Oracle Endeca Commerce

Content Acquisition System Developer's Guide

Version 11.0 • January 2014

Contents

Preface....9
About this guide....9
Who should use this guide....9
Conventions used in this guide....9
Contacting Oracle Support....10

Part I: Introduction to CAS and Crawling Data Sources....11

Chapter 1: Introduction....13
Overview of the Endeca Content Acquisition System....13
About the Endeca CAS Service....15
About the CAS Server....15
About the Component Instance Manager....16
About the Record Store....16
About the Dimension Value Id Manager....17
Overview of the default CAS data sources and manipulators....18
Security requirements....19

Chapter 2: Creating a crawl....21
About creating a crawl....21
About filters....21
About CAS output types and the Deployment Template....23
Creating a crawl using the CAS Server Command-line Utility....23
Creating a crawl using CAS Console....24
Creating a crawl using the CAS Server API....24
Setting document conversion options....24
Configuring document conversion filters....25
Modifying a crawl using the CAS Server Command-line Utility....27

Chapter 3: Configuring a Record Store instance....29
About record generations....29
About transactions....29
About the last read generation for a client....30
About deleted records....31
Configuring a Record Store instance....32
Configuration properties for a Record Store instance....33
Change properties and new Record Store instances....37
Deleting stale generations of records....37
Disabling automatic management of a Record Store instance....37
Performance considerations when using a Record Store instance....38

Chapter 4: Running a crawl....41
Running a crawl....41
Order of execution in a crawl configuration....41
Full and incremental crawling modes....42
Crawls and archive files....43
About writing records to a Record Store instance....45
About the record output file....46

Chapter 5: Running the CAS sample applications....49
About the sample CAS applications....49

Part II: Using CAS data sources....65

Chapter 6: Using the Delimited File data source.....67
Configuration properties for the Delimited File data source.....67

Chapter 7: Using the Endeca Record File data source.....69
Configuration properties for the Endeca Record File data source.....69

Chapter 8: Using the File System data source.....71
Configuration properties for the File System data source.....71

Chapter 9: Using the JDBC data source.....73
Installing a JDBC driver into CAS.....73
Configuration properties for the JDBC data source.....73
Feature notes and known limitations of the JDBC data source.....74

Part III: Loading data into an MDEX Engine.....77

Chapter 10: Creating a Forge pipeline to read from or write to a Record Store.79
Overview of a Forge pipeline.....79
Creating a Forge pipeline 80

Chapter 11: Creating a CAS crawl to write MDEX-compatible output.....85
Overview of a CAS crawl that produces MDEX-compatible output.....85
Loading dimensions, properties, and precedence rules.....86
Loading dimension value records into a Record Store instance.....87
Loading data records into a Record Store instance.....91
About configuring application features in a CAS-based application.....92
Creating a crawl to write MDEX-compatible output.....93

Part IV: CAS Command Line Utilities.....95

Chapter 12: CAS Server Command-line Utility.....97
Overview of the CAS Server Command-line Utility.....97
About CAS capabilities.....98
Saving passwords in a crawl configuration file.....99
Inspecting installed modules.....100
Managing crawls.....103
Managing dimension value Ids.....112
Viewing crawl status and results.....118

Chapter 13: Component Instance Manager Command-line Utility.....123
Overview of the CIM Command-line Utility.....123
Creating a Record Store.....124
Deleting a Record Store.....125
Listing components.....126
Listing types.....126

Chapter 14: Record Store Command-line Utility.....129
Overview of the Record Store Command-line Utility.....129
Writing tasks.....131
Reading tasks.....132
Utility tasks.....135

Part V: Administering CAS.....147

Chapter 15: Running CAS components.....149

About running CAS components....149
Running the Endeca CAS Service from the Windows Services console....150
Starting the Endeca CAS Service from a command prompt....150
Stopping the Endeca CAS Service from a command prompt....152

Chapter 16: Backing up and restoring CAS....153
Coordinating backups and restore operations....153
Online backup and restore operations....153
Offline backup and restore operations....156

Chapter 17: Configuring logging....159
Configuring logging for CAS components and command-line utilities....159
Setting log properties for troubleshooting CMS data source issues....160
Excluding failed records from the CAS Service log file....160
Enabling log timing information for crawl processing steps....161
Examining the Endeca CAS Service log....161

Chapter 18: Tips and troubleshooting CAS....165
Fixing crawl performance issues....165
Modifying the CAS Server connection information for the CAS Console....165
Modifying the CAS Service temporary directory....166
Responding to a "Too many open files" error....166
Setting the group entry size....166

Appendix A: Sample crawl configuration files....169
Common properties for crawl configurations....169
Sample configuration for a file system data source....170
Sample configuration for a Record Store Merger data source....171
Sample configuration for a manipulator....172
Sample configuration for writing output to a Record Store instance....174
Sample configuration for writing output to an MDEX compatible format....176
Sample configuration for writing output to a file....178

Appendix B: File Formats Supported by the CAS Document Conversion Module.181
Archive formats....181
Database formats....181
E-mail formats....182
Multimedia formats....183
Other formats....184
Presentation formats....185
Raster image formats....185
Spreadsheet formats....187
Text and markup formats....188
Vector image formats....188
Word processing formats....190

Appendix C: Record properties generated by crawling....193
Common record properties....193
Record properties generated by file system crawls....194
Common File System properties....196
Record properties for file system crawls on Windows....197
Record properties for file system crawls on UNIX....197
Limitations with ACL properties....197
Document Conversion properties....198
Record properties generated by CMS crawls....198
How CMS crawls handle multiple pieces of content....200

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

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

About this guide

This guide describes how to configure and run CAS to incorporate source data gathered from file systems, CMS data sources, and custom data sources. The guide also explains how to create both Forge pipelines and CAS pipelines that process the data for use in an MDEX Engine.

The guide assumes that you are familiar with Endeca concepts and Endeca application development.

Who should use this guide

This guide is intended for application developers who are using CAS to crawl data sources, manipulate the records if necessary, and incorporate the records into either a Forge pipeline or a CAS crawl.

Conventions used in this guide

This guide uses the following typographical conventions:

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

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

Contacting Oracle Support

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

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

Part 1

Introduction to CAS and Crawling Data Sources

- *Introduction*
- *Creating a crawl*
- *Configuring a Record Store instance*
- *Running a crawl*
- *Running the CAS sample applications*

Chapter 1

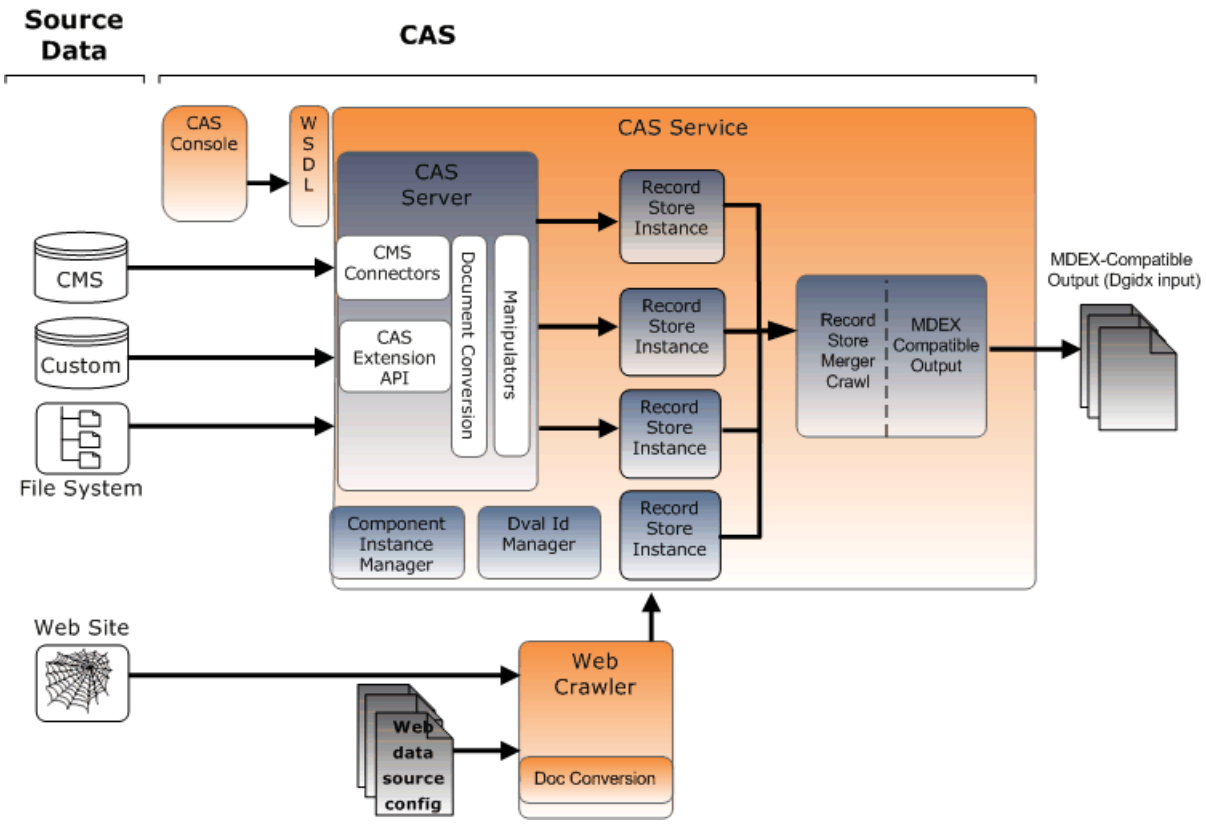
Introduction

This section provides introductory information about the Endeca Content Acquisition System (CAS).

Overview of the Endeca Content Acquisition System

The Endeca Content Acquisition System is a set of components that add, configure, and crawl data sources for use in an Endeca application. Data sources include file systems, content management systems, Web servers, and custom data sources. The Endeca Content Acquisition System crawls data sources, converts documents and files to Endeca records, and stores them for use in an Forge pipeline.

The following image shows the Endeca Content Acquisition System components as they work together in a typical implementation to crawl data sources and produce Endeca records:



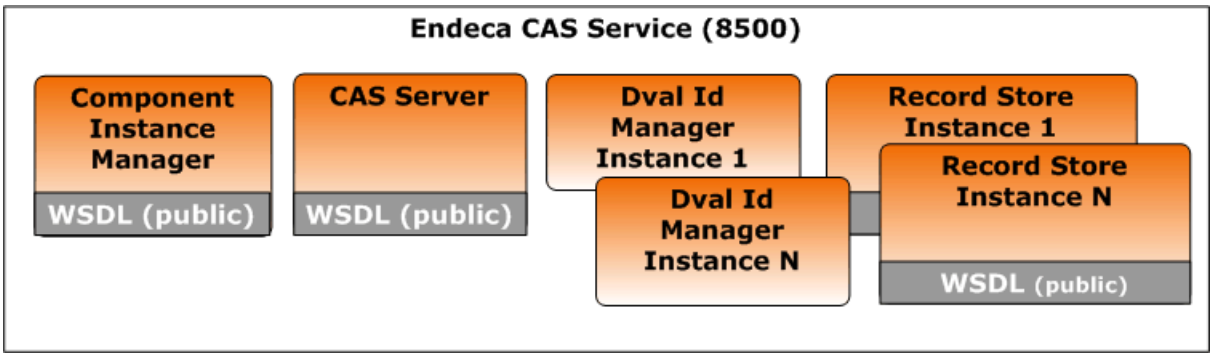
The Endeca Content Acquisition System is made up of the following components:

- The Endeca CAS Service is a servlet container that runs the CAS Server, the Component Instance Manager, and any number of Record Store instances (one per data source).
- The CAS Server is the component that manages all file system and CMS crawling operations. The CAS Server is documented in the *CAS Developer's Guide*.
- The CAS Console for Endeca Workbench is a Web-based application used to crawl various data sources including file systems and content management systems. During the Content Acquisition System installation, the CAS Console is installed as an extension to Endeca Workbench. The CAS Console is documented in the *Endeca CAS Console Help*.
- The CAS Server API allows users to write programs that communicate with the CAS Server. The CAS Server API has a WSDL interface and also a CAS Server Command-line Utility. The API is documented in the *Endeca CAS API Guide*.
- The Dimension Value Id Manager is a CAS component that creates, stores, and retrieves dimension value identifiers.
- The Endeca Web Crawler manages all Web crawl-related operations. This component is documented in the *Endeca Web Crawler Guide*.
- Endeca CMS connectors are available for use in the CAS Console for Endeca Workbench or the CAS Server API. CMS connectors provide a means to access and crawl data sources in a variety of CMS types.
- The Component Instance Manager creates, lists, and deletes Record Store instances. The Component Instance Manager has a WSDL interface and also a CIM Command-line Utility.
- The Endeca Record Store provides persistent storage for generations of records. The Record Store has a WSDL interface and also a Record Store Command-line Utility. The CAS Server writes crawl output from each data source to a unique Record Store instance.
- The CAS Extension API provides interfaces and classes to build extensions such as custom data sources and custom manipulators. You package extensions into a plug-in and install it into the Content Acquisition

System. After you install the plug-in, the extensions are available and configurable using the CAS Console, the CAS Server API, and the CAS Server Command-line Utility.

About the Endeca CAS Service

The Endeca CAS Service is a servlet container that runs the CAS Server, the Component Instance Manager, any number of Dimension Value Id Managers (one per application), and any number of Record Store instances (one per data source).



On Windows, the CAS installation program starts the service automatically and the service is set to restart automatically during system restarts. If you accept the installation defaults, the service runs on port 8500. The following image shows the components running within the Endeca CAS Service:

In the Windows Services console, the service displays as Endeca CAS Service. The service is running `cas-service-wrapper.exe` in `<install path>\CAS\<version>\bin`.

On UNIX, you can start Endeca CAS Service using `cas-service.sh` located in `<install path>/CAS/<version>/bin` and stop it using `cas-service-shutdown.sh`. Or if you set up the service in `inittab`.

About the CAS Server

The CAS Server manages all crawl operations of file system, CMS, and custom data sources. The CAS Server has a WSDL interface and a CAS Server Command Line Utility. The CAS Server runs inside the CAS Service.

The CAS Server has the following characteristics:

- Includes with the CAS Document Conversion Module, which allows the CAS Server to convert binary files (such as Microsoft Word documents and Adobe Acrobat PDF files) into text.
- Uses include and exclude filters to specify which files or folders to retrieve or avoid.
- Configures the logging behavior for a crawl, including setting the log level for various components and specifying output to the console, to a log file, or to both.
- Supports incremental crawls in which the CAS Server processes only the content which has been added, modified, or deleted since the last crawl.
- Enables security by supporting the Endeca Access Control System. Each crawl generates access control list (ACL) properties for each record, based on the corresponding properties for each file (for file system crawls) or entry in the CMS repository (for CMS crawls).

The CAS Server also tags the records with metadata properties that are derived from the source documents. After the CAS Server returns the records, you can configure an Endeca record adapter (from Developer Studio) to read the records into your Endeca implementation's pipeline, where Forge processes the records and can add or modify the record properties. These property values can then be mapped to Endeca dimensions or properties by the property mapper in the pipeline. You can then build an Endeca application to access the records and allow your application users to search and navigate the document contents contained in the records.

You can configure the CAS Server to crawl a data source and generate access control list (ACL) properties for each record. These ACL properties can be used in conjunction with security login modules to limit access to records based on user login profiles. For details on using the Endeca Access Control System, see the *Endeca Platform Services Security Guide*.

About the Component Instance Manager

The Component Instance Manager creates, lists, and deletes Record Store instances and Dimension Value Id Manager instances. The Component Instance Manager has a WSDL interface and also a CIM Command-line Utility.

The Component Instance Manager runs inside the Endeca CAS Service.

About the Record Store

The Endeca Record Store is a Web service that provides persistent storage for generations of records that can later be accessed by Forge for baseline and partial updates or by CAS for acquisition processing. The Endeca Record Store is integrated with the Endeca CAS Server to directly store output in the Record Store instead of sending output to files.

The Record Store has the following features.

Provides an efficient repository for records

Instead of having your source records residing in different directories, they can be consolidated in one place. This consolidation eliminates the need to copy and move source files among different directories.

Retrieves baseline and incremental records

You can store records from both Forge baseline and incremental crawls in the Record Store. You can then run a Forge baseline using all the stored records.

Operates asynchronously

An application (such as the CAS Server) can write records into a Record Store while, at the same time, Forge can read in records to process. Each process runs isolated from changes that the other is currently making.

Operates as a lookup table

Forge can process a record of one type (say, Order records) and then look up a record of another type (say, Product records) for a join.

Creates a separate Record Store instance for each data source

The Record Store Web service ceates a unique Record Store instance for each data source that CAS Server crawls. In general, there is a one-to-one mapping from a data source (such as a file system or a CMS) to a corresponding Record Store instance. A separate Record Store instance for each data source keeps records schemas separate. CAS Console for Endeca Workbench enforces this one-to-one mapping by creating a new Record Store instance for each data source you add. This mapping is not enforced in cases where you explicitly disable automanagement using the `isManaged` property.

Automatically cleans stale records

The Record Store service periodically removes stale generations of records. The time interval to remove stale generations is configurable and the feature can be disabled if necessary.

Easily configured and managed with a Record Store Command-line utility

CAS includes a Record Store Command-line utility to perform Record Store configuration and management. You can use this utility to run get/set commands to update a Record Store instance with configuration settings.

You create a Record Store instance using either the Component Instance Manager Command-line Utility, the CAS Console, or the CAS Server API.

About the Dimension Value Id Manager

The Dimension Value Id Manager is a CAS component that creates, stores, and retrieves dimension value identifiers. Dimension value Ids are used by Endeca components, such as Endeca Workbench with Rule Manager, Experience Manager, and also by the Endeca APIs as part of URLs in an application's navigation state.

There is a command line interface (`cas-cmd`) to the component to manually perform the following operations:

- Create a Dimension Value Id Manager.
- Generate dimension value Ids.
- Export and import dimension value Ids.
- Get dimension value Ids.
- Delete a Dimension Value Id Manager.

Using the Dimension Value Id Manager and the Discover Electronics sample application

Most of the `cas-cmd` operations for the Dimension Value Id Manager are automatic and transparent if you are running the Discover Electronics sample application and the Deployment Template. The `initialize_services` script creates a new instance of a Dimension Value Id Manager. CAS generates dimension value Ids as part of writing MDEX output. You manually delete the Dimension Value Id Manager using the `cas-cmd` utility before removing an Endeca application.

Using the Dimension Value Id Manager with other applications

With other applications controlled by the Deployment Template, you manually create a Dimension Value Id Manager using the `cas-cmd` utility.

However, the remaining steps are the same as the Discover Electronics sample application. CAS generates dimension value Ids as part of writing MDEX-compatible output. You manually delete the Dimension Value Id Manager using the `cas-cmd` utility before removing an Endeca application.

Backing up and restoring dimension value Ids

You should regularly back up dimension value Id mappings by running the `exportDimensionValueIdMap¬pings` task of `cas-cmd`. A backup file ensures you have a snapshot of the Ids used in an application's query URLs, and you can restore those mappings if the disk or machine that was storing that state fails.

As necessary, you can restore the mappings by running the `importDimensionValueIdMappings` task of `cas-cmd`.

Propagating dimension value Ids across environments

In many cases, you have to move a dimension value Id mapping file between environments (i.e. staging, production, fail over, and so on). You can coordinate this work in your Deployment Template script by calling `exportDimensionValueIdMappings()` on the `ContentAcquisitionServerComponent`, copying the file to the necessary machine, and calling `importDimensionValueIdMappings()` to load the file into another instance of a Dimension Value Id Manager.

Lifecycle considerations

There should be one instance of a Dimension Value Id Manager per Endeca application, and that instance should exist for the lifetime of an Endeca application. You manually delete a Dimension Value Id Manager by running the `deleteDimensionValueIdManager` task of `cas-cmd` before deleting the Endeca application itself. The is necessary when you are running the Discover Electronics sample application or any other application.

Overview of the default CAS data sources and manipulators

The Content Acquisition System ships with a set of default data sources and manipulators. Each is described here:

Data Source	Description
Delimited File	Crawls records in delimited text files, including .csv files.
Endeca Record File	Crawls Endeca record files including .xml, .xml.gz, .bin, .bin.gz, .binary, and .binary.gz.
File System	Crawls folders and files on both local drives and network drives.
JDBC	Crawls a JDBC-accessible database.
Record Store Merger	Crawls CAS record store instances.

For information about version support for a particular repository, see the data source's chapter in *CAS Developer's Guide*.

Manipulator	Description
Filtering Script	This manipulator runs an inline BeanShell script that filters Endeca records from crawl output.
Modifying Script	This manipulator runs an inline BeanShell script that modifies Endeca records.

For information about configuring a data source or a manipulator, see the *CAS Console Help* or run the `cas-cmd` utility with the `getModuleSpec` task to return configuration properties.

Security requirements

The CAS Server supports the Endeca Access Control System which you can use to make your front-end Endeca application secure. This topic explains the details involved in making your Endeca application secure based on the ACL properties generated from your CMS repository.

To make use of the ACL properties generated by the CAS Server in your Endeca front-end application, take into account the following considerations:

- The CAS Server tags each record with access control list (ACL) properties that it generates. The generated ACL properties are based on the corresponding properties for each entry in the CMS repository. In other words, the ACL properties generated by a crawl are based on ACL properties created by your CMS repository.

  You can use manipulators to transform the generated ACL properties into the format for ACLs that is used by the Endeca Access Control System. You can then use the modified properties in conjunction with security login modules to limit access to records based on user or group login profiles. For details on using the Endeca Access Control System, see the *Endeca Security Guide*.

- Typically, the generated ACL properties, since they are based on the ACL information specific to a repository, can apply to either users or groups. If they apply to groups, the code for the Endeca front-end application has to map users to their corresponding groups.

Chapter 2

Creating a crawl

This section provides information about creating, crawls, and specifying options for different types of crawls.

About creating a crawl

You can use CAS Console, the CAS Server Command-line Utility, and CAS Server API to create and configure any number of crawls in your application. If you use CAS Console, note that a *crawl* is equivalent to a *data source* in the CAS Console user interface.

For each crawl, you specify configuration options such as:

- The name of the configured crawl.
- The location of the source data for crawl. For example, this could be a seed for a file system crawl, or a CMS repository for a CMS crawl, or a database for a JDBC crawl.
- Filters that include or exclude designated files and folders.
- Repository properties for CMS data sources.
- Manipulators to modify Endeca records as part of the crawl.

About filters

Filters define which folders and files are included and excluded when the CAS Server crawls a data source. You specify filters in the CAS Console or in the CAS Server API.

Filters perform matching operations against a property on an Endeca record and either include or exclude the record based on the filter's evaluation. You specify both the Endeca property to evaluate and the data type and expression to match against that property.

If an include filter matches (evaluates to true) against a property, then that record is included in the record set. If an exclude filter matches (evaluates to true) against a property, then that record is excluded from the record set.

Filters perform matching based on the following data types:

- Date - a date value against which files and folders can be filtered.
- Long - a long value to compare against a numerical property.
- String - a string value to compare against a string property. String filters are either regex or wildcard.

Regex - a regular expression value to compare against a string property. The matching evaluation is one of equality: the string either matches the expression or it does not match.

Wildcard - a wildcard expression value to compare against a string property. The wildcard matcher uses the question-mark (?) character to represent a single wildcard character and the asterisk (*) to represent multiple wildcard characters. The matching evaluation is one of equality: the string either matches the expression or it does not match. Also, there must be either all include filters or all exclude filters per property.

The following sub-headings define the way filters operate:

One filter per Endeca property per file or folder (unless a wildcard)

You can create one filter per Endeca property that applies to a file and one filter per Endeca property that applies to a folder, unless you are creating a wildcard filter. (You can create any number of wildcard filters for an Endeca property.) This also means you can create a file filter and a folder filter that apply to the same Endeca property.

AND'ing and OR'ing

- If you create multiple filters on a single property (wildcard filters), they are logically OR'ed during filter evaluation.
- Filters across properties are logically AND'ed during filter evaluation. Remember that AND means that all filters must match in order for a record to be included or excluded.

Include and exclude filters

- Include filters may apply to either folders or files.
- Exclude filters apply to only to folders.

Filter precedence

If you use both include and exclude filters, exclude filters take precedence. For example, if a `test.doc` file was recently modified and you add an include filter for `test.doc` but then add an exclude filter that excludes all recently modified files, the `test.doc` will not be crawled.

Missing properties on a record

Filters require an Endeca property to match against. In cases where the property for a filter does not exist on a record, the behavior varies depending on whether the filter is an include or an exclude.

- If the filter is an include and the property does not exist on a record, the record is excluded.
- If the filter is an exclude and the property does not exist on a record, the record is included.

Unfilterable properties

Do not use the `Endeca.Document` properties for filter matching. These properties are generated by the CAS Document Conversion Module *after* a file or folder is crawled and filtered, and therefore cannot be used to filter files or folders.

Case sensitivity

Regex filters are case sensitive by default (however, you may construct a regular expression that is case insensitive). Wildcard filters are case insensitive.

About CAS output types and the Deployment Template

The Content Acquisition System can write output from a crawl to MDEX-compatible output, a Record Store instance (the default), or to a record output file.

Characteristics of a MDEX-compatible output

There are several advantages to configuring a CAS crawl to write MDEX-compatible output:

- There are APIs (such as the Record Store API and the Endeca Configuration Import API) that allow programmatic configuration of data records, dimension value records, and schema configuration that are all incorporated into MDEX-compatible output.
- The output includes data-driven dimension values.
- Forge is not required for update processing (baseline or partial) unless the update requires advanced join operations.

Characteristics of a Record Store instance versus record output files

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

Note: Although all storage types are fully supported, Oracle does not recommend writing to a record output file because the operational model and Deployment Template configuration are more complicated than a model that writes to a Record Store instance or MDEX-compatible output.

Creating a crawl using the CAS Server Command-line Utility

If you want to use the CAS Server Command-line Utility, you configure a crawl by creating a crawl configuration file and passing the file as an argument (`-f`) to the `createCrawls` task of the utility.

For new crawls, the CAS Server writes crawl output to a Record Store instance by default. This topic describes how to create a new crawl to write crawl output to any of the CAS output formats including:

- a Record Store instance (the default)
- an MDEX-compatible format (i.e. Dgidx input files)
- an output file (i.e XML or binary)

To create a crawl:

1. Manually create an XML crawl configuration file. The configuration settings in the file vary depending on the output formats you want. See the following samples:
   - *Sample configuration for writing output to a Record Store instance* on page 174
   - *Sample configuration for writing output to an MDEX compatible format* on page 176
   - *Sample configuration for writing output to a file* on page 178.
2. Save and close the XML crawl configuration file.
3. Start a command prompt, navigate to `<install path>\CAS\<version>\bin`, and locate the CAS Server Command-line utility (`cas-cmd`).
4. Run the `createCrawls` task of the `cas-cmd` and specify the `-f` option with an argument that specifies a path to the crawl configuration file you created.
   For example, in a default CAS installation on Windows, this command creates a new crawl configured by the file `crawlConfig.xml`.
   ```
   C:\Endeca\CAS\11.0.0\bin>cas-cmd createCrawls -f C:\tmp\crawlConfig.xml
   ```

Creating a crawl using CAS Console

If you want to use the CAS Console, you add a data source and configure the crawl on the **Data Source** tab, the **Filters** tab, and the **Advanced Settings** tab. The data source (crawl) may or may not have all of the tabs listed. You can also add one or more manipulators to modify Endeca records during the crawl.

The CAS Console includes an online help system, which provides details on the various configuration properties for each data source.

Creating a crawl using the CAS Server API

If you want to use the CAS Server API, you create a crawl using the `CasServer.createCrawl()` method and configure a crawl by setting configuration properties in the `SourceConfig`, `TextExtractionConfig`, `ManipulatorConfigs`, and `OutputConfig` objects (among others).

For details, see the *CAS API Guide* and the *CAS Server API Reference (Javadoc)*.

Setting document conversion options

You can change the behavior of the CAS Document Conversion Module for identifying fallback format, file identification, and extracting hidden text. You change the default document conversion behavior by specifying options via JVM property names and values. Note that you cannot set these options from the CAS Console.

