerComponentType class

A class that describes a LogServerComponent within an application.

The LogServer component controls the use of the Endeca Log Server.

LogServerComponentType properties

- componentID (required) identifies the component to use.
- hostID (required) is a unique string identifier for this host.
- workingDir is a string identifying the working directory for this component. Any relative paths in component properties are be interpreted as relative to the component's workingDir. The workingDir property, if specified, must be an absolute path.
- logFile is a string identifying the log file for this component.
- port (required) is the port on which to run the LogServer.
- outputPrefix (required) is the path and prefix name for the LogServer output.
- gzip (required) controls the archiving of log files. Possible values are true and false.
- startupTimeout (required) specifies the amount of time in seconds that the Application Controller will wait while starting the LogServer. Note that your stub generation tool may generate a Boolean property (for example, startupTimeoutSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

PropertyListType class

A class that represents a collection of PropertyType objects.

PropertyListType property

- properties is a collection of name/value pairs.

PropertyType class

The PropertyType class allows you to add arbitrary properties (that is, name/value pairs) to host and all component elements.

PropertyType properties

- name (required) is a non-null string.
- value is a string.

ProvisioningFault class

An extension of EACFault, the ProvisioningFault class is thrown when there are fatal errors during provisioning.

ProvisioningFault properties

- errors is a list of provisioning errors.
- warnings is a list of provisioning warnings.

RemoveApplicationType class

Related to the Provisioning service, this class serves as input to the incremental remove command.

RemoveApplicationType properties

- applicationID (required) identifies the application to use.
- forceRemove indicates whether or not a remove operation should force any running components or services to stop before attempting the remove. Note that your stub generation tool may generate a Boolean property (for example, forceRemoveSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

RemoveComponentType class

Related to the Provisioning service, this class serves as input to the incremental remove command.

RemoveComponentType properties

- FullyQualifiedComponentIDType (required) identifies the component to use.
- forceRemove indicates whether or not a remove operation should force the component to stop before attempting the remove. Note that your stub generation tool may generate a Boolean property (for example, forceRemoveSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

RemoveHostType class

Related to the Provisioning service, this class serves as input to the incremental remove command.

RemoveHostType properties

- FullyQualifiedHostIDType (required) is a unique string identifier for this host.
- forceRemove is a Boolean that indicates whether or not a remove operation should force any running components or services to stop before attempting the remove. Note that your stub generation tool may generate a Boolean property (for example, forceRemoveSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

RemoveScriptType class

Related to the Provisioning service, this class serves as input to the incremental remove command.

RemoveScriptType properties

- applicationID (required) identifies the application.
- scriptID (required) identifies the script to remove.
- forceRemove is a Boolean that indicates whether or not a remove operation should force any running components or services to stop before attempting the remove. Note that your stub generation tool may generate a Boolean property (for example, forceRemoveSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

ReportGeneratorComponentType class

A class that describes a ReportGenerator component within an application.

The ReportGenerator component runs the Report Generator, which processes Log Server files into HTML-based reports that you can view in your Web browser and XML reports that you can view in Endeca Workbench.

ReportGeneratorComponentType properties

- componentID (required) identifies the component to use.
- hostID (required) is a unique string identifier for this host.
- workingDir is a string identifying the working directory for this component. Any relative paths in component properties are be interpreted as relative to the component's workingDir. The workingDir property, if specified, must be an absolute path.
- logFile is a string identifying the log file for this component. args is a list of command-line flags to pass to the ReportGenerator.
- javaBinary, if used, should indicate a JDK 1.5.x or later. Defaults to the JDK that Endeca installs.
- javaOptions are the command-line options for the javaBinary parameter. This parameter is primarily used to adjust the ReportGenerator memory, which defaults to 1GB. To set the memory, use the following: java_options = -Xmx[MemoryInMb]m -Xms[MemoryInMb]m inputDirOrFile (required) is the path to the file or directory containing the logs to report on. If it is a directory, then all log files in that directory are read. If it is a file, then just that file is read.

- outputFile (required) is the name the generated report file and path to where it is stored.
- stylesheetFile (required) is the filename and path of the XSL stylesheet used to format the generated report.
- settingsFile is the path to the report_settings.xml file.
- timerange sets the time span of interest (or report window). Allowed keywords: Yesterday, LastWeek, LastMonth, DaySoFar,WeekSoFar, and MonthSoFar. These keywords assume that days end at midnight, and weeks end on the midnight between Saturday and Sunday. Note that your stub generation tool may generate a Boolean property (for example, timerangeSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.
- startDate set the report window to the given date and time. The date format should be either yyyy_mm_dd or yyyy_mm_dd.hh_mm_ss.
- stopDate sets the report window to the given date and time. The date format should be either yyyy_mm_dd or yyyy_mm_dd.hh_mm_ss. timeSeries turns on the generation of time-series data and specifies the frequency, Hourly or Daily.
- charts turns on the generation of report charts. Note that your stub generation tool may generate a Boolean property (for example, chartsSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

RunBackupType class

A child of the RunUtilityType class, this class provides all the information you need to perform a backup operation to the Archive utility.

RunBackupType properties

- applicationID (required) is the unique identifier for this application.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- hostID (required) is a unique identifier for the host. The hostID and dirName parameters specify the path to the directory that will be archived.
- dirName (required) is the full path of the directory. The hostID and dirName parameters specify the path to the directory that will be archived.
- backupMethod is either Copy or Move. Note that your stub generation tool may generate a Boolean property (for example, backupMethodSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.
- numBackups specifies the maximum number of archives to store. This number does not include the original directory itself, so if numBackups is set to 3, you would have the original directory plus up to three archive directories, for a total of as many as four directories. The default numBackups is 5. Note that your stub generation tool may generate a Boolean property (for example, numBackupsSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

RunFileCopyType class

A child of the RunUtilityType class, this class provides all the information you need to run the Copy utility.