The options are:

- `stellent.fallbackFormat` determines the fallback format, that is, what extraction format will be used if the CAS Document Conversion Module cannot identify the format of a file. The two valid settings are `ascii8` (unrecognized file types are treated as plain-text files, even if they are not plain-text) and `none` (unrecognized file types are considered to be unsupported types and therefore are not converted). Use the `none` setting if you are more concerned with preventing many binary and unrecognized files from being incorrectly identified as text. If there are documents that are not being properly extracted (especially text files containing multi-byte character encodings), it may be useful to try the `ascii8` option.

- `stellent.fileId` determines the file identification behavior. The two valid settings are `normal` (standard file identification behavior occurs) and `extended` (an extended test is run on all files that are not identified). The `extended` setting may result in slower crawls than with the `normal` setting, but it improves the accuracy of file identification.
- `stellent.extractHiddenText` indicates whether to convert hidden text stored in a content item. Hidden text may include text produced by optical character recognition (OCR) software in addition to other types of hidden text. Specifying `true` for `stellent.extractHiddenText` converts any hidden text stored in the content item. Specifying `false` does not convert hidden text.

Default values for the options

The default settings for the options are listed in the following table.

Option	Defaults
`stellent.fallbackFormat`	none
`stellent.fileId`	extended
`stellent.extractHiddenText`	If unspecified, the default value is `false`.

Setting the options

You set the text extraction options as parameters to the Java Virtual Machine (JVM), via the Java `-D` option. To set the fallback format, use one of these two parameters:

```
-Dstellent.fallbackFormat=ascii8
-Dstellent.fallbackFormat=none
```

To set the file identication behavior, use one of these two parameters:

```
-Dstellent.fileId=normal
-Dstellent.fileId=extended
```

To enable the extraction of hidden text, use this parameter:

```
-Dstellent.extractHiddenText=true
```

To pass these parameters to the JVM, use the `-JVM` flag when you run the Endeca CAS Service script or add JVM arguments to the script itself. Note that for Windows machines, the parameters should be quoted if they contain equal signs, as in this example:

```
cas-service -JVM "-Dstellent.fallbackFormat=ascii8"
```

Note that when using the `-JVM` flag, it must be the last flag on the command line.

Configuring document conversion filters

You can configure a set of filters to apply only to document conversion. These filters either convert or do not convert files of a specified size or type by using include or exclude filters. The include and exclude filters apply only to document conversion.

The document conversion filters perform matching against any Endeca property and include or exclude a file from the document conversion process but still produce an Endeca record for the file. If a file is included for document conversion, the corresponding Endeca record has an `Endeca.Document.Text` property.

Data source extensions built using the CAS Extension API do not support document conversion filters. Any changes you make to `DocumentConversionFilters.xml` are not applied to data source extensions. Also,

CAS does not apply document conversion filters to archive files. However, you can enable the **Expand archives** option and then CAS can process the extracted content.

You configure document conversion filters by modifying `<install path>\CAS\workspace\conf\DocumentConversionFilters.xml`. These document conversion filters apply to all data sources that have document conversion enabled; the filters do not apply on a per-data source basis.

This file has sections for CMS data sources and file system data sources. Within a CMS or file section, the file has a section for include filters and for exclude filters. Here is an example snippet of the structure of the file:

```
<cmsCrawlDocumentConversionFilters>
  <includeFilters>
    ...
  </includeFilters>
  <excludeFilters>
    ...
  </excludeFilters>
 </cmsCrawlDocumentConversionFilters>

<fileCrawlDocumentConversionFilters>
  <includeFilters>
    ...
  </includeFilters>
  <excludeFilters>
    ...
  </excludeFilters>
</fileCrawlDocumentConversionFilters>
```

Inside the `includeFilters` and `excludeFilters` sections are the filters themselves. Each is indicated by a `filter` element. For example, this snippet shows a regular expression filter for file system data sources that includes all files of the types listed:

```
<fileCrawlDocumentConversionFilters>
  <includeFilters>
   <filter xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="regexFilter">
    <scope>FILE</scope>
    <propertyName>Endeca.FileSystem.Name</propertyName>
    <regex>^(?i:.*\.(?:txt|html?|rtf|docx?|xlsx?|
      pptx?|pdf|sxi|sxc|sxw|shw|qpw|wpd|xml))$</regex>
   </filter>
...
  </includeFilters>
```

Each filter is made up of the following XML:

Element or attribute	Description
`type` attribute of `filter`	The value of `type` can be either `regexFilter`, `longFilter`, `dateFilter`, or `wildcardFilter`.
`scope`	The value of scope should always be set to `FILE`. Document conversion filters apply only to files; do not modify this value.
`propertyName`	The value of `propertyName` specifies the Endeca property on which you want the filter to perform matching operations. Common filter properties are `Endeca.FileSystem.Name`, and `Endeca.FileSystem.Extension`.

Element or attribute	Description
`operator`	The `operator` for `type="longFilter"` performs numeric comparisons using any of the following values: `EQUAL`, `GREATER`, `GREATER_EQUAL`, `LESS`, `LESS_EQUAL`, and `NOT_EQUAL`. The `operator` for `type="dateFilter"` performs comparisons against date time values using either `BEFORE` or `AFTER`.
`regexFilter`	Specifies a regular expression to compare against the specified property.
`longFilter`	Specifies a numeric value to compare against the specified property.
`dateFilter`	Specifies a date against which files can be filtered.
`wildcardFilter`	Specifies a wildcard to match against a specified property. The wildcard matcher uses the question-mark (?) character to represent a single wildcard character and the asterisk (*) to represent multiple wildcard characters. Matching is case insensitive: this is not configurable (If case sensitivity is required, consider using a regular expression).

Like other types of filters, document conversion filters cannot have multiple filters with the same `propertyName` unless the filters are `wildcardFilter`.

Note: Mime type properties depend on the data source and you may need to check that you add the correct mime type to your filters. Also, some CMS data sources may not produce an `Endeca.CMS.Con¬tentLength` property and therefore, you may not be able to filter those files by size.

To configure document conversion filters:

1. Navigate to `CAS\workspace\conf\`, and open `DocumentConversionFilters.xml` in a text editor.
2. Add include and and exclude filters according to the syntax described above.
   The `DocumentConversionFilters.xml` file contains default filters that you can use as examples. These filters are include filters for the most common document types such as txt, html, rtf, doc, pdf and so on.
3. Comment out any filters that you do not want applied.
4. Save and close the file.
5. Restart the Endeca CAS Service.
6. Optionally, perform a full crawl of the data source after configuring filters.

The file is validated against the schema and if there are validation errors, the Endeca CAS Service does not start. (This is logged in the CAS Service log file.) The file must conform to `DocumentConversionFilters.xsd`. If the file is missing, then CAS Server converts all documents by default.

Modifying a crawl using the CAS Server Command-line Utility

Modifying a crawl may be useful if you want to re-configure any output settings. The settings vary depending on the output type of the crawl.

For example, if you modify a crawl that writes to an output file, you could re-configure a different output directory, different file format, and so on. If you modify a crawl that writes to MDEX-compatible output, you could

re-configure the output directory, the Dimension Value Id Manager name, and so on. If you modify a crawl that writes to a Record Store instance, you could re-configure a different host machine, or use SSL, and so on.

To modify a crawl:

1. Start a command prompt, navigate to `<install path>\CAS\<version>\bin`, and locate the CAS Server Command-line utility (`cas-cmd`).
2. Run the `getCrawl` task of the `cas-cmd` and specify the `-f` option with an argument that specifies a path for the crawl configuration file and also specify the `-id` option with the ID of the crawl. Optionally, you may want to specify the `-d` option to write default values for the configuration properties.
   For example, in a default CAS installation on Windows, this command identifies a crawl named `itldocset` and gets its configuration and writes it to `C:\tmp\crawlConfig.xml`.

   ```
   C:\Endeca\CAS\11.0.0\bin>cas-cmd getCrawl -d -f C:\tmp\crawlConfig.xml -id
   itldocset
   ```

3. Modify the XML crawl configuration file as necessary. The configuration settings vary depending on the crawl output type. See the following samples:
   - *Sample configuration for writing output to a Record Store instance* on page 174
   - *Sample configuration for writing output to an MDEX compatible format* on page 176
   - *Sample configuration for writing output to a file* on page 178.
4. Save and close the XML crawl configuration file.
5. Run the `updateCrawls` task of the `cas-cmd` and specify the `-f` option with an argument that specifies the name for the XML crawl configuration file you modified in the previous steps.
   For example, in a default CAS installation on Windows, this command creates a data source named `itl¬docset`.`C:\Endeca\CAS\11.0.0\bin>cas-cmd updateCrawls -f C:\tmp\crawlConfig.xml`

If the task succeeds, the console displays a message similar to the following:

```
Updated crawl itldocset
```

Chapter 3

Configuring a Record Store instance

This section provides detailed information the CAS Record Store service and explains how to configure a Record Store instance.

About record generations

A set of records that has been committed to a Record Store instance is a record generation.

For example, if you perform a full file system crawl, all the records returned from the crawl are written to the Record Store and a commit is done. After the commit is done, the Record Store has one generation of records. A subsequent crawl, either full or incremental, results in a second generation of records.

Each record that is read in contains a unique ID. CAS uses that unique ID as the value of the `idPropertyName` Record Store configuration property.

If a record already exists with that unique ID during later CAS crawls, then the later version replaces the earlier one. This ensures that when you run an incremental crawl, you always get the latest version of any given record.

A record generation is removed from a Record Store instance by the `clean` task after the generation becomes stale. A stale generation is a generation that has been in a Record Store instance for a period of time that exceeds the value of the `generationRetentionTime` Record Store configuration property. I

A stale generation is retained in several exception cases:

- A generation is currently in use. This occurs because either a `READ` transaction or a `READ_WRITE` transaction is running.
- If there is only one generation in a Record Store instance, it is not removed, even if it is stale.
- A generation is marked as a last-read generation for a client.

About transactions

A transaction is an access operation in the Record Store by another component, such as the CAS Server API or Forge. Transactions provide a means to keep one operation isolated from another operation and allow each to operate independently.

In other words, one transaction can read while another is writing. Each transaction is either a `READ_WRITE` transaction or a `READ` transaction:

- READ transactions support only Read operations. A Read operation on uncommitted data is not supported. There may be any number of READ transactions running simultaneously. Examples of Read operations are Forge reading a record generation for a baseline update or an administrator using the Record Store Command-line utility with the -c flag to get the count (number of records) in a Record Store instance.
- READ_WRITE transactions support both Read and Write operations. There may be only one READ_WRITE transactions running at any time. An example of a Write operation is a crawler running a full crawl and writing the output to a Record Store instance. Just like in a READ transaction, a Read operation on uncommitted data in a READ_WRITE transaction is not supported.

Each transaction is assigned a transaction ID. When a transaction begins, the Record Store service logs an INFO message with the transaction type and ID, as in this example of Forge performing a READ transaction (with an ID of 2) for a baseline update:

```
Started transaction 2 of type READ
```

An example of a Write transaction message would be the following:

```
Started transaction 3 of type READ_WRITE
```

Each transaction has a status, which is one of the following:

- ACTIVE – The transaction is currently active. For example, the transaction is in the middle of a Write operation.
- COMMITTED – The transaction has successfully finished. An INFO message of “Committed transaction” is logged to indicate this status.
- COMMIT_FAILED – A commit transaction failed. The only operation allowed on the transaction is a rollback.
- ROLLED_BACK – The transaction has been successfully rolled back. No further operations are allowed on the transaction.

The rules for transactions are as follows:

- Once a transaction has been committed or rolled back, additional operations that try to access the transaction will fail.
- Once a Read operation has ended, additional operations that try to access the read cursor will fail.
- Only one operation per transaction can run at a time.
- If a transaction is rolled back, then it cancels operations in progress.

About the last read generation for a client

A Record Store instance can save the last read generation and also track that generation for any number of unique clients. You specify a unique client by creating a client ID (which can be any string, such as forge1) and then set a value to indicate the last-read generation for that client ID.

There are two ways to the last read generation for a client:

- Automatically, in a Forge pipeline, by using a Record Store adapter to read records from the Record Store. In the adapter, use the CLIENT_ID pass-through to specify the client ID to be set for the generation that is being read in.
- Manually, by using the set-last-read-generation task of the Record Store command-line utility.

Here is an example use-case with Forge processing the records:

1. You run a full crawl and it writes the records to a Record Store as Generation 1.
2. You perform a Forge baseline update using Generation 1. The Record Store adapter to Forge uses the READ_TYPE pass-through set to BASELINE and the CLIENT_ID pass-through set to forge1. The use of the CLIENT_ID pass-through means that a client state was saved for the forge1 client.

3. You run a second crawl, either full or incremental, and store the records as Generation 2. Because both crawls use the same `idPropertyName` and the same seeds, some of the records of both generations are identical and the others are delta records (new, modified, or deleted records).
4. You perform a Forge partial update using the delta records between Generation 1 and Generation 2. For this pipeline, the Record Store adapter to Forge uses the READ_TYPE pass-through set to `DELTA` and the CLIENT_ID pass-through set to `forge1`.
5. The delta records are processed by Forge and uploaded to the Endeca MDEX Engine.

To find out which client states are currently saved in a Record Store instance, use the `list-client-states` task of the Record Store command-line utility.

About deleted records

Any client of the Record Store, including the CAS Server, the Web Crawler, Forge, the CAS API, and so on, can modify and delete records that are written to the Record Store. Clients either update or insert a record, delete a record, or delete all records. This topic describes the `Endeca.Action` property that the Record Store examines to determine whether to update or insert or delete records.

Deleting all records for a full crawl

A record that has *only* the `Endeca.Action` property set to `DELETE` (i.e., has no other properties) functions as a logical “Delete All” record. When the Record Store encounters such a record, the Record Store removes all records from a Record Store instance. This is useful when running a full crawl and you want to remove a generation of records before writing a new generation.

For example:

```
<RECORDS>
  <RECORD>
    <PROP NAME="Endeca.Action">
      <PVAL>DELETE</PVAL>
    </PROP>
  </RECORD>
  ...
</RECORDS>
```

Deleting records for an incremental crawl

If a record has an `Endeca.Action` property set to `DELETE`, the record is removed from the Record Store instance. This property setting is useful in an incremental file crawl where files may have been modified or deleted since the last crawl.

If an incremental crawl does not find a file that is listed in the crawl history, the CAS Server treats that file as deleted. For each deleted file, a record is created that contains the location of the deleted file and an `Endeca.Action` property with a value of `DELETE`.

For renamed files, the file with the old name is treated as a deleted file while the file with the new name is treated as a new (added) file.

This example shows the record for a `TestPlan.doc` file that was deleted:

```
<RECORDS>
  <RECORD>
    <PROP NAME="Endeca.Action">
      <PVAL>DELETE</PVAL>
    </PROP>
    <PROP NAME="Endeca.FileSystem.Path">
```

```
      <PVAL>c:\endeca_test_docs\TestPlan.doc</PVAL>
    </PROP>
    <PROP NAME="Endeca.SourceType">
      <PVAL>FILESYSTEM</PVAL>
    </PROP>
    <PROP NAME="Endeca.SourceId">
      <PVAL>FileSystemSource</PVAL>
    </PROP>
  </RECORD>
  ...
</RECORDS>
```

In your pipeline, you should add a record manipulator to remove records that were marked for deletion.

Reading records marked with the DELETE property value

Any client of the Record Store, for example a custom record adapter in a Forge pipeline, can read from a Record Store instance and process records that are marked with the `Endeca.Action` property set to `DELETE`.

Configuring a Record Store instance

Each uniquely named Record Store instance has its own configuration settings. You can run the `get-configuration` task of the Record Store Command-line Utility to save the configuration settings to a file, or you can create the file manually.

You then modify the configuration properties in the file and then run the `set-configuration` task to apply the configuration changes to a particular Record Store instance. Changes to the properties take effect immediately.

Note: If you change the `btreePageSize`, `changePropertyNames`, `idPropertyName`, `jdbmSettings`, or `recordCompressionEnabled` properties, the Record Store deletes all stored data.

To configure a Record Store instance:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin`.
2. Type `recordstore-cmd` and specify the `get-configuration` task with the name of a Record Store instance and the XML file name where you want to save the configuration settings.
   This Windows example gets the configuration for a Record Store named `productdata`:
   ```
   recordstore-cmd.bat get-configuration -a productdata -f C:\tmp\config.xml -n
   ```
3. In a text editor, open the configuration file and modify the property values as described in *Configuration properties for a Record Store instance* on page 33.
4. Save and close the configuration file.
5. In the command prompt window, type `recordstore-cmd` and specify the `set-configuration` task with the name of a Record Store instance and the XML file name that contains the configuration settings.
   This example sets the configuration for a Record Store named `productdata`:
   ```
   recordstore-cmd.bat set-configuration -a productdata -f C:\tmp\config.xml
   ```

Configuration properties for a Record Store instance

The Record Store supports the configuration properties listed in the table below.

Configuration Property	Value
`btreePageSize`	Endeca Internal Use. Endeca does not recommend modifying this property. The number of children per btree node. This value must be greater than `0`. The default is 100. The Record Store validates that the `btreePageSize` property is greater than `0`. If this property is not correctly set, `RecordStore.setConfiguration()` throws a `RecordStoreConfigurationException`.
`changePropertyNames`	The `changePropertyNames` configuration property specifies which record properties to examine when determining whether a record has changed. The `changePropertyNames` configuration property is useful because it allows you to specify exactly which record properties are evaluated to determine if that record has changed between crawls (in other words, if the record is different from the previous generation's record in the Record Store instance). If not specified, the value of `changePropertyNames` defaults to all the properties on a record. If you choose to specify the value `changePropertyNames`, here are several suggested properties: • File system crawls: `Endeca.Document.Text`, `Endeca.FileSystem.ModificationDate` • CMS crawls: `Endeca.Document.Text`, `Endeca.CMS.ModificationDate` • Web crawls: `Endeca.Document.Text`, `Endeca.Web.Last-Modified` If you are gathering native file system properties for file system crawls, you can also use the ACL properties as change properties.
`cleanerInterval`	The `cleanerInterval` property specifies how often (in hours) the Record Store Service checks for stale generations of records in a Record Store. (A stale generation is defined by the `generationRetentionTime` property.) This value of `cleanerInterval` must be greater than or equal to `0`. If not specified, the value defaults to `1` hour. Fractional values (like `0.1`) can be specified if you want the service to check more frequently than once per hour. A value of `0` (zero) does not check for stale generations.

Configuration Property	Value
`dataDirectory`	The `dataDirectory` property specifies the location where the Record Store's data files are stored. The value of the `dataDirectory` property can either be an absolute path or a path relative to the directory where the Record Store is running. If not specified, the value defaults to `<install path>\CAS\<version>\workspace\state\<Record store name>`. This directory is created when a Record Store instance is created if it does not already exist. The Record Store service validates that the `dataDirectory` property specifies a directory for which the user running the Endeca CAS Service has write permissions.
`duplicateRecordCompressionEnabled`	The `duplicateRecordCompressionEnabled` property has been deprecated in 3.1.2, so this property is ignored. The `duplicateRecordCompressionEnabled` property specifies whether to store new versions of records whose *change properties* have not changed. Enabling this feature improves Record Store performance and decreases Record Store disk space. The `duplicateRecordCompressionEnabled` property takes a Boolean value: `true` does not store duplicate copies of records, and `false` stores duplicate copies of records. If not specified, the value defaults to `false`.
`generationRetentionTime`	The `generationRetentionTime` property specifies how long (in hours) a record generation should remain in a Record Store instance before it is considered a stale generation. The next time the `cleanerInterval` value is reached, then stale generations are deleted from the Record Store. However, the Record Store does not remove the most recent generation even if it exceeds the value of `generationRetentionTime`, and the Record Store does not remove the last generation specified with the `set-last-read-generation` task for a client ID. Fractional values (like `0.1`) can be specified if you want a generation to be maintained for less than an hour. If not specified, `generationRetentionTime` defaults to `48` hours (two days). The Record Store validates that the `generationRetentionTime` property is greater than or equal to `0`. If set to `0`, the Record Store instance only stores the latest (single) generation after the clean up.

Configuration Property	Value
idPropertyName	The idPropertyName property specifies the source property from which the record ID is derived. This value must be a non-empty string. If not specified, it defaults to Endeca.Id. When selecting a record property as the idPropertyName, you should choose a property that is present on every record and whose value is unique to each record. That is, all records (except those tagged with Delete All) must have a single unique non-null value for this property. The uniqueness of the property value is important because if two records have the same idPropertyName property value, the second record that is processed overwrites the first one in the Record Store. The Record Store validates that the idPropertyName property is a non-empty string. If this property is not correctly set, RecordStore.setConfiguration() throws a RecordStoreConfigurationException.
ignoreInvalidRecords	The ignoreInvalidRecords property specifies how invalid records are handled. Invalid records are records with missing IDs (either the idPropertyName property is missing or it has a null value) or with invalid action types for the Endeca.Action property. The ignoreInvalidRecords property takes a Boolean value: • If set to true, invalid records are ignored and a warning message is logged. The READ_WRITE operation for the records continues. • If set to false, an invalid record throws an exception and stops the process. In either case, invalid records are not added to the Record Store. If not specified, the value defaults to true. During the development stage of your Record Store application, you may want to set the ignoreInvalidRecords property to false so that an InvalidRecordFault exception is thrown whenever an invalid record is processed. This allows you to immediately see if your source records have the appropriate properties. Once you go into production, you can change the property to true and monitor the logs for warning messages about invalid records.
indexWriteFlushInterval	Oracle Internal Use. Oracle does not recommend modifying this property. This value must greater than 0. The Record Store validates that the indexWriteFlushInterval property is greater than 0. If this property is not correctly set, RecordStore.setConfiguration() throws a RecordStoreConfigurationException.

Configuration Property	Value
`jdbmSettings`	Oracle Internal Use. Oracle does not recommend modifying this property.
`maxDataFileSize`	The property `maxDataFileSize` has been removed in version 3.1.2 and later. Oracle Internal Use. Oracle does not recommend modifying this property. This value must be greater than `0`. The default value is 2 GB. The Record Store validates that the `maxDataFileSize` property is greater than `0`. If this property is not correctly set, `RecordStore.setConfiguration()` throws a `RecordStoreConfigurationException`.
`mergeBufferSize`	Oracle Internal Use. Oracle does not recommend modifying this property. This value must be greater than 0. The default value is 131072 bytes. The Record Store validates that the `mergeBufferSize` property is greater than 0. If this property is not correctly set, `RecordStore.setConfiguration()` throws a `RecordStoreConfigurationException`.
`recordCompressionEnabled`	The `recordCompressionEnabled` property specifies whether records are stored on disk in a compressed format. The `recordCompressionEnabled` property takes a Boolean value: • If set to `true`, records are stored on disk in a compressed format. • If set to `false`, records are stored on disk in an uncompressed format. If not specified, the value defaults to `false`.
`writeBufferSize`	Oracle Internal Use. Oracle does not recommend modifying this property. This value must be greater than 0. The default value is 104857600 bytes. The Record Store validates that the `writeBufferSize` property is greater than 0. If this property is not correctly set, `RecordStore.setConfiguration()` throws a `RecordStoreConfigurationException`.

Example of a configuration file for a Record Store instance

A sample configuration file is shown here:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<recordStoreConfiguration xmlns="http://recordstore.itl.endeca.com/">
    <btreePageSize>100</btreePageSize>
    <changePropertyNames/>
```

```
    <cleanerInterval>1.0</cleanerInterval>
    <dataDirectory>C:\Endeca\CAS\workspace\state\RS1\data</dataDirectory>
    <generationRetentionTime>168.0</generationRetentionTime>
    <idPropertyName>Endeca.Id</idPropertyName>
    <ignoreInvalidRecords>false</ignoreInvalidRecords>
    <indexWriteFlushInterval>50000</indexWriteFlushInterval>
    <jdbmSettings/>
    <recordCompressionEnabled>false</recordCompressionEnabled>
</recordStoreConfiguration>
```

Change properties and new Record Store instances

When the Content Acquisition System creates a new Record Store instance for a data source, CAS uses all record properties as change properties. In other words, CAS evaluates all properties on a record to determine if the record is different from the previous generation's record in the Record Store instance.

If desired, you can restrict the number of properties that CAS evaluates by configuring the `changeProperty¬Names` property. This improves performance by allowing CAS to check a sub-set of record properties when determining if a record has changed. In addition, CAS can check only properties that are meaningful to the data. For example, you can set a change property to check a file version number rather than a property for fetch time or trivial information.

Deleting stale generations of records

This topic provides guidance about how to delete stale generations of records by appropriately setting the `generationRetentionTime` property.

As a general rule, the value of `generationRetentionTime` should be greater than the sum of the following:

- The time between the start of two write operations to a Record Store instance.
- The time between the start of two delta read operations from a Record Store instance.
- Time for a margin of safety (For example, this includes time to revert to an earlier generation, fix any issues in the data, and re-crawl the data.)

For example, suppose a crawl, which writes to a Record Store, takes a few hours to run and runs once a day: the time between the start of two write operations is 24 hours. Next, suppose you run Forge once a day so the time between reads of the Record Store is 24 hours. Last, suppose you want to be able to revert to data up to three days old. You want a margin of safety of 72 hours. This means the value of `generationRetention¬Time` should be at least 120. In this scenario, a value of 120 ensures there are two generations in a Record Store instance.

Note: The Record Store applies a read-lock to the generation being read. If a generation with a read-lock exceeds the `generationRetentionTime` value, the generation is not deleted until the read is complete and the read-lock is released.

Disabling automatic management of a Record Store instance

If you want maintain a Record Store instance separately from its associated crawl configuration, you can disable automatic management of the Record Store instance by the CAS Server and move a Record Store

instance to another host as necessary. This may be useful if storage space for a Record Store instance is a concern and in some migration scenarios.

The `isManaged` property in a crawl configuration determines whether a Record Store instance is created or deleted at the same time as its associated data source configuration. The `isManaged` property has a value of `true` by default. This means that:

- A Record Store instance is automatically created when you create a crawl. The name of a new Record Store instance corresponds directly to the crawl name.
- The associated Record Store instance is automatically deleted when you delete a crawl.

If you disable the `isManaged` property by setting it to `false`, a Record Store instance is not created when you create the crawl configuration. You must create the Record Store instance manually, or configure the crawl to send output to a file. Likewise, any Record Store instance that you create for a crawl configuration is not deleted when you delete the crawl configuration.

To disable automatic management of a Record Store instance:

1. From a command prompt, run the `getCrawl` task of the CAS Server Command-line Utility. Use the `-f <arg>` flag to specify the name of the XML file to write the configuration to.

   For example, you might specify `-f configuration.xml`.

2. In the `configuration.xml` file for the crawl configuration, set the `isManaged` property to `false` as shown in the following example:

```
....
  <crawlConfig>
  ....
   <outputConfig>
    <moduleId>
      <id>Record Store</id>
    </moduleId>
    <moduleProperties>
     <moduleProperty>
      <key>isManaged</key>
      <value>false</value>
     </moduleProperty>
    </moduleProperties>
   </outputConfig>
....
  </crawlConfig>
....
```

3. Save and close the crawl configuration file.
4. Run the `updateCrawls` task of CAS Server Command-line Utility and pass in the crawl configuration file as input. For example:

```
updateCrawls --f configuration.xml
```

Performance considerations when using a Record Store instance

When reading or writing large numbers of records, some `READ_WRITE` operations can take long periods of time. Read operations generally take longer than write operations for similar size record sets, and the transaction time of a `READ_WRITE` operation grows approximately linearly as the number of records grows and the size of the records grows. For this reason, delta updates are generally faster than baseline updates.

If reading or writing operations cause performance concerns, there are several changes you can make:

- Use fast-access drives and RAID with striping on machines that host CAS record stores. This improves disk I/O for better overall record store performance.
- Reduce the `generationRetentionTime` setting so that fewer generations of records are stored. For details, see *Deleting stale generations of records* on page 37.
- Split a crawl into multiple crawls that use multiple Record Store instances.

Chapter 4

Running a crawl

This section provides information about running a crawl using the CAS Server.

Running a crawl

You can run a crawl using any of the following tools or methods: from the CAS Console, from the CAS Server Command-line Utility, and programmatically from the CAS Server API.

Crawling from the CAS Console for Endeca Workbench

The Data Sources page of CAS Console displays all data sources available for crawling. You can start crawling a particular data source by clicking **Start** in the **Acquire Data** column. For further information, see the *CAS Console Help for Endeca Workbench*.

Crawling from the CAS Server Command-line Utility

You can start and stop a crawl from the CAS Server Command-line Utility by running either the `startCrawl` or `stopCrawl` tasks. For further information, see the chapter in this guide on the CAS Server Command-line Utility.

Crawling programmatically from the CAS Server API

You can start a crawl by calling the `CasCrawler.startCrawl()` method from an application. For further information, see the *Endeca CAS API Guide*.

Order of execution in a crawl configuration

A crawl configuration specifies settings and processing instructions for a crawl. When you start a crawl, CAS Server executes the instructions in the following order: `sourceConfig`, `textExtractionConfig`, `manip¬ulatorConfig`, and `outputConfig`. This topic provides additional detail about execution order.

When CAS Server starts a crawl, the following happens:

1. CAS Server crawls files and folders according to the seeds and settings in `sourceConfig`, and CAS Server creates an Endeca record for each file and folder crawled.
2. If `textExtractionConfig` is enabled and contains document conversion settings, then CAS Server performs document conversion and stores the converted text as a property on the Endeca record.

3. If one or more `manipulatorConfig` elements are present, CAS Server passes the record to each manipulator for processing according to its `manipulatorConfig` settings. Manipulators execute in the order in which they are nested within `manipulatorConfigs`.
4. CAS Server then writes the record to a Record Store instance (or an output file) according to the settings in according to `outputConfig`.

This processing continues until all files and folders are crawled and all records are processed. In this way, Endeca records are propagated through a crawl configuration.

Full and incremental crawling modes

The CAS Server crawls a data source in one of two modes:

- `full` mode, in which all content is processed.
- `incremental` mode, in which only new, modified, or deleted content is processed.

Crawling in full mode

Crawling in *full* mode means that CAS processes all the content in a data source according to the filtering criteria you specify. As part of crawling a data source, CAS creates metadata information and stores it in a crawl history. This history includes the Id of each record and information about all properties on the record.

Crawling in incremental mode

Crawling in *incremental* mode means that CAS processes only that content whose metadata information, stored in the crawl history, has changed since the last crawl. Specifically, CAS checks all properties on the record to see if any have changed. If any properties have changed, the CAS Server crawls the content again. This is true in cases where CAS is calculating the incremental difference. An extension developer, using the CAS Extension API, may choose to calculate incremental changes in a data source extension.

CAS automatically determines which crawling mode is necessary. By default, CAS attempts to crawl in incremental mode. If necessary, CAS switches to crawling in full mode, if a crawl's configuration has `unavail¬ableIncrementalSwitchesToFullCrawl` set to `true`, and any of the following conditions are true:

- A data source has not been crawled before, which means no crawl history exists.
- A Record Store instance does not contain at least one record generation. (This applies to cases where the CAS Server is configured to output to a Record Store instance rather than a file on disk.)
- Seeds have been removed from the data source configuration (adding seeds does not require crawling in full mode).
- The document conversion setting has changed.
- Folder filters or file filters have been added, modified, or removed in the data source configuration.
- Repository properties have been changed, such as the **Gather native properties** option for file system data sources.

If `unavailableIncrementalSwitchesToFullCrawl` is set to `false` and any of the above conditions are true, the crawl fails and throw and exception.

This switch from incremental to full mode can occur no matter how you run a crawl (using the CAS Console, the CAS Server API, or the CAS Server Command-line Utility).

After you click **Start** in CAS Console, you can click the link under **Acquisition Status** to see a status message indicating whether a full or incremental crawl is running. After you crawl a data source using the API, the status message is returned.

Incremental mode and MDEX compatible output

An incremental crawl processes only data records. It does not process any configuration stored in the Endeca Configuration Repository (such as dimensions and properties, precedence rules, and so on), and it does not crawl dimension value records. By contrast, a full crawl processes data records, configuration in the Endeca Configuration Repository, and dimension value records.

Crawls and archive files

File system and CMS crawls can process archive files.

Archive expansion is disabled by default. To enable the feature, you must check **Expand archives** for the data source in CAS Console.

Archive expansion means that an Endeca record is created for each archived entry and its properties are populated. Text is extracted if the document conversion option is enabled. Note that native file properties are not gathered for archived entries even if that option is enabled for file system crawls. However, file and CMS permissions of the archive file are propagated to the archive entries.

Archive file support

An archive file is one that holds one or more archived entries (files or directories) within it. Two examples of archives are ZIP files and UNIX TAR files.

The CAS Server identifies archives by their file extensions for file system crawls, or mime types for CMS crawls. The following archive types are supported in file system data sources:

- JAR files (.jar extension)
- TAR files (.tar extension)
- GZIP-compressed Tar files (.tar.gz and .tgz extensions)
- ZIP files (.zip extension)

See the following sections for support details on ZIP and TAR files, as well as an overview of how archive files are handled.

Support for ZIP files

ZIP files are supported as follows:

- ZIP files can have either no compression or the standard Deflate compression algorithm. ZIP files that use a compression scheme other than the Deflate algorithm are not treated as ZIP files. In this case, one record is created for the file, with the `Endeca.File.IsArchive` property set to `false`.
- There is no support for ZIP files with password-protected entries. ZIP files that contain password-protected entries are not fully processed. The actual behavior depends on the form of password protection:
  - If using the AES-128 or AES-256 forms of password encryption, the file is not marked as a ZIP file. One record is created for the file, with the `Endeca.File.IsArchive` property set to `false`.
  - If using the ZipCrypto password protection, the ZIP is recognized, and each entry that is encountered in order that is not password-protected will have a record created for it. Once a password-protected entry is encountered, the processing on the ZIP stops, and no further records are created.
  - For a number of ZIP utilities, directory entries are not password-protected (so that only the files are encrypted), and that directory entries are often put at the beginning of a ZIP. One record is created for the file, with the `Endeca.File.IsArchive` property set to `true`, and additional records are created for those (directories) that are not encrypted.

- There is no support for entries that are split across multiple Zip files. Splitting a file over multiple ZIP files results in two kinds of ZIP files: those that store the partial data for the underlying file and and a "last" one that also stores the entry information. Different tools use different naming conventions, so sometimes the partials have a .zip extension and sometimes they do not. However, the last file will be a .zip file. These files are handled as follows:
  - The partial files will not be recognized as ZIP files. One record is created for the file, with the `Endeca.File.IsArchive` property set to `false`.
  - The last file will be recognized as a ZIP file, but its entry will be unreadable. One record is created for the file, with the `Endeca.File.IsArchive` property set to `true`.

When a Zip file is not treated as a valid ZIP file for any reason, the log file will contain a warning that the ZIP file in question contains an "invalid CEN header", and the record generated for the ZIP file will not indicate that it is an archive.

Note: JAR files are handled the same way as ZIP files. Therefore, any caveats that apply to ZIP files also apply to JAR files as well.

Support for Tar files

The supported Tar formats are the following:

- GNU Tar 1.13
- GNU Tar 1.12 or earlier
- UNIX V7
- POSIX.1-1988 (original USTAR format)
- Any of the above formats, compressed with GZip

Any format that is not listed above is considered an unsupported format. For example, the POSIX.1-2001 format is explicity not supported.

The CAS Server processes Tar files as follows:

- For supported formats, each entry in the Tar file is extracted and written as a record.
- For POSIX.1-2001 Tars, the entries are not extracted and a message is written to the log indicating that the format is not supported.
- For corrupted Tar entries:
  - If the first entry is corrupted, the entire Tar will not be extracted. Instead, it will be treated as any other non-archive file.
  - If any later entry is corrupted, the occurrence of the bad entry is logged. All prior entries are extracted and written as records to the output file.

How archive files are handled

The following is a detailed view of how the CAS Server handles archive files:

- An Endeca record is created for the archive file itself. This record has the `Endeca.File.IsArchive` property set to `true`.
- In addition to the top-level documents (files or directories), nested archive files are also processed.
- Document conversion (if enabled) is performed on all files within the archive, in accordance with document conversion filtering.
- A separate Endeca record is created for each document (including nested archives) found in the archive. The record is processed as follows:

- The record has the `Endeca.File.IsInArchive` property set to `true`. In addition, the `Endeca.File.SourceArchive` and `Endeca.File.PathWithinSourceArchive` properties are added with a reference to the parent archive.
- The filtering behavior works the same for archived files and directories (that is, files and directories in an archive) as it does for non-archived files and directories.
- For records from either file system or CMS crawls, the record Id is a concatenation of the `Endeca.File.SourceArchiveId` property and the `Endeca.File.PathWithinSourceArchive` property:
  - For file system records, the `Endeca.FileSystem.Path` property is the record Id. This property is a canonical string pointing to the file within the archive, and follows this format:
    ```
    /path/to/archive//path/to/archivedfile
    ```
  - For CMS records, the `Endeca.Id` property is the record Id. This property is a canonical string pointing to the file within the archive, and follows this format:
    ```
    reposId:itemId[:optionalContentPieceId]//path/to/archivedfile
    ```

Note:

- Double delimiters represent the boundaries of the archive.
- Path delimiters for the value of the `PathWithinSourceArchive` property appear as forward slashes (they are platform-independent).
- Path delimiters for the value of the `Endeca.FileSystem.Path` property are platform-dependent, so in the case of Windows files, path delimeters on this property appear as backslashes. For example:
  ```
  C:\path\to\archive//path/to/archivedfile
  ```

In the case of nested archives, the `Endeca.File.PathWithinSourceArchive` property takes the following format:

```
//path/to/nested/archive//path/within/nested/archive
```

- While the properties of archived entries are obtained in an Endeca record, the entries themselves are not physically extracted from the archive (that is, no new files are permanently saved to disk).
- If an archive has entries with identical names, the first entry that is processed is kept (that is, an Endeca record is created for it) and the duplicate entry is ignored.
- Seeds are restricted to actual files or directories or entries. That is, seeds cannot point to archived files or directories.

The above behavior is the default for all archives crawled. To avoid processing archives, disable the Expand archives option for the data source.

About writing records to a Record Store instance

A Record Store instance is tightly integrated with the output produced by CAS Server. CAS Server writes the output for file system and CMS data sources directly to a Record Store instance by default, instead of to a file on disk.

About the record output file

This topic describes record output files for full and incremental crawl modes.

You configure the attributes of an record output file from the CAS Server API or in a crawl configuration file that you provide to a command in the CAS Server Command-line Utility.

For example, you set the path of the output directory with the `outputDirectory` property in the API or path in the configuration file. If you do not specify an output directory, a default name of `output` is used for the `crawlID` sub-directory and it is located in the CAS Server's workspace directory.

Record output file

The prefix for the name of a crawl output file is set by the `outputPrefix` property (in the API) or key (in the configuration file). If you do not specify an output prefix, a default name of `CrawlerOutput` is used.

The full name of the output file also depends on two other configuration settings:

- The `outputXml` property. This specifies whether the output format is XML (with a file extension of `.xml`) or Binary (with a file extension of `.bin`).
- The `outputCompressed` property. This determines whether the output file is compressed. If compression is enabled, a `.gz` file extension is added to the `.xml` or `.bin` extension. No extension is added if compression is not enabled.

In addition to the output prefix described above, a second prefix is automatically added to the filename to distinguish which type of crawl was run:

- For full crawls, the `-FULL` suffix is added (e.g., `CrawlerOutput-FULL.xml`).
- For incremental crawls, the `-INCR` suffix is added (e.g., `CrawlerOutput-INCR.xml`).

The maximum size of a binary output file is 512 megabytes. If the maximum size is reached and more records need to be output, the crawler rolls the output into another output file. To distinguish rollover files, the `-sgmt000` prefix is added to the first file, `-sgmt001` is added to the second file, and so on, as shown in this example:

```
CrawlerOutput-FULL-sgmt000.bin.gz
CrawlerOutput-FULL-sgmt001.bin.gz
```

The maximum size of binary output files is not configurable. Note that unlike the binary format, if you choose XML, only one file is output, regardless of its size.

Archived output files

The first time that CAS Server crawls a data source, the output file is named as described in the previous section. For example, if you run a full crawl, the output filename might be `CrawlerOutput-FULL-sgmt000.bin.gz`. If you then run a second crawl (for example, an incremental crawl), the CAS Server works as follows:

1. A directory named `archive` is created under the output directory.
2. The original `CrawlerOutput-FULL-sgmt000.bin.gz` file is moved to the `archive` directory and is renamed by adding a timestamp to the name; for example:

   ```
   CrawlerOutput-FULL-20071026140235-sgmt000.bin.gz
   ```

3. The output file from a second incremental run is named `CrawlerOutput-INCR-sgmt000.bin.gz` and is stored in the output directory.
4. For every subsequent crawl using the same output directory, steps 2 and 3 are repeated.

The timestamp format used for renaming is:

```
YYYYMMDDHHmmSS
```

where:

- YYYY is a four-digit year, such as 2009.
- MM is the month as a number (01-12), such as 10 for October.
- DD is the day of the month, such as 25 (for October 25th).
- HH is the hour of the day in a 24-hour format (00-23), such as 14 (for 2 p.m.).
- mm is the minute of the hour (00-59).
- SS is the second of the minute (00-59).

The timestamp format is not configurable.

Chapter 5

Running the CAS sample applications

This section describes the sample applications.

About the sample CAS applications

This section describes how to run the sample applications of the Content Acquisition System.

There are six sample applications that you can use to exercise the functionality of CAS:

- A Forge application with a Record Store adapter that writes records to a Record Store instance. This is stored in `CAS\<version>\sample\forge-to-recordstore`.
- A Forge application with a Record Store adapter that reads records from a Record Store instance. This is stored in `CAS\<version>\sample\recordstore-to-forge`.
- A Forge application with multiple Record Store adapters that read records from multiple Record Store instances. This is stored in `CAS\<version>\sample\multiple-recordstore-to-forge`.
- A Web Crawler that writes its output to a Record Store instance instead of to a file on disk. This is stored in `CAS\<version>\sample\webcrawler-to-recordstore`.
- A Java client that communicates with a Record Store instance and issues record access requests. This is stored in `CAS\<version>\sample\recordstore-java-client`.
- A Java client that communicates with the CAS Service and issues file system crawling requests. This is stored in `CAS\<version>\sample\cas-server-java-client`.

Order of running the Forge applications

The Forge applications demonstrate how you would run baseline and partial updates using Record Store instances as the storage mechanism. The order of running these applications is the following:

1. Run the `run-sample` script in the `forge-to-recordstore` application. This populates a Record Store with three baseline records.
2. Run the `run-sample` script in the `recordstore-to-forge` application. This causes Forge to run a baseline update using the generation in a Record Store instance. In an actual depolyment, you would run Dgidx on the Forge records and then start the MDEX Engine with the output.
3. Run the `run-partial-sample` script in the `forge-to-recordstore` application. This adds an incremental record to the Record Store.
4. Run the `run-partial-sample` script in the `recordstore-to-forge` application. This causes Forge to run a partial update using the incremental record from the Record Store. In an actual deployment, a Dgraph update command would then be issued so that the MDEX Engine could upload the incremental record.

 **Note:** These scripts do not start or update the MDEX Engine.

Writing records from Forge into the Record Store

This sample Forge application writes records from Forge into a Record Store instance.

The `sample\forge-to-recordstore` directory contains a simple Forge application that demonstrates how Forge can write records to a Record Store instance via a Record Store adapter. The application contains two Forge pipelines:

- A baseline pipeline, which is run with the `run-sample` script and writes three records from the `forge-input-data\data.txt` file.
- A partial-update pipeline, which is run with the `run-partial-sample` script and writes one record from the `forge-partial-input-data\partial-data.txt` file.

Both pipelines have only two components:

- An input record adapter (named LoadData) that reads in the records from the appropriate source file (`data.txt` or `partial-data.txt`).
- An output record adapter (named RecordStoreSink) that is a Record Store adapter (i.e., can write the records to a Record Store instance).

If you open the `forge-to-recordstore\forge-config\recordstore.esp` project in Developer Studio, you can see that the RecordStoreSink adapter is a custom record adapter with these two Java properties set on the **General** tab:

- Class is: `com.endeca.itl.recordstore.forge.RecordStoreSink`
- Class path is: `<install path>/CAS/<version>/lib/recordstore-forge-adapter/recordstore-forge-adapter-11.0.0.jar`

The **Pass Throughs** tab uses the following pass-through name/value pairs:

- For both pipelines, the `HOST` pass-through is set to `localhost` (which is the host machine for the CAS Service).
- For both pipelines, the `PORT` pass-through is set to `8500` (which is the port on which the CAS Service is listening).
- For both pipelines, the `INSTANCE_NAME` pass-through is set to `rs1` (which is the unique name for the Record Store instance).
- For the baseline pipeline, the `WRITE_TYPE` pass-through is set to `BASELINE` while the partial-update pipeline has the pass-through set to `DELTA` (which means that the records are treated as incremental records).

You can also use the `RECORDS_PER_TRANSFER` and `TRANSFER_TYPE` pass-throughs documented in the topic titled "Reading records from the Record Store into Forge."

The `sample\forge-to-recordstore` directory also contains a `recordstore-configuration.xml` file that is specifically configured for records emitted by Forge. In particular, the file has these two configuration properties:

```
<changePropertyNames/>
<idPropertyName>Endeca.Id</idPropertyName>
```

Setting the `idPropertyName` is important because its value is used by the Record Store instance to generate a unique record ID.

To run the forge-to-recordstore sample application:

1. Start the Endeca CAS Service if it is not already running.

- Windows: Start the CAS Service from the Windows Services console.
- UNIX: Run the `cas-service.sh` script.

2. From a command prompt window, change to the `sample\forge-to-recordstore` directory.
3. To use the baseline pipeline, run the `run-sample` script. To run the partial-update pipeline, run the `run-partial-sample` script.

When the `forge-to-recordstore` sample finishes, you see several pages of INFO messages. Scroll through the messages and look for the following:

```
...
INFO    01/19/09 14:43:55.293 UTC (1232376235293)       FORGE   {baseline}: Pro¬
cessed 3 records.
INFO    01/19/09 14:43:55.293 UTC (1232376235293)       FORGE   {baseline}:
Stopping stats clock at
'Mon Jan 19 09:43:55 2009'.
INFO    01/19/09 14:43:55.293 UTC (1232376235293)       FORGE   {baseline}: Fin¬
ished processing records:
...
```

After running the baseline version of the application (via the `run-sample` script), run the baseline version of the `sample\recordstore-to-forge` application to cause Forge to run a baseline update with the new records.

Reading records from the Record Store into Forge

This sample Forge application reads records from a Record Store instance into Forge.

The `sample\recordstore-to-forge` directory contains a Forge application that demonstrates how Forge can run baseline and partial updates by reading records from a Record Store instance via a Record Store adapter. The application contains a dimension server and a property mapper, but it does not include an Indexer adapter (instead, it uses an output Record adapter to write the output to a file on disk).

The application contains two Forge pipelines:

- A baseline pipeline, which is run with the `run-sample` script and runs a baseline update with records read from a Record Store instance.
- A partial-update pipeline, which is run with the `run-partial-sample` script and runs a partial update by reading from a Record Store instance.

The `recordstore-to-forge` pipeline has two input and output components:

- An input record adapter (named RecordStoreSource) that is a Record Store adapter (i.e., it can read records from a Record Store instance).
- An output record adapter (named StoreData) that writes the records to an XML file in `forge-output-data\data.xml`.

If you open the `recordstore-to-forge\forge-config\recordstore.esp` project in Developer Studio, you can see that the RecordStoreSource adapter is a custom record adapter with these two Java properties set on the **General** tab:

- Class is: `com.endeca.itl.recordstore.forge.RecordStoreSource`
- Class path is: `../../lib/recordstore-forge-adapter/recordstore-forge-adapter-11.0.0.jar`

The **Pass Throughs** tab uses the following pass-through name/value pairs:

- Set an `INSTANCE_NAME` pass-through to the name of the unique Record Store instance is that you want to read from. For example, `INSTANCE_NAME = crawlID`.

- For both pipelines, the HOST pass-through is set to localhost (the host machine for the Record Store instance).
- For both pipelines, the PORT pass-through is set to 8500 (the port on which the Record Store instance is listening).
- For the baseline pipeline, the READ_TYPE pass-through is set to BASELINE while the partial-update pipeline has the pass-through set to DELTA.
- For both pipelines, the CLIENT_ID pass-through is set to CLIENT_1.

The CLIENT_ID pass-through specifies the client ID to be set for the generation that is being read in. In effect, this pass-through is performing the set-last-read-generation task that can be performed with the Record Store command-line utility (i.e., state is being set for the client, which is Forge in this case). This pass-through can be used only for READ_TYPE operations.

To run the recordstore-to-forge sample application:

1. Start the Endeca CAS Service if it is not already running.
   - Windows: Start the CAS Service from the Windows Services console.
   - UNIX: Run the cas-service.sh script.
2. Make sure that the Record Store has at least one record generation. For example, you may have to run the forge-to-recordstore sample application to produce the generation.
3. From a command prompt window, change to the sample\recordstore-to-forge directory.
4. To run a baseline update, run the run-sample script. To run a partial update, run the run-partial-sample script.

When the recordstore-to-forge sample finishes, its output is written to the data.xml file in the forge-output-data directory. In addition, you see several screens of INFO messages. Scroll through the messages and look for the following completion messages (elipses indicate deleted INFO messages):

```
INFO    01/19/09 17:10:29.424 UTC (1232385029416)      FORGE   {config}: Starting:
 RecordAdapter 'StoreData'.

...

INFO    01/19/09 17:10:33.283 UTC (1232385033283)       FORGE   {baseline}:
(AdapterRunner): Adapter
 class: com.endeca.itl.recordstore.forge.RecordStoreSource

...

INFO    01/19/09 17:10:35.580 UTC (1232385035580)       FORGE   {baseline}: Pro¬
cessed 3 records.
INFO    01/19/09 17:10:35.580 UTC (1232385035580)       FORGE   {baseline}:
Stopping stats clock at
'Mon Jan 19 12:10:35 2009'.
INFO    01/19/09 17:10:35.580 UTC (1232385035580)       FORGE   {baseline}: Fin¬
ished processing records:
```

Reading records from multiple Record Stores into Forge

This sample Forge application reads records from multiple Record Store instances into Forge.

The sample\multiple-recordstore-to-forge directory contains a Forge application that demonstrates how Forge can run baseline and partial updates by reading records from multiple Record Store instances via a Record Store adapter. The application contains a dimension server and a property mapper, but it does not include an Indexer adapter (instead, it uses an output Record adapter to write the output to a disk file).

The application contains two Forge pipelines:

- A baseline pipeline, which is run with the `run-sample` script and runs a baseline update with records read from multiple Record Store instances.
- A partial-update pipeline, which is run with the `run-partial-sample` script and runs a partial update by reading from multiple Record Store instances.

The `multiple-recordstore-to-forge` pipeline has two input and output components:

- An input record adapter (named RecordStoreSource) that is a Record Store adapter (i.e., it can read records from multiple Record Store instances).
- An output record adapter (named StoreData) that writes the records to an XML file in `forge-output-data\data.xml`.

If you open the `multiple-recordstore-to-forge\forge-config\recordstore.esp` project in Developer Studio, you can see that in the Pipeline Diagram, the RecordStoreSource adapter is a custom record adapter with these two Java properties set on the **General** tab (in the Partial Pipeline, the custom record adapter is called LoadIncrementalCrawls):

- Class is: `com.endeca.itl.recordstore.forge.MultipleRecordStoreSource`
- Class path is: `../../lib/recordstore-forge-adapter/recordstore-forge-adapter-11.0.0.jar`

The **Pass Throughs** tab uses the following pass-through name/value pairs:

- For both pipelines, the `HOST` pass-through is set to `localhost` (the host machine for the CAS Services).
- For both pipelines, the `PORT` pass-through is set to `8500` (the port on which the CAS Service is listening).
- For the baseline pipeline, the `READ_TYPE` pass-through is set to `BASELINE` while the partial-update pipeline has the pass-through set to `DELTA`.
- For both pipelines, the `CLIENT_ID` pass-through is set to `CLIENT_1`.

The `CLIENT_ID` pass-through specifies the client ID to be set for the generation that is being read in. In effect, this pass-through is performing the `set-last-read-generation` task that can be performed with the Record Store command-line utility (i.e., state is being set for the client, which is Forge in this case). This pass-through can be used only for `READ_TYPE` operations.

To run the multiple-recordstore-to-forge sample application:

1. Start the Endeca CAS Service if it is not already running.
   - Windows: Start the CAS Service from the Windows Services console.
   - UNIX: Run the `cas-service.sh` script.
2. Make sure that the Record Store has at least one record generation. For example, you may have to run the `forge-to-recordstore` sample application to produce the generation.
3. From a command prompt window, change to the `sample\multiple-recordstore-to-forge` directory.
4. To run a baseline update, run the `run-sample` script. To run a partial update, run the `run-partial-sample` script.

When the multiple-recordstore-to-forge sample finishes, its output is written to the `data.xml` file in the `forge-output-data` directory. In addition, you see several screens of INFO messages. Scroll through the messages and look for the following completion messages (elipses indicate deleted INFO messages):

```
INFO    01/19/09 17:10:29.424 UTC (1232385029416)      FORGE   {config}: Starting:
 RecordAdapter 'StoreData'.

...

INFO    01/19/09 17:10:33.283 UTC (1232385033283)       FORGE   {baseline}:
```

```
(AdapterRunner): Adapter
 class: com.endeca.itl.recordstore.forge.RecordStoreSource

...