RunFileCopyType properties

- applicationID (required) identifies the application to use.

- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- fromHostID (required) is the unique identifier for the host you are copying the data from. toHostID (required) is the unique identifier for the host you are copying the data to.
- sourcePath (required) is the full path to the source file or directory. If sourcePath contains no wildcards, then destinationPath must be the destination file or directory itself, rather than the parent directory.
- destinationPath (required) is the full path to the destination file or directory.
- recursive, when specified, downloads the directories recursively. Note that your stub generation tool may generate a Boolean property (for example, recursiveSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

RunRollbackType class

A child of the RunUtilityType class, this class provides all the information you need to perform a rollback operation to the Archive utility.

RunRollbackType properties

- applicationID (required) identifies the application to use.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- hostID (required) is a unique identifier for the host. The hostID and dirName parameters specify the path to the directory that will be archived.
- dirName (required) is the full path for the directory. The hostID and dirName parameters specify the path to the directory that will be archived.

RunShellType class

A child of the RunUtilityType class, this class provides all the information you need to run the Shell utility.

RunShellType properties

- applicationID (required) identifies the application to use.
- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.
- hostID (required) is a unique identifier for the host.
- cmd (required) is the command(s). workingDir is the full path for the working directory.

RunUtilityType class

Parent class of the other Utility classes.

RunUtilityType properties

- applicationID (required) identifies the application to use.

- token identifies the token used to stop the utility or to get its status. If you do not specify a token, one is generated and returned when you start the utility.

ScriptListType class

A class that describes a list of scripts.

ScriptListType properties

- script (required) is a collection of scripts comprising this ScriptListType object.

ScriptType class

A class that describes the base type for all scripts within an application.

ScriptType properties

- scriptID (required) is a unique string identifier for the script.
- cmd (required) is the command that is used to start the script.
- logFile is the file for appended stdout/stderr output. It defaults to $ENDECA_CONF/logs/script/(app_id).(script_id).log.
- workingDir is the working directory. It defaults to $ENDECA_CONF/working/(app_id)/.

SSLConfigurationType class

A class used by the DgraphComponentType class to enable SSL on the resulting components.

SSLConfigurationType properties

- certFile (required) specifies the path of the eneCert.pem certificate file that is used by the Dgraph process to present to any client.

  The file name can be a path relative to the component's working directory.

- caFile (required) specifies the path of the eneCA.pem Certificate Authority file that the Dgraph process uses to authenticate communications with other Endeca components. The file name can be a path relative to the component's working directory.

- cipher specifies a cryptographic algorithm that Dgraph will use during the SSL negotiation. If you omit this setting, the Dgraph chooses a cryptographic algorithm from its internal list of algorithms. See the *Endeca Commerce Security Guide* for more information.

StateType class

A class used by the StatusType class to describe the state of a component.

StateType fields

An enumeration of the following fields:

- Starting Starting only applies to server components (Dgraph or LogServer).
- Running
- NotRunning
- Failed

StatusType class

Describes the status of a server component in the Application Controller.

Server components include the Dgraph and LogServer. All other components (Forge, Dgidx, and ReportGenerator) are batch components. Their status is described by the BatchStatusType class.

StatusType properties

- StateType – (required) An enumeration of the following fields: Starting (which only applies to server components (Dgraph or LogServer), Running, NotRunning, or Failed.
- startTime – (required) The time the component started; for example, 10/25/07 3:58 PM.
- failureMessage – The failure message, which tells you that a failure has occurred in the execution of the component. failureMessage is empty unless state is FAILED. (This is different from EACFault, which tells you that a problem has occurred while processing the Web Service request to get the status.)

TimeRangeType class

A class used by the ReportGeneratorComponentType class to set the time span of interest (or report window).

TimeRangeType fields

The enumeration of possible values is as follows:

- Yesterday
- LastWeek
- LastMonth
- DaySoFar
- WeekSoFar
- MonthSoFar

TimeSeriesType class

A class used by the ReportGeneratorComponentType class to turn on the generation of time-series data and specify the frequency, hourly or daily.

TimeSeriesType fields

The enumeration of possible values is as follows:

- Hourly
- Daily

UpdateComponentType class

A class that describes a component to be updated during incremental provisioning.

UpdateComponentType properties

- applicationID (required) identifies the application.
- component (required) identifies the component to update.
- forceUpdate indicates whether or not the Application Controller should force the component to stop before attempting the update. Note that your stub generation tool may generate a Boolean property (for example, forceUpdateSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

UpdateHostType class

A class that describes a host to be updated during incremental provisioning.

UpdateHostType properties

- applicationID (required) identifies the application.
- host (required) identifies the host to update.
- forceUpdate indicates whether the Application Controller should force any components or services running on the host to stop before attempting the update. Note that your stub generation tool may generate a Boolean property (for example, forceUpdateSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

UpdateScriptType class

A class that describes a script to be updated during incremental provisioning.

UpdateScriptType properties

- applicationID (required) identifies the application.
- scriptID (required) identifies the script to update.
- forceUpdate indicates whether the Application Controller should force any components or services running on the host to stop before attempting the update. Note that your stub generation tool may generate a Boolean property (for example, forceUpdateSpecified in .NET) that is used to detect whether the user called the set method for this attribute; the property will be used to determine whether to include this field in the serialized XML.

Index

A

B

C

D

E

F

G

H

I

L

M

O

P

R

S

T

U

V

W

X