INFO    01/19/09 17:10:35.580 UTC (1232385035580)       FORGE   {baseline}: Pro¬
cessed 3 records.
INFO    01/19/09 17:10:35.580 UTC (1232385035580)       FORGE   {baseline}:
Stopping stats clock at
'Mon Jan 19 12:10:35 2009'.
INFO    01/19/09 17:10:35.580 UTC (1232385035580)       FORGE   {baseline}: Fin¬
ished processing records:
```

Running the sample Web Crawler

The sample Web Crawler application writes output to a Record Store instance.

The sample Web Crawler application is located in the `sample\webcrawler-to-recordstore` directory. The directory contains the `run-sample` scripts that runs the sample Web Crawler.

The directory also contains a `recordstore-configuration.xml` file that is configured for records produced by the Web Crawler. In particular, the file has these two configuration properties:

```
<changePropertyNames/>
<idPropertyName>Endeca.Id</idPropertyName>
```

Setting the `idPropertyName` is important because the Record Store instance generates a unique record ID based on the property value.

The sample Web Crawler is configured to write its output directly to a Record Store instance. Specifically, the `site.xml` file in the `conf` directory has these three output properties:

```
<property>
    <name>output.recordStore.host</name>
    <value>localhost</value>
    <description>
    The host of the record store service.
    Default: localhost
    </description>
</property>

<property>
    <name>output.recordStore.port</name>
    <value>8500</value>
    <description>
    The port of the record store service.
    Default: 8500
    </description>
</property>

<property>
    <name>output.recordStore.instanceName</name>
    <value>rs-web</value>
    <description>
    The name of the record store service.
    Default: rs-web
    </description>
</property>
```

Be sure to change the values if you create a Record Store instance with a different host name and port.

To run the sample Web Crawler:

1. Start the Endeca CAS Service if it is not already running.
   - Windows: Start the CAS Service from the Windows Services console.
   - UNIX: Run the `cas-service.sh` script.
2. From a command prompt window, change to the `sample\webcrawler-to-recordstore` directory.
3. Run the `run-sample` script.

When the Web Crawler finishes, its output is written to the Record Store, instead of to a file on disk. If you check `cas-service.log`, you should see these messages similar to this example:

```
Starting new transaction with generation Id 1
Started transaction 1 of type READ_WRITE
Marking generation committed: 1
Committed transaction 1
```

In the example, the Record Store is storing the record generation with an ID of 1.

Using the CAS Server Java Client

The Endeca CAS Server API allows users to build client programs that invoke the Endeca CAS Server to programmatically modify and control a variety of file system and CMS crawling operations.

CAS Server Java Client Sample Files and Directories

This topic describes the contents of the CAS Server Java Client directory.

The CAS Server Java Client (in the `/sample` directory) has the following directory structure:

```
/cas-server-java-client
  /lib
  /src
  .classpath
  .project
  build.xml
```

The contents are as follows:

- `lib` – Contains the Java libraries for the CAS Server Java client application.
- `src` – Contains the Java source file for the CAS Server Java Client application.
- `.classpath` – The classpath file for the Eclipse project.
- `.project` – The Eclipse project file for the recordstore-java-client project.
- `build.xml` – The Ant build file for the Record Store Java client application.

About the CAS Server Java Client Program

The CAS Server Java Client (as coded in the `CasServerSampleClient.java` source file) demonstrates a number of basic crawling operations.

The Endeca CAS Server Java Client is intended to provide a working example of a client that communicates with a running CAS Server and issues file system crawling requests. The sample client program is therefore a template that you can use as a basis for your own client program.

The package includes all the libraries needed to build clients. It also includes an Ant build script (that can compile and run the sample program) as well as Eclipse `.project` and `.classpath` files for the sample client.

Important: Please note that before starting Eclipse, you should run at least `ant compile` so that Eclipse can find the generated Web service stubs.

The sample client application performs the following actions:

1. Makes a connection to the CAS Service.
2. Creates a new file system crawl (named SampleClientTestCrawl), with the current working directory of the sample client (`.\` on Windows or `./` on UNIX) as the seed.
3. Runs a full crawl.
4. Updates the crawl configuration by adding file filters and enabling document conversion.
5. Runs a second full crawl, this time using the new filters and extracting text from documents.
6. Deletes the sample crawl.

Note that a default time limit of 10 seconds is set on both crawls, which means that in most cases the crawl output will not contain all the files on your file system.

The output files are written to the `workspace/output/SampleClientTestCrawl` directory, using a non-compressed XML file format. You can use a text editor to view the contents of the output.

Building and Running the Java Client with Ant

The Ant `build.xml` file can compile and run the sample client program.

As with any Ant build file, you can run a specific target or run them all at once. Before starting Eclipse, you should run at least the `compile` target so that Eclipse can find the generated Web service stubs.

The file has the following targets:

- `compile` - Runs javac to compile the generated client stubs and sample application.
- ·`run-demo` - Runs the previous two targets and then runs the sample client application.
- `clean` - Deletes the build directory.

To run the Ant build script:

1. Start the Endeca CAS Service if it is not already running.
   - Windows: Start the CAS Service from the Windows Services console.
   - UNIX: Run the `cas-service.sh` script.
2. From a command prompt, navigate to the `cas-server-java-client` directory and issue the following command to compile and run the sample client demo:

   ```
   ant run-demo [--host <host name>] [--port <port number>]
   ```

   Note: You can issue the `ant compile` command if you just want to compile (but not run) the sample client program.

The demo file system crawl (named `SampleClientTestCrawl`) will use `C:\` on Windows and `/` on UNIX as the seed. When the demo crawl finishes, the CAS Service's `workspace/output/SampleClientTestCrawl` directory should contain two XML-format output files: `CrawlerOutput-FULL.xml` will have the content of the second crawl (i.e., the updated crawl with file filters), while the timestamped file in the `archive` directory will have the content from the first crawl.

Opening the cas-server-java-client project in Eclipse

If you use Eclipse for your projects, the sample client package includes Eclipse `.project` and `.classpath` files.

To load the sample client project:

1. Make sure that you have run the Ant build file with at least the `compile` target. This generates the necessary Web service stubs.
2. Start Eclipse.
3. Import the project:
   a) Open the **File** menu.
   b) Click **Import....**
   c) Expand the **General** folder.
   d) Select **Existing Projects into Workspace**.
   e) Select the `cas-server-java-client` project.
   f) Click **Finish**.

Running the operations of the Java Client

Assuming that you have opened the `CasServerSampleClient.java` source in Eclipse or another editor, you should note certain important operations of the `Main` class.

1. The values for the host and port of the CAS Service are set by first reading the `commandline.properties` file. If they do not exist, defaults of `localhost` and `8500` are used.

```
String host = System.getproperty(CAS_HOST_PROPERTY);
if (host == null || "".equals(host)) {
   host = "localhost";
  }
  if (port == null || "".equals(port)) {
   port = "8500";
  }
```

2. Arguments are created for the WSDL URL (the service definition interface) and the QName.

```
final URL wsdlUrl = new URL("http://" + host + ":" + port + "/cas?wsdl");
final QName name = new QName("http://endeca.com/itl/cas/2011-12",
"CasCrawlerService");
```

3. Using the WSDL URL and QName values, create a Web service locator and then use the `CasCrawlerService.getCasCrawlerPort()` method to get a handle to the CAS Service port.

```
CasCrawlerService service =  new CasCrawlerService(wsdlUrl, name);
CasCrawler crawler = service.getCasCrawlerPort();
```

4. Using a `CrawlId` object, set the name of the crawl.

```
CrawlId crawlId = new CrawlId();
crawlId.setId("SampleClientTestCrawl");
```

5. Using the `sampleCreateCrawl` method, create the new file system crawl. Text extraction is not enabled, which means that a probe crawl will be run. Note that the `CasCrawler.createCrawl()` method actually creates the crawl.

```
System.out.println("Creating Crawl with CrawlId '" + crawlId.getId() + "' ...");
sampleCreateCrawl(crawler, crawlId);
```

6. Using the `sampleRunFullCrawl` method, run the probe crawl, specifying a maximum of 10 seconds for the crawl duration. The `CasCrawler.startCrawl()` method is used to actually start the crawl, and then the `CasCrawler.stopCrawl()` method is used to stop crawl after 10 seconds has elapsed.

```
System.out.println("Running probe crawl...");
sampleRunFullCrawl(crawler, crawlId, 10);
```

7. Using the `sampleUpdateCrawlAddingFiltersAndTextExtraction` method, enable text extraction and set wildcard (`htm*`) filters that are evaluated against the `Endeca.FileSystem.Extension` record property. The original crawl configuration is retrieved with the `CasCrawler.getCrawlConfig()` method and the updated configuration is sent to the CAS Server with the `CasCrawler.updateConfig()` method.

```
System.out.println("Adding filters and enabling text extraction...");
sampleUpdateCrawlAddingFiltersAndTextExtraction(crawler, crawlId);
```

8. Using the `sampleRunFullCrawl` method, run a second full crawl that does text extraction and uses the added filters. As with the previous crawl, a maximum of 10 seconds is specified for the crawl duration.

```
System.out.println("Running full crawl...");
sampleRunFullCrawl(crawler, crawlId, 10);
```

9. Using the `sampleDeleteCrawl` method, delete the SampleClientTestCrawl demo crawl. Note that the class uses the `CasCrawler.deleteCrawl()` method to actually delete the crawl.

```
System.out.println("Deleting crawl...");
sampleDeleteCrawl(crawler, crawlId);
```

The sample client program also shows the use of other CAS Server API functions, such as the `CasCrawler.listCrawls()`, `CasCrawler.getStatus()` and `CasCrawler.getMetrics()` methods.

You can modify the file and add other crawling operations, such as changing the output options (to send output to a Record Store instance), adding other types of filters (including date and regex filters), enabling archive expansion, and even returning information about the CAS Server. You can also use the sample code as a basis for creating and running CMS crawls.

These operations, and the methods that call them, are described elsewhere in this guide.

Using the Record Store Java Client

The Endeca Record Store Java Client package is intended to provide a working example of a client that communicates with a Record Store instance and issues record access requests. The sample client program is therefore a template that you can use as a basis for your own client program.

The Endeca Record Store API allows users to build client programs that invoke an Endeca Record Store instance to programmatically write records to and read records from the Record Store.

The Record Store API consists of two components:

- Record Store core (WSDL) classes. These are classes that you generate from the Record Store WSDL file using a third-party tool (such as Apache CXF 2.0). For the sake of convenience, Java versions of these classes are included in the `recordstore-api-11.0.0.jar` library in the sample client package.
- Record Store utility (helper) classes, such as the `RecordStoreLocator`, `RecordStoreReader`, and `RecordStoreWriter` classes which are used in the sample client applications. These Java classes are also included in the `recordstore-api-11.0.0.jar` library.

The sample client package includes all the libraries needed to build clients. It also includes an Ant build script (that can compile and run the sample applications) as well as Eclipse `.project` and `.classpath` files for the sample client.

See the Oracle Technology Network for the *Endeca CAS API Guide* (which documents the Record Store API).

Record Store Client Sample Files and Directories

This topic describes the contents of the Record Store Java Client directory.

The Record Store Java Client has the following directory structure:

```
/recordstore-java-client
  /conf
  /lib
  /src
  .classpath
  .project
  build.xml
  run-sample-reader.bat
  run-sample-reader.sh
  run-sample-writer.bat
  run-sample-writer.sh
```

The contents are as follows:

- `conf` – Contains the `log4j.properties` logger configuration file for the sample client application.
- `lib` – Contains the Java libraries for the Record Store Java client application.
- `src` – Contains the Java source files for the Record Store java client application.
- `.classpath` – The classpath file for the Eclipse project.
- `.project` – The Eclipse project file for the recordstore-java-client project.
- `build.xml` – The Ant build file for the Record Store Java client application.
- The scripts to run the sample reader and sample writer client applications (`run-sample-reader.sh` and `run-sample-writer.sh` for UNIX, and `run-sample-reader.bat` and `run-sample-writer.bat` for Windows).

About the Record Store Sample Client Applications

The two Record Store sample client applications demonstrate the write and read functionality of the Record Store API.

The writer client

The writer client (in the `SampleWriter.java` source file) performs the following actions:

1. Creates a record that will be written to the Record Store.
2. Makes a connection to a Record Store instance, assumed to reside on the `localhost` machine with a port of `8500`.
3. Starts a `READ_WRITE` transaction.
4. Using the `RecordStoreWriter` methods, writes the record to the Record Store.
5. Commits the write transaction.

The reader client

The reader client (in the `SampleReader.java` source file) performs the following actions:

1. Makes a connection to a Record Store instance, assumed to reside on the `localhost` machine with a port of `8500`.
2. Starts a READ transaction.
3. Gets the ID of the last-committed generation.

4. Using the `RecordStoreReader.next()` method, reads the record from the Record Store and then writes its contents to standard output.
5. Commits the read transaction.

Note: If either application throws a `RecordStoreFault` exception, it is caught and the transaction is rolled back.

Building and Running the Sample Writer Client with Ant

The Ant `build.xml` file can compile and run the sample writer client program.

The file has the following targets:

- `init` – Creates the build directory structure that will be used by the compile target.
- `compile` – Runs javac to compile the sample client application.
- `run-sample-writer` – Runs the previous two targets and then runs the sample client writer application.
- `run-sample-reader` – Runs the init and compile targets, and then runs the sample client reader application.
- `clean` – Deletes the build directory.

To run the sample writer client with the Ant build script:

1. Start the Endeca CAS Service if it is not already running.
   - Windows: Start the CAS Service from the Windows Services console.
   - UNIX: Run the `cas-service.sh` script.
2. From a command prompt, navigate to the `recordstore-java-client` directory and issue the following command to compile and run the sample writer client demo:

```
run-sample-writer
```

The sample writer client's output messages should be similar to this example:

```
Buildfile: build.xml

init:
  [mkdir] Created dir: C:\Endeca\CAS\11.0.0\sample\recordstore-java-client\build

compile:
  [javac] Compiling 2 source files to C:\Endeca\CAS\11.0.0\sample\recordstore-
java-client\build

run-sample-writer:
  [java] Setting record store configuration ...
  [java] Starting a new transaction ...
  [java] Writing records ...
  [java] Committing transaction ...
  [java] DONE

BUILD SUCCESSFUL
```

```
Total time: 14 seconds
```

You can use the `-c` (count) option with the read-baseline task of the Record Store Command-line Utility to determine if the Record Store has any records:

```
C:\Endeca\CAS\11.0.0\bin> recordstore-cmd.bat read-baseline -a rs1 -c
Records read: 1
```

As the example shows, the Record Store has the one record that was read in by the sample writer client.

Building and Running the Sample Reader Client with Ant

The Ant build.xml file can compile and run the sample reader client program.

The file has the following targets:

- `init` – Creates the build directory structure that will be used by the compile target.
- `compile` – Runs javac to compile the sample client application.
- `run-sample-writer` – Runs the previous two targets and then runs the sample client writer application.
- `run-sample-reader` – Runs the init and compile targets, and then runs the sample client reader application.
- `clean` – Deletes the build directory.

To run the sample reader client with the Ant build script:

1. Start the Endeca CAS Service if it is not already running.
   - Windows: Start the CAS Service from the Windows Services console.
   - UNIX: Run the `cas-service.sh` script.
2. From a command prompt, navigate to the `recordstore-java-client` directory and issue the following command to compile and run the sample reader client demo:

   ```
   run-sample-reader
   ```

The sample reader client's output messages should be similar to this example:

```
Buildfile: build.xml

init:

compile:

run-sample-reader:
  [java] Starting a new transaction ...
  [java] Getting the last committed generation ...
  [java] Reading records ...
  [java]   RECORD: [Endeca.FileSystem.Path=id1, property.name=property.value]
  [java] 1 record(s) read
  [java] Committing transaction ...
  [java] DONE

BUILD SUCCESSFUL
Total time: 8 seconds
```

As the example output shows, the properties of the record in the Record Store are written out.

Opening the recordstore-java-client project in Eclipse

If you use Eclipse for your projects, the sample client package includes Eclipse `.project` and `.classpath` files.

As a prerequisite, make sure that you have run the Ant build file with at least the `compile` target. This will generate the necessary Web service stubs.

To load the sample client project:

1. Start Eclipse.
2. Import the project:
   a) Open the **File** menu.
   b) Click **Import...**
   c) Expand the **General** folder.
   d) Select **Existing Projects into Workspace**
   e) Select the `recordstore-java-client` project.
   f) Click **Finish**.

Running the operations of the Sample Writer Client

This section provides an overview of the more important operations of the sample writer client program. You can modify the files and add other Record Store operations.

The methods for these operations are described in the *CAS API Guide*, and in the *Record Store Javadocs*.

Assuming that you have opened the `SampleWriter.java` source in Eclipse or another editor, you should note the following important operations:

1. A constant is set for the value of the `idPropertyName` configuration property that is used for the Record Store instance.

```
public static final String PROPERTY_ID = "Endeca.FileSystem.Path";
```

2. After a LinkedList of Record objects is instantiated, a record is created and added to the list. The `Record.addPropertyValue()` method is used to add the property values to the record.

```
Collection<Record> records = new LinkedList<Record>();
Record record = new Record();
record.addPropertyValue(new PropertyValue(PROPERTY_ID, "id1"));
record.addPropertyValue(new PropertyValue("property.name",
   "property.value"));
records.add(record);
```

3. Using the `RecordStoreLocator` utility class, create a Web service locator with host name, port number, and Record Store instance name:

```
RecordStoreLocator locator = RecordStoreLocator.create("localhost", 8500,
recordStoreInstance);
```

4. Use the `RecordStore.getRecordStore()` method to establish a connection to the Record Store instance:

```
RecordStore recordStore = locator.getRecordStore();
```

5. Using the transaction ID created by the `RecordStore.startTransaction()` method, the `Record¬StoreWriter.createWriter()` method is used to create a writer.

```
tId = recordStore.startTransaction(TransactionType.READ_WRITE);
```

```
RecordStoreWriter writer =
   RecordStoreWriter.createWriter(recordStore, tId);
```

6. The writer first writes a "Delete All" record, then writes the sample record, and finally closes the writer.

```
writer.deleteAll();
writer.write(records);
writer.close();
```

7. The `RecordStore.commitTransaction()` method closes the transaction.

```
recordStore.commitTransaction(tId);
```

Note: If a `RecordStoreFault` exception occurs during the write process, it is caught and the `RecordStore.rollbackTransaction()` method rolls back the `READ_WRITE` transaction.

Running the operations of the Sample Reader Client

This section provides an overview of the more important operations of the sample reader client program. You can modify the files and add other Record Store operations.

The methods for these operations are described in the*CAS API Guide* and in the *Record Store API Reference (Javadoc)*.

The `SampleReader.java` source program executes the following important operations:

1. Using the `RecordStoreLocator` utility class, create a Web service locator with host name, port number, and Record Store instance name:

```
RecordStoreLocator locator = RecordStoreLocator.create("localhost", 8500,
recordStoreInstance);
```

2. Use the `RecordStore.getRecordStore()` method to establish a connection to the Record Store instance:

```
RecordStore recordStore = locator.getRecordStore();
```

3. Using the transaction ID created by the `RecordStore.startTransaction()` method, the `Record¬ Store.getLastCommittedGenerationId()` method is used to get the ID of the last-committed generation in the Record Store.

```
tId = recordStore.startTransaction(TransactionType.READ);

GenerationId gId = recordStore.getLastCommittedGenerationId(tId);
```

4. The `RecordStoreReader.createBaselineReader()` method uses the transaction ID and the generation ID to create a reader.

```
RecordStoreReader reader =
   RecordStoreReader.createBaselineReader(recordStore, tId, gId);
```

5. Within a while loop, the `RecordStoreReader` `hasNext()` and `next()` methods are used to read the sample record. The reader is closed when there are no more records to be read.

```
int count = 0;
while (reader.hasNext()) {
   Record record = reader.next();
   System.out.println("  RECORD: " + record);
   count++;
}
reader.close();
```

6. The `RecordStore.commitTransaction()` method closes the transaction.

```
recordStore.commitTransaction(tId);
```

Note: As with the writer client, if a `RecordStoreFault` exception occurs during the read process, it is caught and the `RecordStore.rollbackTransaction()` method rolls back the READ transaction.

Part 2

Using CAS data sources

- *Using the Delimited File data source*
- *Using the Endeca Record File data source*
- *Using the File System data source*
- *Using the JDBC data source*

Chapter 6

Using the Delimited File data source

You can configure the data source by using CAS Console, `cas-cmd`, or with the CAS Server API. For details on using the API, see the *Endeca CAS API Guide*.

Configuration properties for the Delimited File data source

To configure a Delimited File data source, specify values for the configuration properties listed below. If you are configuring the data source in CAS Console, the CAS Property Display Name appears in the user interface. If you are configuring the data source using a crawl configuration file or the CAS Server API, specify a value for the CAS Property Name in the configuration file or pass a value to the API.

CAS Property Display Name	CAS Property Name	Property Description
Path to Input File(s)	`inputFilePath`	(Required). Specify an absolute path to the delimited files you want to crawl. Wildcards may be used in the filename but not in the path preceding the filename.
Record Id Column	`recordIdColumn`	(Required). Specify the name of the column that you want to map to the record ID property in the generated records.
Delimiter Character	`delimiterChar`	(Required). Specify a single character that delimits the fields in the records. The default delimiter is a comma (,).
Quote Character	`quoteCharacter`	(Required). Specify a single character that escapes occurrences of the delimited character within a field. The default quote character is a quote (").
Column Names	`columnNames`	(Optional). Specify column names in the order in which they appear in a delimited text file. This optional configuration is typically only necessary in cases where a

CAS Property Display Name	CAS Property Name	Property Description
		delimited file does not contain a header row. If **Column Names** are unspecified, the data source treats the first row of the file as the header row and uses the column names as Endeca property names.
Multi-Assign Delimiter Character	`multiAssignDelimiterChar`	(Optional). Specify a single character that delimits multi-assign values within a multi-assign column. If you specify a value and omit adding any **Multi-Assign Columns**, the data source parses all columns in the file as if they may contain multi-assign values.
Multi-Assign Columns	`multiAssignColumns`	(Optional). Specify each column in the file that contains multi-assign values and name it as appropriate for the column's value.
Trim Whitespace	`trimWhitespace`	(Optional). Specify **true** to trim the leading or trailing whitespace from the data stored in columns of the delimited file. The default value is **true**.
Character Encoding	`characterEncoding`	(Optional). Specify the character encoding of the delimited file that is being crawled. If unspecified, the default value is **UTF-8**.

Note: Properties are case sensitive.

Chapter 7

Using the Endeca Record File data source

You can configure the data source by using CAS Console, `cas-cmd`, or with the CAS Server API. For details on using the API, see the *Endeca CAS API Guide*.

Configuration properties for the Endeca Record File data source

To configure an Endeca Record File data source, specify values for the configuration properties listed below. If you are configuring the data source in CAS Console, the CAS Property Display Name appears in the user interface. If you are configuring the data source using a crawl configuration file or the CAS Server API, specify a value for the CAS Property Name in the configuration file or pass a value to the API.

CAS Property Display Name	CAS Property Name	Property Description
Path to Input File(s)	`inputFiles`	(Required). Specify an absolute path to the files you want to crawl. Wildcards may be used in the filename but not in the path preceding the filename.
Record Id Property	`recordIDProperty`	(Required). Specify the name of the source property that you want to map to the record ID property in the generated records. This property must be unique across all files being crawled.

 **Note:** Properties are case sensitive.

Chapter 8

Using the File System data source

You can configure the data source by using CAS Console, `cas-cmd`, or with the CAS Server API. For details on using the API, see the *Endeca CAS API Guide*.

Configuration properties for the File System data source

To configure an Endeca Record File data source, specify values for the configuration properties listed below. If you are configuring the data source in CAS Console, the CAS Property Display Name appears in the user interface. If you are configuring the data source using a crawl configuration file or the CAS Server API, specify a value for the CAS Property Name in the configuration file or pass a value to the API.

CAS Property Display Name	CAS Property Name	Property Description
Seeds	`seeds`	(Required). Specify an absolute path to a folder you want to crawl. Seeds may be local folders or network drives. Note that for Windows, you should specify network drives by universal naming convention (UNC) syntax rather than by using the letter of a mapped drive. For UNIX, you can specify mounted or local drives using standard file path syntax.
Retrieve ACLs	`gatherNativeFileProperties`	(Optional.) Enabling this option creates ACL properties on the crawled records. This option is available to file system data sources. It is not available to CMS data sources and custom data sources.
Expand archive files	`expandArchives`	(Optional.) Enabling this option creates a record for each archived entry and populates the record's properties.

Note: Properties are case sensitive.

Chapter 9

Using the JDBC data source

You can configure the data source by using CAS Console, `cas-cmd`, or with the CAS Server API. For details on using the API, see the *Endeca CAS API Guide*.

Installing a JDBC driver into CAS

Before creating a JDBC data source, you must install the JDBC driver for the database that you want to crawl into the CAS `cas-server-plugins` directory. For example, if you are crawling an Oracle database, you install the JDBC driver for Oracle.

You must install the JDBC driver into CAS before creating the JDBC data source.

To Install a JDBC driver into CAS:

1. Stop Endeca CAS Service.
2. Navigate to `<install path>\CAS\<version>\lib\cas-server-plugins\cas-jdbc-datasource`.
3. Copy the JAR file for the JDBC driver into the `cas-jdbc-datasource` directory.
4. Start Endeca CAS Service.

Configuration properties for the JDBC data source

To configure a JDBC data source, specify values for the configuration properties listed below. If you are configuring the data source in CAS Console, the CAS Property Display Name appears in the user interface. If you are configuring the data source using a crawl configuration file or the CAS Server API, specify a value for the CAS Property Name in the configuration file or pass a value to the API.

Note: In addition to configuring the data source-specific properties listed below, you must enter values for the data source **Username** and **Password**.

CAS Property Display Name	CAS Property Name	Property Description
Driver class	`driver`	(Required). Specify the fully qualified Java class name for the JDBC driver.

CAS Property Display Name	CAS Property Name	Property Description
JDBC URL	`jdbcUrl`	(Required). Specify the connection string that includes, at a minimum, the database vendor, the host and port, and the database instance name. If desired, you can also specify the **Username** and **Password** as part of the connection string.
Connection Properties	`connectionProperties`	(Optional). Specify any additional connection properties that your database may require. Specify properties in the format: `name=value`.
SQL/Record Generation	`sql`	(Required). Specify a SQL query to execute against the database.
Key Column	`keyColumn`	(Required). Specify the name of the column in the database that you want to map to the record ID property in the generated records.

 **Note:** Properties are case sensitive.

Feature notes and known limitations of the JDBC data source

BLOBs and document conversion

If the document conversion option is enabled, any columns that contain binary data are processed by the CAS Document Conversion Module.

Record spec and Key Column values

The data type of **Key Column** property cannot be a `BLOB` (binary large object) or other type of binary object (`BINARY`, `VARBINARY`, `LONGVARBINARY`).

BLOBs

The JDBC data source supports a maximum of one column containing BLOB data per result set.

Unsupported data types

The JDBC data source does not support the following data types:

- `NULL`
- `OTHER`
- `JAVA_OBJECT`
- `DISTINCT`
- `STRUCT`
- `ARRAY`

- REF

Part 3

Loading data into an MDEX Engine

- *Creating a Forge pipeline to read from or write to a Record Store*
- *Creating a CAS crawl to write MDEX-compatible output*

Chapter 10

Creating a Forge pipeline to read from or write to a Record Store

This section describes how to build a Forge pipeline that reads Endeca records from one or more Record Store instances and that writes Endeca records to a Record Store instance.

Overview of a Forge pipeline

The CAS Server and the Endeca Web Crawler create Endeca records ready for processing by any type of Forge pipeline (baseline or partial).

The CAS Server stores the records in either a Record Store instance or in a file on disk. By default record storage is written to a Record Store instance. The Web Crawler stores records, by default, in a file on disk but can be configured to store records in a Record Store instance. (Using a Record Store instance is the recommended approach.)

To read the records into a Forge pipeline, you add an input record adapter to your Developer Studio project.

If the record adapter is reading from a CAS output file, you specify the input format of either XML or binary (depending on how you configure the output format). The URL field of the record adapter specifies the location of the CAS output file.

If the record adapter is reading from a Record Store instance, you configure the record adapter as a custom adapter.

Depending on the needs of your application, you can create these types of Forge pipelines:

- Baseline-update pipeline. This type of pipeline is intended for sites that perform only full crawls and wish to perform only baseline updates. The topics in this chapter describe this type of Forge pipeline.
- Delta-update pipeline. This type of pipeline is intended for sites that perform both full and incremental crawls and wish to perform baseline updates on both sets of data. This type of application is not documented in this guide. Instead, see the solution article "Implementing Delta Updates" available online from the Oracle Technology Network.
- Baseline-update and partial-update pipelines. These pipelines are used if the site wants to perform partial updates. This type of application is not documented in this guide. Instead, refer to the *Endeca Partial Updates Guide*.

Regardless of the type of Forge pipeline, you create it as well as perform the rest of the back-end application tasks (such as creating Endeca properties and dimensions, search interfaces, and so on) with Endeca Developer Studio.

Creating a Forge pipeline

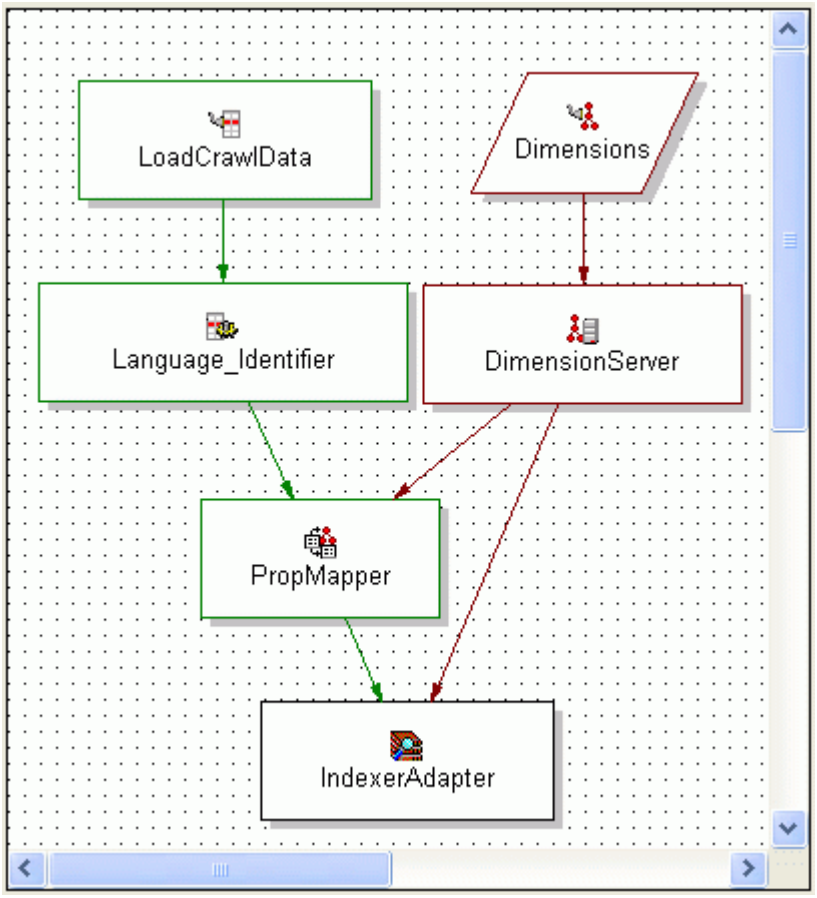
A baseline-update pipeline can be used for sites that perform only full crawls (not incremental crawls). This section describes how to create and configure a simple Forge pipeline that runs a baseline update and also configure a record adapter to read and write from record store instances.

The goal of this section is to describe the Forge pipeline components that are specific to modifying records produced by crawling. Therefore, for simplicity, components that are common to non-crawler pipelines (dimension server, property mapper, indexer adapter, and so on) are not described here.

The high-level overview of a baseline-update pipeline is as follows:

1. Create a record adapter to read the Endeca records that CAS produced (required).
2. Map the record properties to Endeca properties and dimensions (required, but not documented in this guide. See *Endeca Developer Studio Help*.).

The following illustration shows a baseline-update pipeline with the components necessary to process records produced by crawling a data source. This includes the components common to any pipeline (dimension adapter, dimension server, and indexer adapter).



Creating a record adapter to read from one or more Record Store instances

By default, the CAS writes output from a CMS, file system, or custom data source to a Record Store instance. The Web Crawler can also be configured to write output to a Record Store instance. If an application contains

multiple data sources, there are multiple Record Store instances that result. Forge can read the Endeca records from any number of Record Store instances using a custom record adapter.

You configure a custom record adapter in Developer Studio with Java properties set on the **General** tab and with several pass-through values set on the **Pass Throughs** tab.

To create a record adapter to read from one or more Record Store instances:

1. Open your project in Developer Studio.
2. In the Project tab, double-click **Pipeline Diagram**.
3. In the Pipeline Diagram editor, click **New**.
4. Select **Record** > **Adapter**.
5. In the **Name** text box, specify the name of this record adapter.
6. In the **Direction** frame, select **Input**.
7. From the **Format** list, choose **Custom Adapter**.
8. In the **Class** field of Java Properties, specify one of the following:
   - To read from one Record Store instance, specify `com.endeca.itl.recordstore.forge.RecordStoreSource`.
   - To read from multiple Record Store instances, specify `com.endeca.itl.recordstore.forge.MultipleRecordStoreSource`. This class instructs Forge to contact the Component Instance Manager, request a list of all available Record Store instances, and read from each.
9. In the **Classpath** field of Java Properties, specify the path to `<install path>/CAS/<version>/lib/recordstore-forge-adapter/recordstore-forge-adapter-11.0.0.jar`.

   Endeca recommends that you keep this JAR file in the `lib` directory because of the large number of dependencies on other JAR files in that location.
10. Select the **Pass Throughs** tab of the Record Adapter editor.
11. On the **Pass Throughs** tab, create the following name/value pairs:
    - Set a `HOST` pass-through to the fully qualified host name of the machine running the Endeca CAS Service. For example, `HOST = hostname.endeca.com`.
    - Set a `PORT` pass-through to the port number that the Endeca CAS Service is listening on. For example, `PORT = 8500`.
    - If reading from one Record Store instance, set an `INSTANCE_NAME` pass-through to the name of the Record Store instance that you want Forge to read from. For example, `INSTANCE_NAME = crawlID`. This pass-through is not required if the adapter is reading from multiple Record Store instances.
    - For a baseline pipeline, set a `READ_TYPE` pass-through to `BASELINE`. The `BASELINE` setting instructs Forge to read the latest version of all records in the Record Store. For example, `READ_TYPE = BASE¬ LINE`.

      For a partial-update pipeline, set a `READ_TYPE` pass-through to `DELTA`. The `DELTA` setting instructs Forge to read records that have been modified or added between the last committed generation in the Record Store and the last generation read by the same client as identified by `CLIENT_ID` setting. For example, `READ_TYPE = DELTA`.
    - Set a `CLIENT_ID` pass-through to a string that distinguishes this client from others that may also be reading from the Record Store instances. For example, `CLIENT_ID = FORGE`. The `CLIENT_ID` pass-through specifies the client ID to be set for the generation that is being read in. In effect, this pass-through is performing the set-last-read-generation task that can be performed with the CAS Server Command-line Utility (i.e., state is being set for the client, which is Forge in this case). This pass-through can be used only for `READ_TYPE` operations.

- Optionally, set a `RECORDS_PER_TRANSFER` pass-through to the number of records to transfer at a time for each Record Store instance. The default is `500`. Click **OK** to add the new record adapter to the project.
- Optionally, to enable SSL with server only authentication, add pass through options for the truststore location (`SSL_TRUSTSTORE`), type (`SSL_TRUSTSTORE_TYPE`), password (`SSL_TRUSTSTORE_PASSWORD`), and CAS port usage (`IS_PORT_SSL`).

  Note: A value of `true` means that `PORT` is an SSL port and the record adapter uses HTTPS for connections. A value of `false` means that `PORT` is a non-SSL port and the record adapter uses HTTPS redirects. Specify `false` if you enabled redirects from a non-SSL port to an SSL port.

  For example: `SSL_TRUSTSTORE = C:\Endeca\CAS\workspace\conf\truststore.ks`, `SSL_TRUSTSTORE_TYPE = JKS`, `SSL_TRUSTSTORE_PASSWORD = endeca`, `IS_PORT_SSL = false`.

- Optionally, to enable SSL with mutual authentication, add pass-through options for the keystore location (`SSL_KEYSTORE`), type (`SSL_KEYSTORE_TYPE`), and password (`SSL_KEYSTORE_PASSWORD`).

  For example: `SSL_KEYSTORE = C:\Endeca\CAS\workspace\conf\keystore.ks`, `SSL_KEYSTORE_TYPE = JKS`, `SSL_KEYSTORE_PASSWORD = endeca`, `IS_PORT_SSL = false`.

12. Click **OK** to add the new record adapter to the project.
13. Save the project by selecting **Save** from the File menu.

In some cases, you may get an `Out of Memory` error if Forge is reading or writing records from a Record Store instance. To work around this error, you can increase the amount of memory allocated to the JVM running Forge. To increase the memory, run Forge with `--javaArgument` flag and the `-Xmx` argument, for example `--javaArgument -Xmx512m`.

Creating a record adapter to read from crawl output files

If you configured the CAS Server or Web Crawler to write to output files (rather than a Record Store instance), the Endeca records in the file can be loaded into a pipeline using a record adapter that you create in Developer Studio.

To create a record adapter to read from crawl output files:

1. Open your project in Developer Studio.
2. In the Project tab, double-click **Pipeline Diagram**.
3. In the Pipeline Diagram editor, click **New**.
4. Select **Record** > **Adapter**. The Record Adapter editor displays.
5. In the **Name** text box, specify the name of this record adapter.
6. In the Direction frame, select **Input**.
7. From the Format drop-down list, choose either **Binary** or **XML** depending on how you configured the output records format.
8. In the **URL** text box, specify the name of the crawler output records file. Keep in mind that you can use a wildcard (*) with the name in the URL text box (such as `../output/crawlID/*.bin.gz`).

   A wildcarded URL is especially useful if your crawls produce multiple segmented Binary files.
9. Configure the remaining settings (**Encoding**, **Require data**, and so on) according to the needs of your application. Be sure to check the **Multi file** checkbox if you are using a wildcarded URL.
10. Click **OK** to add the new record adapter to the project.

11. Save the project by selecting **Save** from the File menu.

After the pipeline is complete, be sure to check the Forge log when you run the pipeline for the first time. In particular, look for file input errors similar to this example:

```
Unable to open C:\Crawls\data\incoming\*.bin.gz for input:
opening file "C:\Crawls\data\incoming\*.bin.gz":
The filename, directory name, or volume label syntax is incorrect.
```

If you see this error, recheck the setting of the **URL** text box to ensure that it points to the correct name and location of the record output files.

Creating a record adapter to write to a Record Store instance

Forge can write Endeca records to a Record Store instance using a custom output record adapter.

There must be records available in a Forge pipeline before you can instruct Forge to write to a Record Store instance. Typically, Endeca records have already been read into a Forge pipeline from one or more data sources using an input record adapter. In this scenario, the processing flow is as follows:

1. Read records from data sources using input record adapters.
2. Perform Forge pipeline processing as necessary.
3. Store the processed records in a Record Store instance using a custom output record adapter.
4. Perform additional Forge pipeline processing as necessary.

You configure a custom output record adapter with Java properties set on the **General** tab and with pass-through values set on the **Pass Throughs** tab.

To create a record adapter to write to a Record Store instance:

1. Open your project in Developer Studio.
2. In the Project tab, double-click **Pipeline Diagram**.
3. In the Pipeline Diagram editor, click **New**.
4. Select **Record** > **Adapter**.
5. In the **Name** text box, specify the name of this record adapter.
6. In the **Direction** frame, select **Output**.
7. From the **Format** list, choose **Custom Adapter**.
8. In the **Class** field of Java Properties, specify `com.endeca.itl.recordstore.forge.RecordStoreSink`.
9. In the **Classpath** field of Java Properties, specify the path to `<install path>/CAS/<version>/lib/recordstore-forge-adapter/recordstore-forge-adapter-11.0.0.jar`.

   Endeca recommends that you keep this JAR file in the `lib` directory because of the large number of dependencies on other JAR files in that location.
10. Select the **Pass Throughs** tab of the Record Adapter editor.
11. On the **Pass Throughs** tab, create the following name/value pairs:
    - Set a `HOST` pass-through to the fully qualified host name of the machine running the Endeca CAS Service. For example, `HOST = hostname.endeca.com`.
    - Set a `PORT` pass-through to port number that the Endeca CAS Service is listening on. For example, `PORT = 8500`.
    - Set an `INSTANCE_NAME` pass-through to the name of the unique Record Store instance is that you want Forge to write to. For example, `INSTANCE_NAME = crawlID`.

- For a baseline pipeline, set a `WRITE_TYPE` pass-through to `BASELINE`. For example, `WRITE_TYPE = BASELINE`.

  For a partial-update pipeline, set a `WRITE_TYPE` pass-through to `DELTA`. The `DELTA` setting instructs Forge to write records that have been modified or added. You indicate modified or added records with an `Endeca.Action` property that has a value of `UPSERT`. For example, `WRITE_TYPE = DELTA`.

- Optionally, to enable SSL with server only authentication, add pass through options for the truststore location (`SSL_TRUSTSTORE`), type (`SSL_TRUSTSTORE_TYPE`), password (`SSL_TRUSTSTORE_PASSWORD`), and CAS port usage (`IS_PORT_SSL`).

  Note: A value of `true` means that `PORT` is an SSL port and the record adapter uses HTTPS for connections. A value of `false` means that `PORT` is a non-SSL port and the record adapter uses HTTP for connections. Specify `false` if you enabled redirects from a non-SSL port to an SSL port.

  For example: `SSL_TRUSTSTORE = C:\Endeca\CAS\workspace\conf\truststore.ks`, `SSL_TRUSTSTORE_TYPE = JKS`, `SSL_TRUSTSTORE_PASSWORD = endeca`, `IS_PORT_SSL = false`.

- Optionally, to enable SSL with mutual authentication, add pass through options for the keystore location (`SSL_KEYSTORE`), type (`SSL_KEYSTORE_TYPE`), and password (`SSL_KEYSTORE_PASSWORD`).

  For example: `SSL_KEYSTORE = C:\Endeca\CAS\workspace\conf\keystore.ks`, `SSL_KEYSTORE_TYPE = JKS`, `SSL_KEYSTORE_PASSWORD = endeca`, `IS_PORT_SSL = false`.

- Optionally, set a `RECORDS_PER_TRANSFER` pass-through to the number of records to transfer at a time. Default is `500`.

12. Click **OK** to add the new record adapter to the project.
13. Save the project by selecting **Save** from the File menu.

In some cases, you may get an `Out of Memory` error if Forge is reading or writing records from a Record Store instance. To work around this error, you can increase the amount of memory allocated to the JVM running Forge. To increase the memory, run Forge with `--javaArgument` flag and the `-Xmx` argument, for example `--javaArgument -Xmx512m`.

Chapter 11

Creating a CAS crawl to write MDEX-compatible output

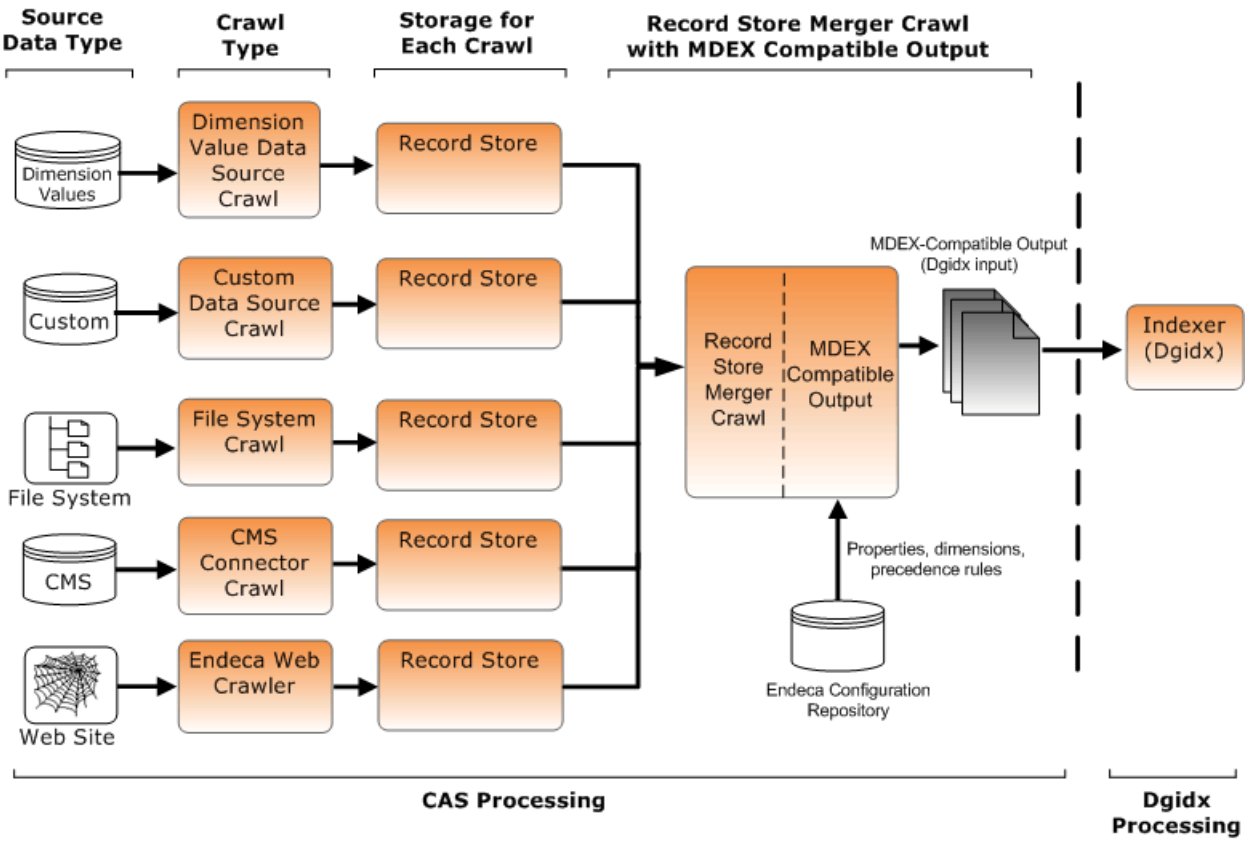
This section describes how to build a CAS crawl that crawls multiple data sources, merges records from multiple record store instances, and writes MDEX-compatible output (Dgidx input files).

Overview of a CAS crawl that produces MDEX-compatible output

You create a CAS crawl that produces MDEX-compatible output (Dgidx input files) by creating a crawl (of any type) and configuring the crawl to write MDEX-compatible output. After CAS writes the output, Dgidx can process the files, create index files, and the MDEX Engine can load the index files.

Although you can create a crawl of any data source type, the most common scenario is to create a Record Store Merger crawl that is configured to write MDEX-compatible output. The advantage of this type of crawl configuration is the capability to merge many types of data and then write output that can be directly processed by Dgidx without using Forge. For example, this data can include product inventory, product descriptions, extended product reviews, and so on.

Here is a diagram of a CAS crawl that uses a Record Store Merger crawl to read from five record store instances and then write MDEX-compatible output for use by Dgidx:



This high-level process consists of the following steps:

- Specify dimensions, properties, and precedence rules and store them in the Endeca Configuration Repository.
- Load dimension values into a Record Store instance.
- Load data records into a Record Store instance.
- Create a Record Store Merger crawl to read the dimension values and data records.
- Configure the Record Store Merger crawl to write MDEX-compatible output.

Each step is covered in topics below.

Loading dimensions, properties, and precedence rules

CAS relies on an Endeca instance configuration that you load into the Endeca Configuration Repository (ECR) before you run a crawl that produces MDEX-compatible output. In this release, the instance configuration is limited to dimensions, properties, and precedence rules.

For details about how to load the instance configuration, see the *Endeca Configuration Import API Reference (Javadoc)* that is installed with Tools and Frameworks (`ToolsAndFrameworks\<version>\config_import_api\apidoc`).

Once the instance configuration is loaded into the ECR, you can adjust, test, and make small modifications to the index configuration by using the Index Configuration Command-line Utility. For details about this utility, see "Modifying Index Configuration for an Application" in the *Deployment Template Usage Guide*.

Loading dimension value records into a Record Store instance

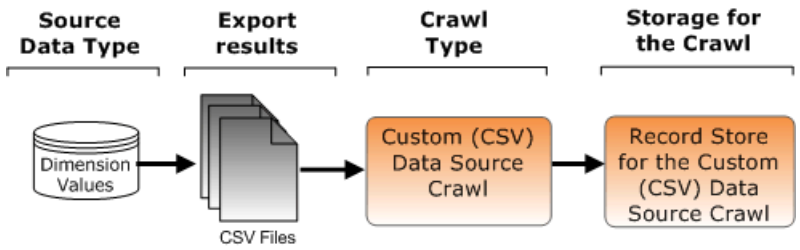
There are several common scenarios to load dimension values into a Record Store instance. In all of the scenarios, the goal is the same—to crawl hierarchy information (the dimension values) and then populate a Record Store instance with Endeca records that represent the dimension values.

Scenario 1 - Crawling extracts of the source data

This scenario consists of the following steps:

1. Export hierarchy information from the source data to a common output format, for example, a CSV file. Each row in a CSV file typically corresponds to one node in a hierarchy.
2. Add columns to the CSV file with Endeca properties that map each node to a corresponding Endeca dimension value. For details, see *Record properties for all dimension values* on page 88.
3. Implement a custom data source that can crawl the exported source data. You can implement your own data sources using the CAS Extension API. For details, see the *Endeca CAS Extension API Guide*.
4. Crawl the exported source data (i.e. the CSV file). CAS creates one Endeca record for each dimension value and by default, writes the records to a Record Store instance.

Here is a diagram of this example scenario:

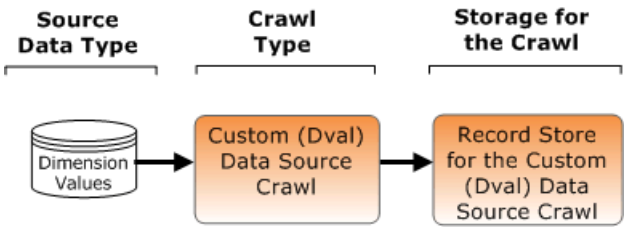


Scenario 2 - Crawling the source data

This scenario consists of the following steps:

1. Add Endeca properties to the source data that map each node in the source data to a corresponding dimension value. For details, see *Record properties for all dimension values* on page 88.
2. Implement a custom data source that can crawl the source data. You can implement your own data sources and manipulators using the CAS Extension API. For details, see the *Endeca CAS Extension API Guide*.
3. Crawl the source data. CAS creates one Endeca record for each dimension value and by default, writes the records to a Record Store instance.

Here is a diagram of this example scenario:



Scenario 3 - Programmatically writing to a Record Store instance

In this scenario, you programmatically create Endeca records that define dimension values and write the records to a Record Store instance using the Record Store API. For details about these interfaces see the *Endeca CAS API Guide*.

Record properties for all dimension values

CAS produces one dimension value for each Endeca record that CAS crawls. Each Endeca record may have record properties listed in the table below to specify information about a dimension value. The properties describe the name, specification, parent relationship, display order, and any synonyms for the dimension value.

Property Name	Property Value
`dimval.spec`	Required. The unique Id of a dimension value. The `dimval.spec` must be unique within a dimension.
`dimval.dimension_name`	Required. The name of the dimension that the dimension value belongs to.
`dimval.display_order`	Optional. A numeric value that specifies the display order of a dimension value relative to its sibling dimension values when returned as refinements. Dimension values with lower values display before dimension values with higher values. Valid values are integers between 0 and 2147483647. If this property is unspecified, the dimension value displays lower than dimension values with specified display orders. **Note:** The `dimval.display_order` property sets display order for dimension values. You can set the display order of dimensions, relative to other dimensions, using the Endeca Configuration Import API or the Endeca Index Configuration Command-line Utility. For details about the API, see, *Endeca Configuration Import API Reference (Javadoc)*. For details about the utility, see the *Deployment Template Usage Guide*.
`dimval.parent_spec`	Required. The unique Id of a parent dimension value. Root dimension values should have a value of forward slash (/).
`dimval.display_name`	Optional. The name of the dimension value indicated by `dim¬val.spec`.
`dimval.match.use_spec`	Optional. A Boolean value that specifies whether property matching is based on `dimval.spec`. A value of `true` uses a record's `dim¬val.spec` to match dimension values, `false` does not use `dim¬val.spec`. If a record contains range value properties, the default value is `false`. If not, the default is `true`. For guidance about configuring this property, see *About dimension value matching* on page 90.
`dimval.search_synonym`	Optional. A synonym for the dimension value's display name. Synonyms provide alternative ways of describing and consequently, searching a particular dimension.

Record properties for range dimension values

You can create a range dimension value by adding the properties listed below to an Endeca record. The properties specific to range dimensions describe the lower bound, upper bound, and whether the bounds are inclusive, for a range dimension value.

Property Name	Property Value
`dimval.range.lower_bound`	Optional. A string value that specifies the lower boundary constraint for a range.
`dimval.range.lower_bound_inclusive`	Optional. A Boolean value that specifies whether `dimval.range.lower_bound` is included in the range. If unspecified, then `dimval.range.lower_bound` is exclusive.
`dimval.range.upper_bound`	Optional. A string value that specifies the upper boundary constraint for a range.
`dimval.range.upper_bound_inclusive`	Optional. A Boolean value that specifies whether `dimval.range.upper_bound` is included in the range. If unspecified, then `dimval.range.upper_bound` is exclusive.

Example of price range dimension values

The following table lists the properties required to create a set of range dimension values for a Price dimension. Each row represents one Endeca record for a dimension value.

`dimval.spec`	`dimval.dimension_name`	`dimval.parent_spec`	`dimval.range.lower_bound` (inclusive)	`dimval.range.upper_bound` (inclusive)
Under 25	product.price_range	/	0 (true)	25 (false)
25 - 50	product.price_range	/	25 (true)	50 (false)
50 - 100	product.price_range	/	50 (true)	100 (false)
100 - 250	product.price_range	/	100 (true)	250 (false)
Over 250	product.price_range	/	250 (true)	(empty string) (false)

In addition to these properties, typical range dimension values also have properties for `dimval.display_name` and perhaps properties for inclusive and exclusive range boundaries.

About automatically generating dimension values

CAS can automatically generate dimension values for a dimension based on property values in the data records. This is useful if you want to automatically populate a flat dimension with a large number of dimension values.

For example, in the Discover Electronics application, the Brand Name dimension has its dimension values automatically generated. If a data record contains a `product.brand.name` property, then CAS automatically generates a dimension value based on that property. This feature dramatically reduces the amount of time it takes to create dimension values. However, automatically generated dimensions are always flat.

To automatically generate dimension values in your application, set the `isAutoGen` property to `true` for a dimension and then run a full crawl configured to produce MDEX-compatible output. The Dimension Value Id Manager generates the dimension values (names and IDs) and stores them for each automatically generated

dimension. If you later modify the `isAutoGen` property to `false` and run a full crawl, this state is deleted. CAS does not process changes to the `isAutoGen` property in incremental crawls.

Note: Automatically generated dimension values are still stored by CAS even if the properties on which they are based are removed from the data records. (This is to ensure that content spotlights do not fail if they depend on the generated dimension values.) The dimension values are only deleted when the `isAutoGen` property is set to `false` or the dimension itself is deleted.

You cannot manually create dimension values for a dimension and also automatically generate dimension values. This situation results in an error during a crawl.

You set the `isAutoGen` property to `true` using either the Endeca Configuration Import API or the Endeca Index Configuration Command-line Utility. For details on the utility, see the *Deployment Template Usage Guide*.

About dimension value matching

The value of `dimval.match.use_spec` determines how CAS performs property to dimension value matching. As mentioned earlier, setting `dimval.match.use_spec` to `true` instructs CAS to use a record's `dimval.spec` for matching and `false` does not use the record's `dimval.spec`.

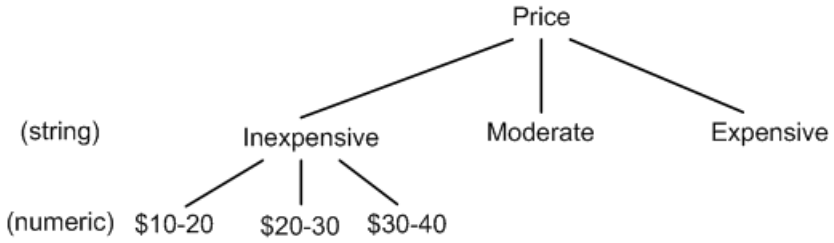
No matter how `dimval.match.use_spec` is set, CAS first attempts to match properties to dimension values using an exact matching approach. If an exact match fails, CAS attempts to match properties into range boundaries if they have been defined.

Range dimension values

These default settings work appropriately where all range dimension values in a hierarchy are numeric and the property values in the data records are numeric.

However, there may be infrequent cases where setting it to `true` is necessary if you want CAS to match a `dimval.spec` against a string property that may appear in a hierarchy that contains dimension values that are strings and dimension values that are numeric ranges.

For example, suppose you have the following Price dimension where the first refinement level is made of dimension values that are strings and the next refinement level is made up of dimension values that are numeric ranges:



If a data record has a price property with a value of 15, you want that record to match to the range dimension value of $10-20. In this case, you specify `dimval.match.use_spec` to `true` so that CAS first tries to match 15 against the non-range dimension values. CAS fails to match a numeric value of "15" against the "Inexpensive" dimension value, but succeeds in matching it against the $10-20 range dimension value.

Length limitations on name and spec values

The name and specification record properties have a length limitation of 65,536 characters. In particular, this limitation applies to: `dimval.spec`, `dimval.parent_spec`, and `dimval.dimension_name`.

The Dimension Value Id Manager throws validation errors if it processes names or spec property values that exceed the 65,536 character limit, and the crawl itself will fail.

Loading data records into a Record Store instance

Loading records that represent dimension values is the same as loading records that represent product data. You can crawl the records directly, you can crawl an export of the records, or you can programmatically write records to a Record Store instance. For data records, the content you load could be product inventory, product descriptions, enterprise data, and so on, rather than dimension value data.

Property mapping for data records

During a crawl that produces MDEX-compatible output, CAS maps values from source properties on data records to Endeca properties or dimensions on MDEX records. CAS uses the source property values to populate the Endeca properties or dimensions.

Property mappings can be either explicit or implicit.

You can create an *explicit* mapping by specifying the source property and the target Endeca property or dimension. In general, Oracle recommends that you explicitly map properties. You do this using either the Endeca Configuration Import API or the Endeca Index Configuration Command-line Utility. For details, see the *Endeca Configuration Import API Reference (Javadoc)* and also "Modifying Index Configuration for an Application" in the *Deployment Template Usage Guide*.

If you do not explicitly map properties, CAS creates an *implicit* mapping from source properties to one of three possible targets:

- A source property can map to a dimension value. In this case, the source property name must match the target dimension name.
- A source property can map to an Endeca property value. In this case, the source property name must match the target Endeca property name.
- A source property can be dropped from matching with any target property. This occurs when a source property name does not match with a target property or dimension value. For example, if `Endeca.Action` is not defined in your instance configuration, the property is omitted in MDEX records.

Dimension values and data records

If you add a new dimension value to your application, ensure that you add a record property to each data record that should be tagged with that dimension value.

For example, suppose the dimension values in an application describe product categories like `components` such as `motherboards`, `RAM`, and so on. You add a new dimension value for `software`. You also add a record property to each software product record that belongs with the `software` dimension value.

About configuring application features in a CAS-based application

If you are using CAS to process your data, rather than Forge, there is a subset of application features that you must configure manually in the instance configuration XML files for an application. You cannot use Oracle Endeca Workbench or Developer Studio to configure these particular features.

The following table provides a mapping between application features and the corresponding XML file where you configure each feature. The instance configuration files are stored in `<app dir>\<app name>\config\mdex`.

Application feature	File Name	Description
Dimension search	`dimsearch_config.xml`	The `dimsearch_config.xml` file controls how dimension searches behave. This file configures search matching, spelling correction, filtering, and relevance ranking for dimension search.
Dimensions	`dval_refs.xml`	The `dval_refs.xml` file allows you to specify a dimension value's ID, whether it is navigable or not, and whether it is collapsible or not.
Property and dimension annotation	`key_props.xml`	The `key_props.xml` file allows property and dimension keys to be annotated with metadata key/value pairs called key properties (since they are properties of a dimension or property key).
Source data / I18N	`languages.xml`	The `languages.xml` file specifies language identification codes for values in named dimensions or properties. This identification allows a single MDEX Engine to contain records in multiple languages.
Front end	`record_sort_config.xml`	The `record_sort_config.xml` file specifies properties or dimensions (attributes) that have precomputed sort indexes.
Partial updates	`record_spec.xml`	The `record_spec.xml` file specifies a property to identify records.
Record search and dimension search	`recsearch_config.xml`	The `recsearch_config.xml` configures record search and also Did You Mean and automatic spelling correction options for dimension search.
Relevance ranking	`relrank_strategies.xml`	The `relrank_strategies.xml` file contains the relevance ranking strategies for an application.
Search characters	`search_chars.xml`	The `search_chars.xml` file contains searchable characters for your application. The MDEX Engine treats such characters as searchable rather than treating them as punctuation.
Spelling correction	`spell_config.xml`	The `spell_config.xml` file specifies which spelling correction dictionary, or combination of dictionaries, and which languages the MDEX

Application feature	File Name	Description
		Engines uses to provide spelling correct suggestions.
Stemming	`stemming.xml`	The `stemming.xml` file contains an element for each language that is enabled for stemming. The stemming feature allows the system to consider alternate forms of individual words as equivalent for the purpose of search query matching.
Stop words	`stop_words.xml`	The `stop_words.xml` file contains words that should be eliminated from a query before it is processed by the MDEX Engine.

For details about how to configure these files, see the *Platform Services XML Reference*.

Creating a crawl to write MDEX-compatible output

You can configure any crawl to write MDEX-compatible output; however, the most common scenario is to create a Record Store Merger crawl to write MDEX-compatible output. When the crawl runs in full-crawl mode, it does the following high-level operations:

- Merges index configuration from multiple owners, if applicable.
- Processes the dimensions, properties, precedence rules, and dimension value records.
- Processes data records.
- Writes the configuration and records into MDEX-compatible output.

Note: A crawl configured to produce MDEX-compatible output does not write dimension value records or any configuration stored in the Endeca Configuration Repository (such as properties and dimensions, precedence rules, and so on) when running in incremental mode. Run a full crawl if you want to create MDEX-compatible output that includes dimension value records and configuration stored in the Endeca Configuration Repository.

Part 4

CAS Command Line Utilities

- *CAS Server Command-line Utility*
- *Component Instance Manager Command-line Utility*
- *Record Store Command-line Utility*

Chapter 12

CAS Server Command-line Utility

This section describes how to run the tasks of the CAS Server Command-line Utility.

Overview of the CAS Server Command-line Utility

The CAS Server Command-line Utility creates and manages data source crawls.

The CAS Server Command-line Utility is a script named `cas-cmd.sh` (for Linux/UNIX systems) and `cas-cmd.bat` (for Windows) that you run from a command prompt. The scripts are in the `bin` directory.

Help options

The CAS Server Command-line Utility has two help options that display the usage syntax. The `--help` option displays a summary of the tasks. The `--help-detail` option displays detailed usage information for all the tasks. For example:

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd.bat --help
usage: cas-cmd <task-name> [options]
[Inspecting Installed Modules]
  getAllModuleSpecs
  getModuleSpec
  listModules
[Managing Crawls]
  createCrawls
  deleteCrawl
  getAllCrawls
  getCrawl
  getCrawlIncrementalSupport
  listCrawls
  startCrawl
  stopCrawl
  updateCrawls
[Managing Dimension Value Ids]
  createDimensionValueIdManager
  deleteDimensionValueIdManager
  exportDimensionValueIdMappings
  generateDimensionValueId
  getDimensionValueId
  importDimensionValueIdMappings
[Viewing Crawl Status and Results]
  getAllCrawlMetrics
  getCrawlMetrics
```

```
  getCrawlStatus

For detailed usage information including task options, use --help-detail
For detailed usage information for individual task options, use <task-name>
--help
```

Command-line options

The command syntax for executing the tasks is:

```
cas-cmd task-name [options]
```

The *task-name* argument is the task to be performed by the utility, such as the `createCrawls` task. The task options vary, depending on the task. However, these options can be used with any task:

- `-h` (or `--host`) specifies the host name of the machine on which the CAS Service is running. If the flag is omitted, it defaults to the value of the `com.endeca.itl.cas.server.host` property in `<install path>\CAS\workspace\conf\commandline.properties`. If the property is not set, the value then defaults to `localhost` as the host name.
- `-p` (or `--port`) specifies the port on which CAS Service is listening. If the flag is omitted, it defaults to the value of the `com.endeca.itl.cas.server.port` property in `workspace\CAS\conf\commandline.properties`. If the property is not set, the value then defaults to `8500` as the port number.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

Host and port settings

You first specify the host and port settings for the Endeca CAS Service as part of the installation wizard. That host runs the CAS Server, the Component Instance Manager, and all Record Store instances. The installation wizard then writes the host (`com.endeca.itl.cas.server.host`) and port (`com.endeca.itl.cas.server.port`) settings as properties in `commandline.properties`. All of the CAS command-line utilities use these settings as default values if you omit the `-h` and `-p` flags when executing any tasks.

Setting the bin directory in the PATH environment variable

Although not required, it is recommended that you set the path of the `bin` directory in your system's PATH environment variable. This allows you to run the CAS Server Command-line Utility script from any location.

About error handling

- If desired, you can re-configure the default logging settings in `<install path>\CAS\workspace\conf\cas-cmd.log4j.properties`.
- Errors print to standard error, unless you redirect `stderr` to a file instead.
- Errors of mis-configured command-line tasks or incorrect input parameters are written to standard out.

About CAS capabilities

The Content Acquisition System provides a list of capabilities that describe whether a data source or manipulator supports an optional set of CAS features. For example, if a data source or manipulator has the `Supports Incrementals` capability, then it can run in an incremental crawl.

You get the capabilities for a data source or manipulator by running the `listModules` task or the `getModuleSpec` task of `cas-cmd`.

The list of CAS capabilities available to a data source or manipulator includes the following:

- `Binary Content Accessible via FileSystem` - Indicates that the data source supports local caching for files accessible from a file system. This capability does not apply to manipulators.
- `Data Source Filter` - Indicates that the data source supports filter configuration. This capability does not apply to manipulators.
- `Has Binary Content` - Indicates that the data source supports document conversion. This capability does not apply to manipulators.
- `Expand Archives` - indicates that the data source supports archive expansion as part of a crawl. This capability does not apply to manipulators.
- `Supports Incrementals` - Indicates that the manipulator can run as part of an incremental crawl. This capability does not apply to data sources.

Saving passwords in a crawl configuration file

Although data sources and manipulators can be configured with passwords, their crawl configurations are retrieved by the `getCrawl` or `getAllCrawls` tasks without passwords.

There are two ways to specify a password for a data source or a manipulator:

- You can specify a password when prompted by the `createCrawls` task of `cas-cmd`.
- You can save the password in a crawl configuration file.

Note: The `updateCrawls` task of `cas-cmd` does not prompt for a password because CAS Server stores the password during the create process, and the `updateCrawls` task uses the stored password.

Recall that passwords are indicated in Endeca CMS data sources with the hardcoded `password` configuration property. However, in cases where a plug-in developer creates a data source or manipulator with a password configuration property, the property may have any name the plug-in developer chooses. (In this situation, the plug-in developer specifies a password configuration property by adding the `isPassword=true` attribute in the property's annotation.)

To save a password in a crawl configuration:

1. In a text editor, open the crawl configuration file and locate the `<configuration>` element for the given crawl and within `<configuration>` locate the `<sourceConfig>` element.
2. Within `<sourceConfig>`, locate the `<moduleProperty>` element that specifies the password configuration property.
   - For Endeca CMS data sources, this is the `<moduleProperty>` with `<key>password</key>`.
   - For data sources or manipulators created by a plug-in developer, you can locate the password configuration property by running the `getModuleSpec` and looking for the property that has `*Password: true`.

   For example:

```
<moduleProperty>
   <key>password</key>
</moduleProperty>
```

3. Directly below the `<key> ...</key>` line, enter `<value>` followed by a value you wish to set as the password, and then the closing `</value>` tag.

For example

```
<moduleProperty>
   <key>password</key>
   <value>p@ssw0rd</value>
</moduleProperty>
```

4. Save and close the configuration file.
5. Specify this configuration file with the `-f` option of the `createCrawls` task.

Inspecting installed modules

The following `cas-cmd` tasks return information about the modules you have installed.

Getting the specifications of all modules

The `getAllModuleSpecs` task retrieves all module specifications. A module specification includes the configuration properties, capabilities, and `moduleInfo` of a particular module.

The syntax for this task is:

```
cas-cmd getAllModuleSpecs [-h HostName] [-l true|false] [-p PortNumber] [-t Mod¬
uleType]
```

Where:

- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-t` (or `--type`) specifies the type of module to list. If unspecified, the task returns the specifications of all modules. A value of `SOURCE` returns the specifications of all data sources. A value of `MANIPULATOR` returns the specifications of all manipulators. Optional.

To get the specifications of all modules:

1. Start a command prompt and navigate to *<install path>*`\CAS\<version>\bin` (for Windows), or *<install path>*`/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `getAllModuleSpecs` task.

Note: This task name is case sensitive.

Getting the specification of a module

The `getModuleSpec` task retrieves the specification of a particular module. A module specification includes the configuration properties, capabilities, and `moduleInfo` of a particular module.

The syntax for this task is:

```
cas-cmd getModuleSpec -id ModuleId [-h HostName]
[-p PortNumber] [-l true|false]
```

Where:

- `-id` (or `--module_id`) specifies the ID of a module that you have installed into CAS.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

If necessary you can first run the `listModules` task to list the modules that you have installed.

To get the specification of a module:

1. Start a command prompt and navigate to *<install path>*`\CAS\<version>\bin` (for Windows), or *<install path>*`/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `getModuleSpec` task with the id of the module for which to retrieve the specification.

Note: This task name is case sensitive.

Example of getting the specification of a module

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd getModuleSpec -id "File System"
File System
===========
[Module Information]
 *Id: File System
 *Type: SOURCE
 *Description: No description available for File System
 *Capabilities:
   *Binary Content Accessible via FileSystem
   *Data Source Filter
   *Has Binary Content
   *Expand Archives

[File System Configuration Properties]
Group: Seeds
------------
 *Help Link: /casconsole-infocenter/index.jsp?topic=/com.endeca.itl.docset.cas-
c
onsole-eclipse-help/src/tcasc_adding_a_new_fs_ds.html
Seeds:
 *Name: seeds
```

```
 *Type: {http://www.w3.org/2001/XMLSchema}string
 *Required: true
 *Max Length: 255
 *Multiple Values: true
 *Multiple Lines: false
 *Password: false
 *Always Show: true

Group:
-------
Gather Native File Properties:
 *Name: gatherNativeFileProperties
 *Type: {http://www.w3.org/2001/XMLSchema}boolean
 *Required: false
 *Description: Gather Native File Properties
 *Multiple Values: false
 *Multiple Lines: false
 *Password: false
 *Always Show: false

Expand Archives:
 *Name: expandArchives
 *Type: {http://www.w3.org/2001/XMLSchema}boolean
 *Required: false
 *Description: Expand Archives
 *Multiple Values: false
 *Multiple Lines: false
 *Password: false
 *Always Show: false
```

Listing modules

The `listModules` task lists modules you can include in a crawl. Modules include CMS data sources that you have licensed and enabled and any other data sources and manipulators.

The syntax for this task is:

```
cas-cmd listModules [-t ModuleType] [-h HostName] [-p PortNumber] [-l true|false]
```

Where:

- `-t` (or `--type`) specifies the type of module to list. If unspecified, the task returns all modules. A value of `SOURCE` returns a list of all data sources enabled on the CAS Server. A value of `MANIPULATOR` returns a list of all manipulators installed on the CAS Server. Optional.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

To list modules:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin` (for Windows), or `<install path>/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `listModules` task.

Note: This task name is case sensitive.

Managing crawls

The following `cas-cmd` tasks manage crawls.

Creating crawls

The `createCrawls` task creates and stores named crawls.

The syntax for this task is:

```
cas-cmd createCrawls -f CrawlConfig.xml [-h HostName] [-p PortNumber] [-l
true|false]
```

Where:

- `-f` (or `--file_name`) specifies the pathname of the input XML file containing the crawl configuration(s). Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`, in `<install path>\CAS\workspace\conf` (on Windows) or `<install path>/CAS/workspace/conf` (on UNIX). If the property is not set, the value then defaults to `8500`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

Note:

- If you are running `createCrawls` as part of migrating from a previous version of CAS to the current version, the `createCrawls` task handles updating all aspects of the crawl configuration file.
- If conflicts arise when running the `createCrawls` task (such as multiple crawl configurations occuring with the same `crawlId`), the utility prompts you to either ignore the listed conflicts and continue creating the rest of the crawls, or to abort the task. If a crawl cannot be created, the CAS Server logs an error and ignores that crawl.
- When the CAS Server Command-line Utility loads a crawl configuration that contains an empty password property, the user is prompted for a password. If a password is entered incorrectly, the crawl is not saved.
- You may add a password to the crawl configuration and update CAS Server with this modified configuration. Or, you may enter the password when prompted by running the task. The password is saved only on the server running the Endeca CAS Service.

To create crawls:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin` (for Windows), or `<install path>/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify `createCrawls` with the `-f` or `--file_name` flag, and the absolute path to the crawl configuration file.

Note: This task name is case sensitive.

Example of creating crawls

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd createCrawls -f C:\tmp\fileCrawlConfig.xml
Created crawl FileCrawl
```

Related Links

About creating a crawl on page 21
You can use CAS Console, the CAS Server Command-line Utility, and CAS Server API to create and configure any number of crawls in your application. If you use CAS Console, note that a *crawl* is equivalent to a *data source* in the CAS Console user interface.

Sample configuration for a Record Store Merger data source on page 171
This sample shows a crawl configuration of a Record Store Merger data source.

Sample configuration for writing output to an MDEX compatible format on page 176
This sample shows a crawl configuration that writes output to an MDEX compatible format.

Deleting a crawl

The `deleteCrawl` task deletes a crawl.

The syntax for this task is:

```
cas-cmd deleteCrawl -id CrawlName [-h HostName] [-p PortNumber] [-l true|false]
```

Where:

- `-id` (or `--crawl_id`) specifies the name of the crawl to be deleted. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin` (for Windows), or `<install path>/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `deleteCrawl` task with the id of the crawl to be deleted.

Note: This task name is case sensitive.

Example of deleting a crawl

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd deleteCrawl -id FileCrawl
```

Getting all crawls

The `getAllCrawls` task retrieves all crawl configurations.

The syntax for this task is:

```
cas-cmd getAllCrawls [-f FileName.xml] [-h HostName] [-p PortNumber]
[-d] [-l true|false]
```

Where:

- `-f` (or `--file_name`) specifies the name of the XML file to write the configuration to. If omitted, the crawl configuration is sent to standard output. Optional.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-d` (or `--fill_in_defaults`) specifies whether to populate the configuration file with the default values for unspecified properties. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

Crawls are retrieved without password values if there are any configuration properties marked as `isPassword`.

To get all crawls:

1. Start a command prompt and navigate to *<install path>*\CAS\<version>\bin (for Windows), or *<install path>*/CAS/<version>/bin (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `getAllCrawls` task, optionally with the `-f` (or `--file`) flag and the name of the XML file to write the crawl configuration(s) to.

Note: This task name is case sensitive.

Example of getting all crawls

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd getAllCrawls
<?xml version="1.0" encoding="UTF-8"?>

<configurations xmlns="http://endeca.com/itl/cas/2011-12">
  <crawlConfig>
    <crawlId>
      <id>FileCrawl</id>
    </crawlId>
    <unavailableIncrementalSwitchesToFullCrawl>true</unavailableIncremental¬
SwitchesToFullCrawl>
    <sourceConfig>
      <moduleId>
        <id>File System</id>
```

```
      </moduleId>
      <moduleProperties>
        <moduleProperty>
          <key>expandArchives</key>
          <value>false</value>
        </moduleProperty>
        <moduleProperty>
          <key>gatherNativeFileProperties</key>
          <value>true</value>
        </moduleProperty>
        <moduleProperty>
          <key>seeds</key>
          <value>C:\tmp\itldocset</value>
          <value>C:\tmp\iapdocset</value>
        </moduleProperty>
      </moduleProperties>
      <excludeFilters/>
      <includeFilters/>
    </sourceConfig>
    <textExtractionConfig>
      <enabled>true</enabled>
      <makeLocalCopy>false</makeLocalCopy>
    </textExtractionConfig>
    <manipulatorConfigs/>
    <outputConfig>
      <moduleId>
        <id>Record Store</id>
      </moduleId>
      <moduleProperties/>
    </outputConfig>
  </crawlConfig>
  <crawlConfig>
    <crawlId>
      <id>SecondFileCrawl</id>
    </crawlId>
    <sourceConfig>
      <moduleId>
        <id>File System</id>
      </moduleId>
      <moduleProperties>
        <moduleProperty>
          <key>expandArchives</key>
          <value>false</value>
        </moduleProperty>
        <moduleProperty>
          <key>gatherNativeFileProperties</key>
          <value>true</value>
        </moduleProperty>
        <moduleProperty>
          <key>seeds</key>
          <value>C:\tmp\mdexdocset</value>
        </moduleProperty>
      </moduleProperties>
      <excludeFilters/>
      <includeFilters/>
    </sourceConfig>
    <textExtractionConfig>
      <enabled>true</enabled>
      <makeLocalCopy>false</makeLocalCopy>
    </textExtractionConfig>
```

```
    </manipulatorConfigs>
    <outputConfig>
      <moduleId>
        <id>Record Store</id>
      </moduleId>
      <moduleProperties/>
    </outputConfig>
  </crawlConfig>
</configurations>
```

Getting a crawl

The `getCrawl` task retrieves a single crawl configuration.

The syntax for this task is:

```
cas-cmd getCrawl -id CrawlName [-f FileName.xml] [-h HostName]
[-p PortNumber] [-d] [-l true|false]
```

Where:

- `-id` (or `--crawl_id`) specifies the name of the crawl for which you want to retrieve the crawl configuration. Required.
- `-f` (or `--file_name`) specifies the XML output file to which you want to write the crawl configuration. Optional.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-d` (or `--fill_in_defaults`) specifies whether to populate the configuration file with the default values for unspecified properties. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

The XML input and output files resulting from the `getAllCrawls` and `createCrawls` operations are similar to those from `getCrawl`, except that `getAllCrawls` returns a series of `<crawlConfig>` elements because it pertains to multiple crawls.

Crawls are retrieved without password values if there are any configuration properties marked as `isPassword`.

To get a crawl:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin` (for Windows), or `<install path>/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `getCrawl` task with the id of the crawl for which to retrieve the configuration.

Note: This task name is case sensitive.

Example of getting a crawl

```
C:\Endeca\CAS\3.1.2\bin>cas-cmd getCrawl -id FileCrawl
<?xml version="1.0" encoding="UTF-8"?>

<configurations xmlns="http://endeca.com/itl/cas/2011-12">
  <crawlConfig>
    <crawlId>
      <id>FileCrawl</id>
    </crawlId>
    <unavailableIncrementalSwitchesToFullCrawl>true</unavailableIncremental¬
SwitchesToFullCrawl>
    <sourceConfig>
      <moduleId>
        <id>File System</id>
      </moduleId>
      <moduleProperties>
        <moduleProperty>
          <key>expandArchives</key>
          <value>false</value>
        </moduleProperty>
        <moduleProperty>
          <key>gatherNativeFileProperties</key>
          <value>true</value>
        </moduleProperty>
        <moduleProperty>
          <key>seeds</key>
          <value>C:\tmp\itldocset</value>
          <value>C:\tmp\iapdocset</value>
        </moduleProperty>
      </moduleProperties>
      <excludeFilters/>
      <includeFilters/>
    </sourceConfig>
    <textExtractionConfig>
      <enabled>true</enabled>
      <makeLocalCopy>false</makeLocalCopy>
    </textExtractionConfig>
    <manipulatorConfigs/>
    <outputConfig>
      <moduleId>
        <id>Record Store</id>
      </moduleId>
      <moduleProperties/>
    </outputConfig>
  </crawlConfig>
</configurations>
```

Getting the incremental support status of a crawl

The `getCrawlIncrementalSupport` task indicates whether a specified crawl configuration supports incremental crawling and also indicates which manipulators within the crawl configuration do not support incremental crawling.

The syntax for this task is:

```
cas-cmd getCrawlIncrementalSupport [-h HostName] -id CrawlName [-l true|false]
[-p PortNumber]
```

Where:

- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-id` (or `--crawl_id`) specifies the name of the crawl to retrieve incremental support status for. Required.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.

To get the incremental support status of an incremental crawl:

1. Start a command prompt and navigate to *<install path>*`\CAS\<version>\bin` (for Windows), or *<install path>*`/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `getCrawlIncrementalSupport` task with the id of the crawl.

Note: This task name is case sensitive.

Example of getting the support status of an incremental crawl

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd getCrawlIncrementalSupport -id Test
Incrementals Supported: yes
```

Listing crawls

The `listCrawls` task lists all crawls in the Endeca CAS Service.

The syntax for this task is:

```
cas-cmd listCrawls [-h HostName] [-p PortNumber] [-l true|false]
```

Where:

- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

To list crawls:

1. Start a command prompt and navigate to *<install path>*`\CAS\<version>\bin` (for Windows), or *<install path>*`/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `listCrawls` task.

 **Note:** This task name is case sensitive.

Example of listing crawls

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd listCrawls
FileCrawl
FileCrawl2
```

Starting acquisition from a data source

The `startCrawl` task starts acquisition from a data source.

When you acquire from a data source, the CAS Server automatically determines which acquisition mode is necessary. By default, the CAS Server attempts incremental acquisition, and it switches to full acquisition if any of the following conditions are true:

- A data source has not been acquired before, which means no crawl history exists.
- A Record Store instance that stores record output does not contain at least one record generation. This applies to the default case in which the CAS Server is configured to output to a Record Store instance rather than a file on disk.
- Seeds have been removed from the data source configuration (adding seeds does not require full acquisition).
- The document conversion setting has changed.
- Filters have been added, modified, or removed in the data source configuration.
- Repository properties have changed, such as the `username` property setting for CMS data sources.

In all other cases, the CAS Server acquires incrementally. However, you may force full acquisition of a data source by specifying the `-full` option.

The syntax for this task is:

```
cas-cmd startCrawl -id CrawlName [-full] [-h HostName]
[-p PortNumber] [-l true|false]
```

Where:

- `-full` (or `--full_crawl`) specifies whether to force a full crawl. If unspecified, CAS Server runs an incremental crawl. Optional
- `-id` (or `--crawl_id`) specifies the ID of the acquisition to start. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

To start acquisition from a data source:

1. Start a command prompt and navigate to *<install path>*`\CAS\<version>\bin` (for Windows), or *<install path>*`/CAS/<version>/bin` (for UNIX).

2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `startCrawl` task with the required arguments.

Note: This task name is case sensitive.

Example of starting data acquisition from a data source

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd startCrawl -id FileCrawl
```

Stopping acquisition from a data source

The `stopCrawl` task stops acquisition from a data source.

The syntax for this task is:

```
cas-cmd stopCrawl -id CrawlName [-h HostName] [-p PortNumber] [-l true|false]
```

Where:

- `-id` (or `--crawl_id`) specifies the ID of the acquisition to stop. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

1. Start a command prompt and navigate to *<install path>*`\CAS\<version>\bin` (for Windows), or *<install path>*`/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `stopCrawl` task with the proper arguments.

Note: This task name is case sensitive.

Example of stopping acquisition from a data source

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd stopCrawl -id FileCrawl
```

Updating crawls

The `updateCrawls` task updates one or more existing crawl configurations with a new crawl configuration. The task does not create new crawl configurations. It updates existing crawl configurations with changes.

The syntax for this task is:

```
cas-cmd updateCrawls -f CrawlConfig.xml [-h HostName] [-p PortNumber] [-l
true|false]
```

Where:

- `-f` (or `--file`) specifies the pathname of the input XML file containing the crawl configuration(s). Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`, in `CAS\workspace\conf` (on Windows) or `CAS/workspace/conf` (on UNIX). If the property is not set, the value then defaults to `8500`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

Note: This task does not create a new crawl. The task throws an exception if you attempt to update a crawl that does not already exist.

To update crawls:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin` (for Windows), or `<install path>/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify `updateCrawls` with the `-f` or `--file_name` flag, and the absolute path to the crawl configuration file.

Note: This task name is case sensitive.

Example of updating crawls

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd updateCrawls -f C:\tmp\newCrawlConfig.xml
Updated crawl FileCrawl
```

Managing dimension value Ids

The following `cas-cmd` tasks manage dimension value Ids.

Creating a Dimension Value Id Manager

The `createDimensionValueIdManager` task creates a new instance of a Dimension Value Id Manager. In cases where you are not working with the Discover Electronics sample application, you can run this task to create a Dimension Value Id manager.

The syntax for this task is:

```
cas-cmd createDimensionValueIdManager [-h HostName] [-l true|false] -m managername
 [-p PortNumber]
```

Where:

- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.

- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-m` (or `--dimension_value_id_manager`) specifies name of the Dimension Value Id Manager you are creating. Required.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`, in `<install path>\CAS\workspace\conf` (on Windows) or `<install path>/CAS/workspace/conf` (on UNIX). If the property is not set, the value then defaults to `8500`. Optional.

To create a Dimension Value Id Manager:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin`.
2. Run the `createDimensionValueIdManager` task and specify the `-m` option with a name argument for the Dimension Value Id Manager.

Example of creating a Dimension Value Manager

This example creates a Dimension Value Id Manager named `dvalmgr`:

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd.bat createDimensionValueIdManager -m dvalmgr
```

Deleting a Dimension Value Id Manager

The `deleteDimensionValueIdManager` task deletes a Dimension Value Id Manager. You typically run this task before you delete an Endeca application from your environment. There is no automatic mechanism to delete a Dimension Value Id Manager. You do not need to run this task after you delete individual crawls.

The syntax for this task is:

```
cas-cmd deleteDimensionValueIdManager [-h HostName] [-l true|false] -m managername
 [-p PortNumber]
```

Where:

- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-m` (or `--dimension_value_id_manager`) specifies name of the Dimension Value Id Manager you are deleting. Required.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin`.
2. Run the `deleteDimensionValueIdManager` task and specify the `-m` option with a name argument for the Dimension Value Id Manager you want to delete.

Example of deleting a Dimension Value Manager

This example deletes a Dimension Value Id Manager named `dvalmgr`:

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd.bat deleteDimensionValueIdManager -m dvalmgr
```

Exporting dimension value Id mappings

The `exportDimensionValueIdMappings` task exports dimension value Id mappings from a Dimension Value Id Manager to a CSV file. You may want to run this task to back up dimension value Id mappings to a file or to copy dimension value Id mappings across environments.

The syntax for this task is:

```
cas-cmd exportDimensionValueIdMappings [-h HostName] [-l true|false] -m dvalmgr
[-p PortNumber] -f mappings.csv
```

Where:

- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-m` (or `--dimension_value_id_manager`) specifies name of the Dimension Value Id Manager. Required.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`, in *<install path>*`\CAS\workspace\conf` (on Windows) or *<install path>*`/CAS/workspace/conf` (on UNIX). If the property is not set, the value then defaults to `8500`. Optional.
- `-f` (or `--file_name`) specifies the path of the CSV file, which must be accessible to the CAS Service process (not just accessible to the host running the CAS Service). The file is encoded as UTF-8. The accessibility requirement has implications for files that reside on mapped network drives. For details, see the following note. Required.

Note: For Windows, you should specify network drives by universal naming convention (UNC) syntax rather than by using the letter of a mapped drive. For UNIX, you can specify mounted or local drives using standard file path syntax.

Examples of local folders on Windows:

- `D:\documents\mappings.cvs`
- `C:\tmp\mappings.csv`

Examples of syntax for network drives:

- `\\abchost.endeca.com\mappings.csv`
- `\\xyzhost\home\smith\mappings.csv`

To export dimension value Id mappings:

1. Start a command prompt and navigate to *<install path>*`\CAS\<version>\bin`.
2. Run the `exportDimensionValueIdMappings` task and specify at least the required options (`-m` and `-f`) listed above.

Examples of exporting dimension value Id mappings

This Windows example exports mappings to a local drive:

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd.bat exportDimensionValueIdMappings -m dvalmgr
-f C:\temp\mappings.csv
Successfully exported mappings from C:\temp\mappings.csv
```

This Windows example exports mappings from a mapped network drive:

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd.bat exportDimensionValueIdMappings -m dvalmgr
-f \\jsmith-t60\tmp\mappings.csv
Successfully exported mappings from \\jsmith-t60\tmp\mappings.csv
```

Generating a dimension value Id

The `generateDimensionValueId` task generates Ids given a dimension name and a dimension specification. In general, you should rarely need to run this task because CAS generates dimension value Ids as part of writing MDEX output. In cases where you need to create dimension value configuration before the data ingest process, you can run this task manually.

The syntax for this task is:

```
cas-cmd generateDimensionValueId -d dimensionname [-h HostName] [-l true|false]
-m managername [-p PortNumber]
```

Where:

- `-d` (or `--dimension`) specifies the name of the dimension that you want to generate Ids for. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-m` (or `--dimension_value_id_manager`) specifies name of the Dimension Value Id Manager that is generating the Ids. Required.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`, in *<install path>*`\CAS\workspace\conf` (on Windows) or *<install path>*`/CAS/workspace/conf` (on UNIX). If the property is not set, the value then defaults to `8500`. Optional.
- `-s` (or `--spec`) specifies the dimension value specification that you want to get the Ids of. Required.

To generate dimension value Ids:

1. Start a command prompt and navigate to *<install path>*`\CAS\<version>\bin`.
2. Run the `generateDimensionValueId` task and specify at least the required options described above.

Example of generating dimension value Ids

This example creates a Dimension Value Id Manager named `dvalmgr`:

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd.bat generateDimensionValueId -m dvalmgr -d Region
 -s Bordeaux
Generated id: 1
```

Getting a dimension value Id

The `getDimensionValueId` task returns a dimension value Id when you specify a dimension name and dimension value specification. (This task is the reverse of `getDimensionValueSpec`.) This information may be useful for debugging.

The syntax for this task is:

```
cas-cmd getDimensionValueId -d dimensionname [-h HostName] [-l true|false] -m
managername [-p PortNumber] - s dvalspec
```

Where:

- `-d` (or `--dimension`) specifies the name of the dimension that you want to get the Ids for. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-m` (or `--dimension_value_id_manager`) specifies name of the Dimension Value Id Manager that is getting the Ids. Required.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`, in *<install path>*`\CAS\workspace\conf` (on Windows) or *<install path>*`/CAS/workspace/conf` (on UNIX). If the property is not set, the value then defaults to `8500`. Optional.
- `-s` (or `--spec`) specifies the dimension value specification that you want to get the Ids of. Required.

To get a dimension value Id:

1. Start a command prompt and navigate to *<install path>*`\CAS\<version>\bin`.
2. Run the `getDimensionValueId` task and specify at least the required options listed above.

Example of getting dimension value Ids

This example gets the dimension value Ids of the Bordeaux dimension value:

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd.bat getDimensionValueId -m dvalmgr -d Region -
s Bordeaux
Bordeaux
1
```

Getting a dimension value specification

The `getDimensionValueSpec` task returns a dimension value specification when you specify a dimension name and a dimension value Id. (This task is the reverse of `getDimensionValueId`.) This information may be useful for debugging.

The syntax for this task is:

```
cas-cmd getDimensionValueSpec [-h HostName] -i dimension_value_id [-l true|false]
 -m managername [-p PortNumber]
```

Where:

- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-i` (or `--dimension_value_id`) specifies the Id of the dimension that you want to get the specification for. Required.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-m` (or `--dimension_value_id_manager`) specifies name of the Dimension Value Id Manager that is getting the Ids. Required.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`, in `<install path>\CAS\workspace\conf` (on Windows) or `<install path>/CAS/workspace/conf` (on UNIX). If the property is not set, the value then defaults to `8500`. Optional.

To get dimension a value specification:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin`.
2. Run the `getDimensionValueSpec` task and specify at least the required options listed above.

Example of getting dimension value Ids

This example gets the dimension value specification of the Bordeaux dimension value:

```
C:\Endeca\CAS\3.1.2\bin>cas-cmd.bat getDimensionValueSpec -m dvalmgr  -i 1
Dimension: Region
Spec: Bordeaux
```

Importing dimension value Id mappings

The `importDimensionValueIdMappings` task imports dimension value Id mappings from a CSV file into a Dimension Value Id Manager. The restore process completely replaces all dimension value Id mappings stored in the Dimension Value Id Manager.

The syntax for this task is:

```
cas-cmd importDimensionValueIdMappings [-h HostName] [-l true|false] -m dvalmgr
[-p PortNumber] -f mappings.csv
```

Where:

- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-m` (or `--dimension_value_id_manager`) specifies name of the Dimension Value Id Manager. Required.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`, in `<install path>\CAS\workspace\conf` (on Windows) or `<install path>/CAS/workspace/conf` (on UNIX). If the property is not set, the value then defaults to `8500`. Optional.

- `-f` (or `--file_name`) specifies the path of the CSV file, which must be accessible to the CAS Service process (not just accessible to the host running the CAS Service). This requirement has implications for files that reside on mapped network drives. For details, see the following note. Required.

Note: For Windows, you should specify network drives by universal naming convention (UNC) syntax rather than by using the letter of a mapped drive. For UNIX, you can specify mounted or local drives using standard file path syntax.

Examples of local folders on Windows:

- `D:\documents\mappings.cvs`
- `C:\tmp\mappings.csv`

Examples of syntax for network drives:

- `\\abchost.endeca.com\mappings.csv`
- `\\xyzhost\home\smith\mappings.csv`

To import dimension value Id mappings:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin`.
2. Run the `importDimensionValueIdMappings` task and specify at least the required options (`-m` and `-f`) listed above.

Examples of importing dimension value Id mappings

This Windows example imports mappings from a local drive:

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd.bat importDimensionValueIdMappings -m dvalmgr
-f C:\temp\mappings.csv
Successfully imported mappings from C:\temp\mappings.csv
```

This Windows example imports mappings from a mapped network drive:

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd.bat importDimensionValueIdMappings -m dvalmgr
-f \\jsmith-t60\tmp\mappings.csv
Successfully imported mappings from \\jsmith-t60\tmp\mappings.csv
```

Viewing crawl status and results

The following `cas-cmd` tasks return information about crawl status and crawl results.

Getting metrics for all crawls

The `getAllCrawlMetrics` task retrieves a list of crawl IDs and their associated metrics.

The syntax for this task is:

```
cas-cmd getAllCrawlMetrics [-h HostName] [-p PortNumber] [-l true|false]
```

Where:

- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.

- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

To get metrics for all crawls:

1. Start a command prompt and navigate to *<install path>*`\CAS\<version>\bin` (for Windows), or *<install path>*`/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `getAllCrawlMetrics` task.

Note: This task name is case sensitive.

Example of getting metrics for all crawls

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd getAllCrawlMetrics
Metrics for crawl FileCrawl:

ARCHIVED_DIRECTORIES_CRAWLED:   0
ARCHIVED_DIRECTORIES_FILTERED:  0
ARCHIVED_FILES_CRAWLED: 0
ARCHIVED_FILES_FILTERED:        0
CRAWL_MODE:     FULL_CRAWL
CRAWL_STATE:    NOT_RUNNING
CRAWL_STOP_CAUSE:    COMPLETED
DELETED_RECORDS:        0
DIRECTORIES_CRAWLED:    3009
DIRECTORIES_FILTERED:   0
DURATION_IN_SECONDS:    595
END_TIME:       Thu Apr 23 13:46:27 EDT 2009
FAILED_TEXT_EXTRACTIONS:        65
FILES_CRAWLED:  28849
FILES_FILTERED: 0
NEW_OR_UPDATED_RECORDS: 31858
NONARCHIVED_DIRECTORIES_CRAWLED:        3009
NONARCHIVED_DIRECTORIES_FILTERED:       0
NONARCHIVED_FILES_CRAWLED:      28849
NONARCHIVED_FILES_FILTERED:     0
START_TIME:     Thu Apr 23 13:36:32 EDT 2009
SUCCESSFUL_TEXT_EXTRACTIONS:    1420
SUCCESSFUL_TEXT_EXTRACTIONS_AFTER_RETRY:        1
TOTAL_RECORDS:  31858

Metrics for crawl FileCrawl2:

ARCHIVED_DIRECTORIES_CRAWLED:   3787
ARCHIVED_DIRECTORIES_FILTERED:  0
ARCHIVED_FILES_CRAWLED: 62085
ARCHIVED_FILES_FILTERED:        0
CRAWL_MODE:     FULL_CRAWL
CRAWL_STATE:    NOT_RUNNING
CRAWL_STOP_CAUSE:    COMPLETED
DELETED_RECORDS:        0
DIRECTORIES_CRAWLED:    16504
```

```
DIRECTORIES_FILTERED:   0
DURATION_IN_SECONDS:    1569
END_TIME:       Thu Apr 23 14:37:53 EDT 2009
FAILED_TEXT_EXTRACTIONS:        67
FILES_CRAWLED:  153511
FILES_FILTERED: 0
NEW_OR_UPDATED_RECORDS: 170015
NONARCHIVED_DIRECTORIES_CRAWLED:        12717
NONARCHIVED_DIRECTORIES_FILTERED:       0
NONARCHIVED_FILES_CRAWLED:      91426
NONARCHIVED_FILES_FILTERED:     0
START_TIME:     Thu Apr 23 14:11:44 EDT 2009
SUCCESSFUL_TEXT_EXTRACTIONS:    7109
SUCCESSFUL_TEXT_EXTRACTIONS_AFTER_RETRY:        1
TOTAL_RECORDS:  170015
```

Getting the metrics for a crawl

The `getCrawlMetrics` task retrieves metrics for a particular crawl.

The syntax for this task is:

```
cas-cmd getCrawlMetrics -id CrawlName [-h HostName] [-p PortNumber] [-l true|false]
```

Where:

- `-id` (or `--crawl_id`) specifies the name of the crawl for to retrieve metrics for. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

1. Start a command prompt and navigate to *<install path>*`\CAS\<version>\bin` (for Windows), or *<install path>*`/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `getCrawlMetrics` task with the id of the crawl for which you want to get metrics.

Note: This task name is case sensitive.

Example of getting the metrics for a crawl

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd getCrawlMetrics -id Test
ARCHIVED_DIRECTORIES_CRAWLED:   0
ARCHIVED_DIRECTORIES_FILTERED:  0
ARCHIVED_FILES_CRAWLED: 0
ARCHIVED_FILES_FILTERED:        0
CRAWL_MODE:     FULL_CRAWL
CRAWL_STATE:    NOT_RUNNING
CRAWL_STOP_CAUSE:       COMPLETED
```

```
DELETED_RECORDS:        0
DIRECTORIES_CRAWLED:    97
DIRECTORIES_FILTERED:   0
DURATION_IN_SECONDS:    25
END_TIME:       Thu Jan 07 16:33:17 EST 2010
FAILED_RECORDS: 0
FAILED_TEXT_EXTRACTIONS:        0
FILES_CRAWLED:  688
FILES_FILTERED: 0
NEW_OR_UPDATED_RECORDS: 785
NONARCHIVED_DIRECTORIES_CRAWLED:        97
NONARCHIVED_DIRECTORIES_FILTERED:       0
NONARCHIVED_FILES_CRAWLED:      688
NONARCHIVED_FILES_FILTERED:     0
START_TIME:     Thu Jan 07 16:32:51 EST 2010
SUCCESSFUL_TEXT_EXTRACTIONS:    557
SUCCESSFUL_TEXT_EXTRACTIONS_AFTER_RETRY:        0
TOTAL_RECORDS:  785
```

Getting the status of a crawl

The `getCrawlStatus` task returns the status of a specific crawl.

The syntax for this task is:

```
cas-cmd getCrawlStatus -id CrawlName [-h HostName] [-p PortNumber] [-l true|false]
```

Where:

- `-id` (or `--crawl_id`) specifies the name of the crawl to retrieve status for. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

To get the status of a crawl:

1. Start a command prompt and navigate to *<install path>*`\CAS\<version>\bin` (for Windows), or *<install path>*`/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `getCrawlStatus` task with the id of the acquisition.

Note: This task name is case sensitive.

Example of getting the status of a crawl

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd getCrawlStatus -id FileCrawl
RUNNING
```

Chapter 13

Component Instance Manager Command-line Utility

This section describes how to run the tasks of the Component Instance Manager (CIM) Command-line Utility.

Overview of the CIM Command-line Utility

The Component Instance Manager (CIM) Command-line Utility is a tool to create components, delete components, and view components. The Endeca CAS Service must be running before you can execute any of the CIM Command-line Utility tasks.

In this version of CAS, the types of components you can manage with the CIM Command-line Utility are Record Store components. In future releases, the utility may be extended to manage additional types of components.

The CIM Command-line Utility is a script named `component-manager-cmd.sh` (for Linux/UNIX systems) and `component-manager-cmd.bat` (on Windows) that you run from a command prompt. The script is in the `bin` directory.

Help options

The CIM Command-line Utility has two help options that display the usage syntax. The `--help` option displays a summary of the tasks. The `--help-detail` option displays detailed usage information for all the tasks.

For example:

```
C:\Endeca\CAS\11.0.0\bin>component-manager-cmd.bat --help
usage: component-manager-cmd <task-name> [options]

  list-types
  list-components
  create-component
  delete-component

For detailed usage information including task options, use --help-detail
For detailed usage information for individual task options, use <task-name> --
help
```

Command-line options

The command syntax for executing the tasks is:

```
component-manager-cmd task-name [options]
```

The *task-name* argument is the task to be performed by the utility, such as the `createCrawls` task. The task options vary, depending on the task. However, these options can be used with any task:

- `-h` (or `--host`) specifies the host name of the machine on which the CAS Service is running. If the flag is omitted, it defaults to the value of the `com.endeca.itl.cas.server.host` property in `workspace\CAS\conf\commandline.properties`. If the property is not set, the value then defaults to `localhost` as the host name.
- `-p` (or `--port`) specifies the port on which CAS Service is listening. If the flag is omitted, it defaults to the value of the `com.endeca.itl.cas.server.port` property in *<install path>*`\workspace\CAS\conf\commandline.properties`. If the property is not set, the value then defaults to `8500` as the port number.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

Host and port settings

You first specify the host and port settings for the Endeca CAS Service as part of the installation wizard. That host runs the CAS Server, the Component Instance Manager, and all Record Store instances. The installation wizard then writes the host (`com.endeca.itl.cas.server.host`) and port (`com.endeca.itl.cas.server.port`) settings as properties in `commandline.properties`. All of the command-line utilities use these settings as default values if you omit the `-h` and `-p` flags when executing any tasks.

Setting the bin directory in the PATH environment variable

Although not required, it is recommended that you set the path of the `bin` directory in your systems' PATH environment variable. This allows you to run the Component Instance Manager Command-line Utility script from any location.

About error handling

- If desired, you can re-configure the default logging settings in *<install path>*`\CAS\workspace\conf\component-manager-cmd.log4j.properties`.
- Errors print to standard error, unless you redirect `std err` to a file instead.
- Errors of mis-configured command-line tasks or incorrect input parameters are written to standard out.

Creating a Record Store

The `create-component` task creates a Record Store instance.

The syntax for this task is:

```
component-manager-cmd create-component -n RecordStoreName  -t RecordStore
[-h HostName] [-p PortNumber] [-l true|false]
```

Where:

- `-n` specifies the name of the component you are creating. Required.
- `-t` specifies the type of the component instance you want to create. Specify `RecordStore`. Required.
- `-h` (or `--host`) specifies the host where the CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.

- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

To create a Record Store:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin`.
2. Run the `create-component` task and specify the `-t` option with an argument of `RecordStore` and specify the`-n` option with a Record Store instance name of your choice.

Example of creating a Record Store

This example creates a Record Store named RS1:

```
C:\Endeca\CAS\11.0.0\bin>component-manager-cmd.bat create-component
 -n RS1 -t RecordStore
```

Deleting a Record Store

The `delete-component` task deletes a Record Store.

The syntax for this task is:

```
component-manager-cmd delete-component -n RecordStoreName
[-h HostName] [-p PortNumber] [-l true|false]
```

where:

- `-n` specifies the name of the component you are deleting. Required.
- `-h` (or `--host`) specifies the host where the Component Instance Manager is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Component Instance Manager. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

To delete a Record Store:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin`.
2. Run the `delete-component` task and specify the `-n` option.

Example of deleting a Record Store

This example deletes a Record Store named RS1:

```
C:\Endeca\CAS\11.0.0\bin>component-manager-cmd.bat
delete-component -n RS1
```

Listing components

The `list-components` task lists all component instances that are managed by the Component Instance Manager. Executing the task returns a list of all managed components in the CAS Service.

The syntax for this task is:

```
component-manager-cmd list-components [-h HostName] [-p PortNumber]
[-l true|false]
```

where:

- `-h` (or `--host`) specifies the host where the Component Instance Manager is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Component Instance Manager. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

To list components:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin`.
2. Run the `list-components` task with any of the options listed above.

Example of listing components

This example lists the Record Store instances and Dimension Value Id Manager for a sample application running in the Endeca CAS Service:

```
C:\Endeca\CAS\11.0.0\bin>component-manager-cmd.bat list-components
NAME            TYPE            STATUS
ebizsampleapp-trigger-dimensions        RecordStore     RUNNING
ebizsampleapp-products  RecordStore     RUNNING
ebizsampleapp-dimension-value-id-manager        DimensionValueIdManager RUNNING
ebizsampleapp-category-dimension        RecordStore     RUNNING
```

If no components have been created, the `list-components` task returns the following:

```
No components have been provisioned.
```

Listing types

The `list-types` task lists all component types that are managed by the Component Instance Manager. Executing the task returns a list of all managed component types in the CAS Service.

In this release, the only supported component type is `RecordStore`.

The syntax for this task is:

```
component-manager-cmd list-types [-h HostName] [-p PortNumber]
[-l true|false]
```

where:

- `-h` (or `--host`) specifies the host where the Component Instance Manager is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-p` (or `--port`) specifies the port of the Component Instance Manager. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

To list component types:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin`.
2. Run the `list-types` task with any of the options listed above.

Example of listing types

This example lists the type of components running on the Endeca CAS Service:

```
C:\Endeca\CAS\11.0.0\bin>component-manager-cmd.bat list-types
ID              PATH
DimensionValueIdManager dval-id-mgr-core-11.0.0.jar
RecordStore     recordstore-core-11.0.0.jar
```

Chapter 14

Record Store Command-line Utility

This section describes how to run the tasks of the Record Store Command-line Utility.

Overview of the Record Store Command-line Utility

The Record Store Command-line Utility provides the ability to read records from and write records to a Record Store instance, in addition to a number of utility tasks such as setting client IDs and rolling back transactions.

The Record Store Command-line Utility is a script named `recordstore-cmd.sh` (for Linux/UNIX systems) and `recordstore-cmd.bat` (for Windows) that you run from a command prompt. The scripts are in the `bin` directory.

Transactions

Read and write operations take place within the scope of a transaction. You can specify the start, commit, or roll back of a transaction. This is useful in cases where you want to perform multiple operations within the scope of a single transaction. If you do not explicitly control the transaction, all read and write operations take place in a default auto commit mode.

Help options

The Record Store Command-line Utility has two help options that display the usage syntax. The `--help` option displays a summary of the tasks. The `--help-detail` option displays detailed usage information for all the tasks. For example:

```
C:\Endeca\CAS\11.0.0\bin>recordstore-cmd --help
usage: recordstore-cmd <task-name> [options]

[READ TASKS]
  read-baseline
  read-delta
  read-by-id
[UTILITY TASKS]
  clean
  clear-last-read-generation
  commit-transaction
  get-configuration
  get-last-committed-generation
  get-last-read-generation
  get-write-generation
  list-active-transactions
  list-client-states
```

```
  list-generations
  rollback-transaction
  set-configuration
  set-last-read-generation
  start-transaction
[WRITE TASKS]
  write

For detailed usage information including task options, use --help-detail
For detailed usage information for individual task options, use <task-name> --
help
```

Command-line options

With one exception, the command syntax for executing the tasks is:

```
recordstore-cmd task-name [options]
```

The exception to this syntax format is the `read-by-id` task, which is explained in its own topic.

The *task-name* argument is the task to be performed by the utility, such as the `read-delta` task. The task options vary, depending on the task. However, these options can be used with any task:

- `-h` (or `--host`) specifies the host name of the machine on which the Record Store is running. If the flag is omitted, it defaults to the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost` as the host name.
- `-p` (or `--port`) specifies the port on which the Record Store is listening. If the flag is omitted, it defaults to the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500` as the port number.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.

Host and port settings

You first specify the host and port settings for the Endeca CAS Service as part of the installation wizard. That host runs the CAS Server, the Component Instance Manager, and all Record Store instances. The installation wizard then writes the host (`com.endeca.itl.cas.server.host`) and port (`com.endeca.itl.cas.server.port`) settings as properties in `commandline.properties`. All of the command-line utilities use these settings as default values if you omit the `-h` and `-p` flags when executing any tasks.

Setting the bin directory in the PATH environment variable

Although not required, it is recommended that you set the path of the `bin` directory in your systems' PATH environment variable. This allows you to run the Record Store Command-line Utility script from any location.

About error handling

- If desired, you can re-configure the default logging settings in *<install path>*`\CAS\workspace\conf\recordstore-cmd.log4j.properties`.
- By default, errors print to a log file named `recordstore-cmd.log` that is located in the `logs` directory.

Writing tasks

The following `recordstore-cmd` tasks perform write operations to a Record Store instance.

Writing records

The `write` task writes a list of records into a specified Record Store instance.

The syntax for this task is:

```
recordstore-cmd write -a RecordStoreInstanceName [-b]
-f InputFile [-h HostName] [-l true|false] [-p PortNumber] [-r Type] [-x Id]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-b` (or `--baseline`) is a flag with no arguments that specifies that this is to be a baseline write. If the Record Store has any existing generations, a baseline write will not delete those previous generations, however, it will mark them as "to be deleted" and the cleaner will delete them when it runs (if the records are older than the generation retention time). If the flag is omitted, the write operation is considered an incremental write to the last-committed generation. Optional.
- `-f` (or `--file`) specifies the file that contains Endeca records. The filename extension will determine the format of the input file. Valid extensions for the file are `.xml` (for an XML format) and `.bin` (for a binary format); either file type can also have an additional, optional `.gz` extension if it is a compressed file. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-x` (or `--transaction`) specifies the active transaction ID to use. If you use this option, you must follow the command with a `commit-transaction` task to commit the write operation. If this flag is omitted, the operation is done in auto-commit mode. Optional.

Examples of writing records

If there are two generations in the Record Store, this command:

```
recordstore-cmd write -a RS2 -b -f basedata.xml
```

will write the records in the `basedata.xml` file as a baseline write operation. If you check the log output, you should see messages similar to these:

```
Starting new transaction with generation Id 3
Started transaction 10 of type READ_WRITE
Processing delete all for generation 3
Marking generation committed: 3
Committed transaction 10
```

The Delete message (`Processing delete all for generation 3`) indicates that the transaction that created Generation 3 also marked the previous generations for deletion.

If you then perform a subsequent incremental write command:

```
recordstore-cmd write -f incrdata.xml
```

the console or log output messages should look like these:

```
Starting new transaction with generation Id 4
Started transaction 11 of type READ_WRITE
Marking generation committed: 4
Committed transaction 11
```

At this point, the Record Store has two generations: Generation 3 is a baseline generation and Generation 4 is an incremental generation. If you then run a Forge baseline update, it will use both generations.

Reading tasks

The following `recordstore-cmd` tasks perform read operations from a Record Store instance. Note that read operations on uncommitted data are not supported.

Reading baselines

The `read-baseline` task reads the baseline records from a Record Store instance.

The syntax for this task is:

```
recordstore-cmd read-baseline -a RecordStoreInstanceName
[-c] [-f FileName.xml] [-g GenId] [-h HostName] [-l true|false]
[-p PortNumber] [-n NumRecs] [-x id]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-c` (or `--count`) that only prints the record count from the read. Optional.
- `-f` (or `--file`) specifies the pathname of the file to which the Endeca records will be output. The filename extension determines the format of the output file. Valid extensions for the file are `.xml` (for an XML format) and `.bin` (for a binary format); the file can also have an additional, optional `.gz` extension if it is a compressed file. If unspecified, the record are written to the console. Optional.
- `-g` (or `--generation`) specifies the ID of the generation from which the records are read. If omitted, records from the last-committed generation are read. Optional.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-n` (or `--firstN`) specifies that only the first *numRecs* records of the baseline will be read. If omitted, all records are read. Optional.

- `-x` (or `--transaction`) specifies the active transaction ID to use. If you use this option, you must follow it with a `commit-transaction` task to commit the read operation. If this flag is omitted, the operation is done in auto-commit mode. Optional.

Examples of reading baselines

The first example reads the first 50 baseline records (from the last-committed generation) and outputs them to a file:

```
recordstore-cmd read-baseline -a RS1 -n 50 -f c:\recdata\basedata.xml
```

The output is written in an XML format to the `basedata.xml` file located in the `C:\recdata` directory.

The second example prints the number of records in the baseline:

```
recordstore-cmd read-baseline -a RS1 -c -g 2
```

The command prints out the number of records in generation 2 of the Record Store.

Reading delta records

The `read-delta` task reads the delta between two or more generations in the Record Store.

Delta records can be one of three types:

- Modified records. A modified record has the same record ID as the previous version, but the content (as determined from the `changePropertyNames` property) has changed.
- Added records. An added (new) record will have a record ID that does not appear in the previous generations.
- Deleted records. A deleted record will have a valid record ID, but its `Endeca.Action` property will be set to `DELETE`.

The syntax for this task is:

```
recordstore-cmd read-delta -a RecordStoreInstanceName [-c] [-f FileName]
[-n NumRecs] [-h HostName] [-l true|false] [-p PortNumber] [-s StartGenId]
[-e EndGenId] [-x Id]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-c` (or `--count`) prints the record count from the read. Optional.
- `-f` (or `--file`) specifies the pathname of the file to which the Endeca records will be output. The filename extension will determine the format of the output file. Valid extensions for the file are `.xml` (for an XML format) and `.bin` (for a binary format); the file can also have an additional, optional `.gz` extension if it is a compressed file. Optional.
- `-n` (or `--firstN`) specifies that only the first *numRecs* number of delta records will be read. If omitted, all records are read. Optional.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.

- `-s` (or `--startGeneration`) specifies the ID of the start generation from which the diff will be done. If omitted, the initial generation is used. Optional.
- `-e` (or `--endGeneration`) specifies the ID of the end generation from which the diff will be done. If omitted, the last-committed generation is used. Optional.
- `-x` (or `--transaction`) specifies the active transaction ID to use. If you use this option, you must follow it with a `commit-transaction` task to commit the read operation. If this flag is omitted, the operation is done in auto-commit mode. Optional.

For this task, it is recommended that you explicitly specify the start and end generations.

Example of reading delta records

This example reads all the delta records that constitute the difference between Generation 1 and Generation 2 and writes them to a file:

```
recordstore-cmd read-delta -a RS1 -f c:\recdata\diffdata.xml -s 1 -e 2
```

The delta records are written in an XML format to the `diffdata.xml` file located in the `C:\recdata` directory.

If you only want a record count of the difference, use the `-c` option:

```
recordstore-cmd read-delta -a RS1 -c -s 1 -e 2
```

The number of delta records read is output to the console.

Reading specific records

The `read-by-id` task reads one or more specific records from a Record Store instance.

The `read-by-id` task uses a syntax that is different from the other tasks. The difference is that you specify the record IDs at the end of the command line (i.e., after all the options have been specified).

The syntax for this task is:

```
recordstore-cmd read-by-id -a RecordStoreInstanceName [-c] [-f FileName]
[-g GenId] [-h HostName] [-l true|false] [-p PortNumber] [-x Id]
[RecId1 [RecId2 [... RecIdN]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-c` (or `--count`) prints the record count from the read. Optional.
- `-f` (or `--file`) specifies the pathname of the file to which the Endeca records are output. The filename extension determines the format of the output file. Valid extensions for the file are `.xml` (for an XML format) and `.bin` (for a binary format); the file can also have an additional, optional `.gz` extension to create a compressed file. Optional.
- `-g` (or `--generation`) specifies the ID of the generation from which the records are read. If omitted, records from the last-committed generation are read. Optional.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.

- `-x` (or `--transaction`) specifies the active transaction ID to use. If you use this option, you must follow it with a `commit-transaction` task to commit the read operation. If this flag is omitted, the operation is done in auto-commit mode. Optional.
- `recId` is the ID of the record to read. The record ID is the value of the record property being used for the `idPropertyName` configuration property. For multiple records, you must specify a space-delimited list of record IDs. If an ID contains a space, enclose the ID within double quotation marks.

Example of reading a specific record

Assume that the `idPropertyName` configuration in the Record Store instance is set to the `Endeca.Web.URL` record property. Also assume that you want to read a record that has this value:

```
<PROP NAME="Endeca.Web.URL">
   <PVAL>http://endeca.com/contact.html</PVAL>
</PROP>
```

This means that the string `http://endeca.com/contact.html` is the ID of that record. You would therefore retrieve that record with this command:

```
recordstore-cmd read-by-id -a RS1 -f rec.xml http://endeca.com/contact.html
```

The record will be written in an XML format to the `rec.xml` file.

Utility tasks

The following `recordstore-cmd` tasks perform utility operations to manage a Record Store instance.

Cleaning a Record Store instance

The `clean` task manually removes stale generations of records from a specified Record Store instance.

By default, the `clean` task runs automatically as a background process, at time intervals specified by the `cleanerInterval` configuration property. The `clean` task automatically removes record generations that exceed the `generationRetentionTime` configuration property.

The task syntax is:

```
recordstore-cmd clean -a RecordStoreInstanceName [-h HostName]
[-l true|false] [-p PortNumber]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.

Note that only one `clean` process can run at a time, and a `clean` task cannot run when other transactions are running. If a `clean` task is running when you issue this command, an exception is thrown and the second `clean` process does not run.

There are several cases where the `clean` task does not remove eligible generations in a Record Store instance:

- If it is the only generation in a Record Store instance.
- If the generation is in use.
- If it is the last committed generation.
- If it is the last generation read by a client, such as a Forge Record Store adapter or the Record Store API.

Clearing the last read generation

The `clear-last-read-generation` task clears the last-read generation for a given client ID. This task is the counterpart of `set-last-read-generation`.

The syntax for this task is:

```
recordstore-cmd clear-last-read-generation  -a RecordStoreInstanceName
-c ClientId [-h HostName] [-l true|false] [-p PortNumber] [-x Id]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-c` (or `--client`) specifies a string to identify the client ID. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-x` (or `--transaction`) specifies the active transaction ID to use. If you use this option, you must follow it with a `commit-transaction` task to commit the read operation. If this flag is omitted, the operation is done in auto-commit mode. Optional.

Example of setting the last-read generation

This example clears the last read generation flag for the client ID `forge1`:

```
recordstore-cmd clear-last-read-generation -a RS1 -c forge1
```

Committing transactions

The `commit-transaction` task commits an active (uncommitted) transaction for a specified Record Store instance.

The syntax for this task is:

```
recordstore-cmd commit-transaction -a RecordStoreInstanceName -x Id
[-h HostName] [-l true|false] [-p PortNumber]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-x` (or `--transaction`) specifies the ID of the transaction that will be committed. Required.

Example of committing a transaction

This example commits the transaction with an ID of 8:

```
recordstore-cmd commit-transaction -a RS1 -x 8
```

If the command succeeds, it prints the following message:

```
Committed transaction: 8
```

If the command fails, it prints the following error message:

```
Failed to commit transaction: 8
```

Getting the configuration of a Record Store instance

The `get-configuration` task returns the configuration settings of a specified Record Store instance.

A Record Store instance has a default configuration that you can retrieve and save. You can modify the configuration and use it to configure a new Record Store or reconfigure an existing Record Store instance.

The syntax for this task is:

```
recordstore-cmd get-configuration -a RecordStoreInstanceName
-f FileName.xml [-h HostName] [-l true|false] [-n] [-p PortNumber]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-f` (or `--file`) specifies the XML file name where you want to save the configuration settings. Omitting this option sends the XML for the configuration settings to stdout. Optional.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-n` (or `--normalize`) specifies whether to normalize the configuration settings. Specifying this option returns all default configuration settings and their associated default values. Omitting this option returns only user-specified settings. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.

To get the configuration of a Record Store:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin`.
2. Run `recordstore-cmd` and specify options as documented above.

Example of getting the configuration of a Record store

This Windows example gets the configuration for a Record Store named RS1:

```
recordstore-cmd.bat get-configuration -a RS1 -f config.xml -n
```

The command output of the example above is stored in `config.xml` and is also shown here:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<recordStoreConfiguration xmlns="http://recordstore.itl.endeca.com/">
    <btreePageSize>100</btreePageSize>
    <changePropertyNames/>
    <cleanerInterval>1.0</cleanerInterval>
    <dataDirectory>C:\Endeca\CAS\workspace\state\RS1\data</dataDirectory>
    <generationRetentionTime>168.0</generationRetentionTime>
    <idPropertyName>Endeca.Id</idPropertyName>
    <ignoreInvalidRecords>false</ignoreInvalidRecords>
    <indexWriteFlushInterval>50000</indexWriteFlushInterval>
    <jdbmSettings/>
    <mergeBufferSize>131072</mergeBufferSize>
    <writeBufferSize>104857600</writeBufferSize>
    <recordCompressionEnabled>false</recordCompressionEnabled>
</recordStoreConfiguration>
```

Getting the ID of the last-committed generation

The `get-last-committed-generation` task retrieves the ID of the last generation that was committed to a Record Store instance.

The syntax for this task is:

```
recordstore-cmd get-last-committed-generation -a RecordStoreInstanceName
[-h HostName] [-l true|false] [-p PortNumber] [-x Id]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-x` (or `--transaction`) specifies the active transaction ID to use. If you use this option, you must follow it with a `commit-transaction` task to commit the read operation. If this flag is omitted, the operation is done in auto-commit mode. Optional.

Example of getting the last-committed generation ID

The output of this command:

```
recordstore-cmd get-last-committed-generation -a RS1
```

will be similar to this example:

```
The last committed generation: 4
```

The command output shows that Generation 4 was the last generation to be committed to the Record Store.

Getting the last-read generation

The `get-last-read-generation` task retrieves the last-read generation for a given client ID.

Before running this task, make sure to use the `set-last-read-generation` task to set a last-read generation for a specific client ID.

The syntax for this task is:

```
recordstore-cmd get-last-read-generation -a RecordStoreInstanceName
-c ClientId [-h HostName] [-l true|false] [-p PortNumber] [-x Id]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-c` (or `--client`) specifies a client ID that was previously set with the `set-last-read-generation` task. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-x` (or `--transaction`) specifies the active transaction ID to use. If you use this option, you must follow it with a `commit-transaction` task to commit the read operation. If this flag is omitted, the operation is done in auto-commit mode. Optional.

Example of getting the last-read generation

This example gets the last-read generation for the client ID of `forge1`:

```
recordstore-cmd get-last-read-generation -a RS1 -c forge1
```

The command output is similar to this example:

```
The last read generation id saved for client forge1 is: 2
```

In the example, Generation 2 had been previously set as the last read generation for the `forge1` client ID.

Getting the ID of the write generation

The `get-write-generation` task returns the ID of the write generation.

The syntax for this task is:

```
recordstore-cmd get-write-generation -a RecordStoreName -x id
[-h HostName] [-l true|false] [-p PortNumber]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-x` (or `--transaction`) specifies the active transaction ID to use. The transaction must be a `READ_WRITE` type. Required.

Example of getting the write-generation ID

```
recordstore-cmd get-write-generation -a RS1 -x 5
Write generation: 2
```

The output of the `get-write-generation` task shows that Generation 2 is the current write generation.

Listing active transactions

The `list-active-transactions` task lists all the existing active transactions of a specified Record Store instance.

Uncommitted transactions are often the result of an unexpected termination of a crawl or other write operation. In this case, you see an error in the log file that includes the ID of the uncommitted transaction.

The syntax for this task is:

```
recordstore-cmd list-active-transactions  -a RecordStoreInstanceName
[-h HostName] [-l true|false] [-p PortNumber]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.

The task output consists of these fields:

- `ID` - an integer that is the transaction ID.

- TYPE - the transaction type, which is READ (which supports only Read operations) or READ_WRITE (which supports both Read and Write operations).
- STATUS - the status of the transaction, which is ACTIVE (the transaction is in progress), COMMITTED (the transaction has been committed), COMMIT_FAILED (the commit task failed for this transaction), or ROLLED_BACK (the transaction was rolled back).
- WRITING_GEN - either the new generation ID (for READ_WRITE types) or N/A (for READ types, because a new generation is not being written).
- LAST_COMMITTED - the generation ID of the last committed generation.

If no transactions are active, this message is displayed:

```
There are no active transactions right now.
```

Example of listing active transactions

If there are active transactions, the output of this command:

```
recordstore-cmd list-active-transactions -a RS1
```

will be similar to this example:

```
ID      TYPE            STATUS  WRITING_GEN     LAST_COMMITTED_GEN
13      READ            ACTIVE  N/A             2
14      READ_WRITE      ACTIVE  3               2
```

Listing generations

The list-generations task lists information about the generations that are currently in a Record Store instance.

The syntax for this task is:

```
recordstore-cmd list-generations -a RecordStoreInstanceName
[-h Hostname] [-l true|false] [-p PortNumber]
```

where:

- -a (or --instanceName) specifies the name of a Record Store instance. Required.
- -h (or --host) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the com.endeca.itl.cas.server.host property in commandline.properties. If the property is not set, the value then defaults to localhost. Optional.
- -l (or --isPortSsl) specifies whether to communicate with the service using an HTTPS connection. A value of true uses HTTPS and treats the com.endeca.itl.cas.server.port property as an SSL port. A value of false uses HTTP and treats com.endeca.itl.cas.server.port as a non-SSL port. Specify false if you enabled redirects from a non-SSL port to an SSL port. Optional.
- -p (or --port) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the com.endeca.itl.cas.server.port property in commandline.properties. If the property is not set, the value then defaults to 8500. Optional.

The task output consists of these fields:

- ID - an integer that is the generation ID.
- STATUS - the status of the generation, which is STARTED (the generation is being written to the Record Store instance), COMPLETED (the generation has been written and committed to the Record Store instance), or BEING_CLEANED (the cleaner is currently cleaning the generation).
- CREATION TIME - the date (in YYYYMMDD format) and time (in UTC format) that the generation was created. This value is based on the clock of the machine running the Endeca CAS Service.

If no generations have been written, this message displays:

```
There are no generations in the record store
```

Example of listing generations

If there are generations in the Record Store instance, the output of this command:

```
recordstore-cmd list-generations -a RS1
```

is similar to this example:

```
ID      STATUS          CREATION TIME
1       COMPLETED       2008-05-21T14:35:16.326Z
2       COMPLETED       2008-05-21T17:32:30.658Z
```

The sample output shows that there are two committed generations in the Record Store instance.

Rolling back transactions

The `rollback-transaction` task rolls back an active (uncommitted) transaction for a specified Record Store instance. Once a transaction is rolled back, this can not be undone.

The syntax for this task is:

```
recordstore-cmd rollback-transaction  -a RecordStoreInstanceName -x Id
[-h HostName] [-l true|false] [-p PortNumber]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-x` (or `--transaction`) specifies the ID of the transaction that is rolled back. Required.

Note that uncommitted transactions are often the result of an unexpected termination of a crawl. In this case, you see an error in the log file that includes the ID of the uncommitted transaction.

Example of a transaction rollback

This example rolls back the transaction with an ID of 7:

```
recordstore-cmd rollback-transaction -a RS1 -x 7
```

If the command succeeds, it prints the following message:

```
Rolled back transaction: 7
```

If the command fails, it prints the following error message:

```
Failed to roll back transaction: 7
```

Setting the configuration of a Record Store instance

The `set-configuration` task sets configuration settings for a specified Record Store instance.

You can configure a new Record Store instance or reconfigure an existing Record Store instance by specifying an XML configuration file for it.

The syntax for this task is:

```
recordstore-cmd set-configuration -a RecordStoreInstanceName
-f FileName.xml [-h HostName] [-l true|false] [-p PortNumber]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-f` (or `--file`) specifies the XML file name that contains the configuration settings for a Record Store. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.

Note: If you set configuration using a file that modifies any of the following properties, the set operation automatically clears all record data in the Record Store instance:

- `btreePageSize`
- `changePropertyNames`
- `idPropertyName`
- `jdbmSettings`
- `recordCompressionEnabled`

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin`.
2. Run `recordstore-cmd` and specify options as documented above.

Example of setting the configuration of a Record store

This example sets the configuration for a Record Store named RS2:

```
recordstore-cmd.bat set-configuration -a RS2 -f config.xml
```

where the contents of config.xml are as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<recordStoreConfiguration xmlns="http://recordstore.itl.endeca.com/">
    <btreePageSize>100</btreePageSize>
    <changePropertyNames/>
    <cleanerInterval>1.0</cleanerInterval>
    <dataDirectory>C:\Endeca\CAS\workspace\state\RS1\data</dataDirectory>
   <duplicateRecordCompressionEnabled>false</duplicateRecordCompressionEnabled>

    <generationRetentionTime>168.0</generationRetentionTime>
    <idPropertyName>Endeca.Id</idPropertyName>
    <ignoreInvalidRecords>false</ignoreInvalidRecords>
```

```
    <indexWriteFlushInterval>50000</indexWriteFlushInterval>
    <jdbmSettings/>
    <maxDataFileSize>2147483647</maxDataFileSize>
    <recordCompressionEnabled>false</recordCompressionEnabled>
</recordStoreConfiguration>
```

Note: This example deletes all records per the note above.

Setting the last-read generation

The `set-last-read-generation` task sets the last-read generation for a given client ID. This task is the counterpart of `clean-last-read-generation`.

As a result, you are setting the state for that client. This task is mainly used to save the last-read generation for use by a future delta read.

The syntax for this task is:

```
recordstore-cmd set-last-read-generation  -a RecordStoreInstanceName -g GenId
-c ClientId [-h HostName] [-l true|false] [-p PortNumber] [-x Id]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-g` (or `--generation`) specifies the generation ID to set as the last read for the client. Required.
- `-c` (or `--client`) specifies a string for which the last-read generation will be set. You can use any string for the client ID, as it is used only as an identifier. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-x` (or `--transaction`) specifies the active transaction ID to use. If you use this option, you must follow it with a `commit-transaction` task to commit the read operation. If this flag is omitted, the operation is done in auto-commit mode. Optional.

Typically, you use this command so that the Record Store instance can save the state. For example, if you do a baseline-read of Generation 2, you might later come back to the Record Store instance and do a delta read of your last-read generation (in this case it is Generation 2) and the most recently-committed generation. So you would save the first baseline-read of Generation 2 for the client, for example, `forge1`, and then perform a delta read later after getting the last-read generation.

Example of setting the last-read generation

This example sets Generation 2 as the last read for the client ID of `forge1`:

```
recordstore-cmd set-last-read-generation -a RS1 -c forge1 -g 2
```

If the command succeeds, it prints out this message:

```
Set the last read generation id for client forge1 to 2.
```

By using this method, Forge does not need to maintain state in order to use the Record Store; instead, it can push to the Record Store instance to maintain the state.

Starting transactions

The `start-transaction` task begins a Read or Write transaction. Explictly starting and committing transactions is useful if you want to group multiple operations within a single transaction.

If you choose not to use transactions, all read and write operations are performed in auto-commit mode.

The syntax for this task is:

```
recordstore-cmd start-transaction -a RecordStoreInstanceName -t Type
[-h HostName] [-l true|false] [-p PortNumber]
```

where:

- `-a` (or `--instanceName`) specifies the name of a Record Store instance. Required.
- `-h` (or `--host`) specifies the host where the Endeca CAS Service is running. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.host` property in `commandline.properties`. If the property is not set, the value then defaults to `localhost`. Optional.
- `-l` (or `--isPortSsl`) specifies whether to communicate with the service using an HTTPS connection. A value of `true` uses HTTPS and treats the `com.endeca.itl.cas.server.port` property as an SSL port. A value of `false` uses HTTP and treats `com.endeca.itl.cas.server.port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port. Optional.
- `-p` (or `--port`) specifies the port of the Endeca CAS Service. If the flag is omitted, the default is the value of the `com.endeca.itl.cas.server.port` property in `commandline.properties`. If the property is not set, the value then defaults to `8500`. Optional.
- `-t` (or `--transactionType`) specifies the type of transaction to be performed: `READ` (which supports only Read operations) or `READ_WRITE` (which supports both Read and Write operations). Note that the transaction-type arguments are case sensitive. Required.

Note: A `java.lang.IllegalArgumentException` is thrown if the `-t` argument is invalid (such as using lower case).

Example of starting a transaction

This example starts a Read transaction:

```
recordstore-cmd start-transaction -a RS1 -t READ
```

If the command is successful, it prints a message similar to this example:

```
Started transaction: 15
```

Part 5

Administering CAS

- *Running CAS components*
- *Backing up and restoring CAS*
- *Configuring logging*
- *Tips and troubleshooting CAS*

Chapter 15

Running CAS components

This section provides information on how to run the CAS components.

About running CAS components

This topic provides an overview of the recommended way to run CAS components, optional ways to run CAS components, and explains differences between Windows and UNIX platforms.

You run CAS components in any of the following ways:

- In the Endeca CAS Service
- Using command-line utilities
- From the CAS Console for Oracle Endeca Workbench
- Programmatically from the CAS APIs (For details, see the *Endeca CAS API Guide*.)

Running CAS components in the Endeca CAS Service

As discussed in the introduction, the Endeca CAS Service is a container process that runs the CAS components such as the CAS Server, the Component Instance Manager, and one or more Record Store instances. In a typical implementation, running the CAS Service is the recommended way to run CAS components.

Running CAS components using the command-line utilities

The Content Acquisition System provides several convenience utilities so you can run any component manually from a command prompt, if you choose to. These utilities include the following:

- CAS Server Command-line Utility
- Component Instance Manager Command-line Utility
- Record Store Command-line Utility

Each utility is described in this guide.

Running CAS components from the CAS Console for Oracle Endeca Workbench

As a user adds data sources, crawls data sources, and so on, the CAS Console runs the appropriate CAS component.

Running the Endeca CAS Service on Windows

On Windows, the Endeca CAS Service is automatically started as part of the installation process. Oracle recommends that you start and stop the service from the Microsoft Services console.

You may optionally choose to start the Endeca CAS Service on Windows using the `cas-service.bat` script in `<install path>\CAS\<version>\bin` and stop it using `cas-service-shutdown.bat`.

Running the Endeca CAS Service on UNIX

On UNIX, you run the Endeca CAS Service using the scripts in `<install path>/CAS/<version>/bin`. You start the service with `cas-service.sh` or via the `inittab` and stop it with `cas-service-shutdown.sh`. Oracle recommends using the `inittab` in production environments.

Restarting the Endeca CAS Service

On either platform, you can leave the service running as a background process. The only time you must restart the service is if you modify any of the configuration files in `<install path>\CAS\<version>\conf`. For example, you might change the CAS Service logging configuration and therefore have to restart the service.

Running the Endeca CAS Service from the Windows Services console

On Windows, the Endeca CAS Service is registered as a Windows Service and starts automatically when the operating system starts. This is the recommended way of running CAS on Windows.

If you have changed any of the CAS configuration files, you can stop and restart the Endeca CAS service, using the Windows Console, for those changes to take effect.

Note: The CAS Service can be slow to startup if it contains large Record Store instances and has not been cleanly shutdown.

To run the Endeca CAS Service on Windows:

1. From the Windows **Start** menu, go to **Control Panel** > **Administrative Tools** > **Services**.
2. Locate the Endeca CAS Service from the list and right-click it.
3. From the context menu, select **Stop**, **Restart**, or **Start** as necessary.
4. Exit the Windows Service console.

Starting the Endeca CAS Service from a command prompt

In UNIX environments, you start the Endeca CAS Service from a command prompt. In Windows enviroments, you can start the service from a command prompt, but it is optional.

The only prerequisite to this task is that the CAS software must be installed. This means that the `cas-service.bat` (for Windows) or `cas-service.sh` (for UNIX) scripts must be available. Note that for the UNIX `cas-service.sh` script, you must have Execute rights to the file.

Note: The CAS Service can be slow to startup if it contains large Record Store instances and has not been cleanly shutdown.

To start the Endeca CAS Service from a command prompt:

1. Open a command prompt and navigate to the `bin` directory.
   In a default installation on Windows, this is `C:\Endeca\CAS\<version>\bin`.
2. Run the `cas-service.bat` script (for Windows) or `cas-service.sh` script (for UNIX).
   For example, in a default installation on a Windows machine, the command is as follows:
   ```
   C:\Endeca\CAS\11.0.0\bin>cas-service
   ```
   This command starts the CAS Service on the default port 8500 with a workspace directory of `Endeca\CAS\workspace`.
   Note that you do not see any startup messages. All messages are sent to the CAS Service log (by default, `workspace\logs\cas-service.log`). However, if there is an error in setting up logging, all messages are sent to the console.
3. Verify that the server is running by opening a Web browser and entering a URL with the CAS Service host and port number followed by `cas/?wsdl`. For example, in a default installation:
   ```
   http://localhost:8500/cas/?wsdl
   ```
   You see the CasCrawlerService WSDL in the Web browser window, which indicates that the Endeca CAS Service is running.

The workspace directory has these sub-directories:

- `state` contains a `crawlerDb` subdirectory that stores a unified crawl history (that is, one crawl history for all crawls defined for this CAS Server) and contains an `output` directory which is the default location for the crawl output files.
- `logs` contains the `cas-service.log` log file for the CAS Service.

Command-line flags to CAS Service

The Endeca CAS Service startup script has an optional Java Virtual Machine (-JVM) flag.

Note: Flag names are case sensitive.

cas-service Flag	Flag Argument
`-JVM`	Allows arguments on the command line to be passed to the JVM. If this flag is used, any arguments after it are passed to the CAS Service and any arguments afterwards are appended to those passed to the JVM. Note that on Windows machines, the flag parameters should be quoted if they contain equal signs. Optional.

Specifying JVM arguments

To pass arguments to the JVM, you can use the `-JVM` script flag. For example, assume you want to override the default maximum heap size setting of 1024 MB with a setting of 2048 MB. The command line is as follows:

```
cas-service -JVM -Xmx2048m
```

Keep in mind that this flag must be the last flag on the command line, because any arguments that follow it are appended to those passed to the JVM.

Stopping the Endeca CAS Service from a command prompt

In UNIX environments, you stop the Endeca CAS Service from a command prompt. In Windows enviroments, you can optionally stop the service from a command prompt, or use the Windows Services console (recommended).

Note: If you start the service using the command prompt you should also stop the service using the command prompt.

To stop the Endeca CAS Service:

1. Open a command prompt and navigate to the `bin` directory.
   In a default installation on Windows, this is `C:\Endeca\CAS\<version>\bin`.
2. Run the `cas-service-shutdown.bat` script (for Windows) or `cas-service-shutdown.sh` script (for UNIX).
   For example, in a default installation on a Windows machine, the command is as follows:

   ```
   C:\Endeca\CAS\11.0.0\bin>cas-service-shutdown
   ```

Chapter 16

Backing up and restoring CAS

This section describes how to back up and restore CAS state, crawl configurations, and Record Store instance data.

Coordinating backups and restore operations

Online backups are often done more frequently than offline backups. For example, you might perform a full offline backup once a week, and perform smaller online back ups on a daily basis. So when you restore the backup, you would have to first restore the weekly offline backup and then the series of daily online backups.

Online backup and restore operations

The administration tasks in this section can be performed while the Endeca CAS Service is running. The tasks are generally more focused and specific than the tasks you can perform while the Endeca CAS Service is offline. And there are some elements of CAS that you cannot back up online, for example, you cannot back up crawl histories while the Endeca CAS Service is running.

Backing up crawl configurations

You back up crawl configurations using the CAS Server Command line Utility.

To back up crawl configurations:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin` (for Windows), or `<install path>/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `getAllCrawls` task with the `-f` (or `--file`) flag and the name of the XML file to write the crawl configurations to.
   For example:

   ```
   C:\Endeca\CAS\11.0.0\bin>cas-cmd getAllCrawls -f C:\tmp\backupconfig.xml
   ```

   Note that password configuration properties are not stored in the crawl configuration.

Backing up the last generation of Endeca records

This procedure describes how to back up the last generation of Endeca records in a Record Store instance and back up the corresponding configuration for the Record Store instance. This task does not describe backing up multiple generations or deltas between generations.

To back up the last generation of Endeca records:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin` (for Windows), or `<install path>/CAS/<version>/bin` (for UNIX).
2. To list the available Record Store instances in the Content Acquisition System, type `component-manager-cmd.bat` (for Windows), or `component-manager-cmd.sh` (for UNIX) and specify the `listComponents` task.
   For example:
   ```
   C:\Endeca\CAS\11.0.0\bin>component-manager-cmd.bat list-components
   NAME            TYPE            STATUS
   Test         RecordStore     RUNNING
   ```
3. From the list, identify the Record Store instance that contains the generation of records you want to back up.
4. Type `recordstore-cmd.bat` (for Windows), or `recordstore-cmd.sh` (for UNIX) and specify the `read-baseline` task with the `-a` (or `--instanceName`) flag and the name of a Record Store instance and also the `-f` (or `--file`) flag and the pathname of the file to which the Endeca records will be output.

   Valid extensions for the file are `.xml` (for an XML format) and `.bin` (for a binary format); the file can also have an additional, optional `.gz` extension if it is a compressed file. Endeca recommends using `.bin.gz` because it is the most compact format.

   For example:
   ```
   C:\Endeca\CAS\11.0.0\bin>recordstore-cmd.bat read-baseline -a Test -f
   C:\tmp\RSIbackup.xml
   ```

   The `read-baseline` operation writes the last generation of Endeca records. It does not write all generations.
5. To back up the configuration for a Record Store instance, type `recordstore-cmd.bat` (for Windows), or `recordstore-cmd.sh` (for UNIX) and specify the `get-configuration` task with the `-a` (or `--instanceName`) flag and the name of a Record Store instance and also the `-f` (or `--file`) flag and the XML file name where you want to save the configuration settings.
   For example:
   ```
   C:\Endeca\CAS\11.0.0\bin>recordstore-cmd.bat get-configuration -a Test -f
   C:\tmp\RSIbackup_configfile.xml
   ```

Backing up dimension value Id mappings

You back up dimension value Id mappings to a CSV file using the CAS Server Command-line Utility.

To back up dimension value Id mappings:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin` (for Windows), or `<install path>/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `exportDimensionValueIdMappings` task with the following:

   a) `-f` (or `--file`) flag and the path of the CSV file to write mappings to. (The file is encoded as UTF-8.)

b) `-m` (or `--dimension_value_id_manager`) flag and the name of the Dimension Value Id Manager to export from.

For example:

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd exportDimensionValueIdMappings -m dvalmgr -f
 C:\tmp\mappings.csv
```

Restoring crawl configurations

You restore crawl configurations using the CAS Server Command line Utility.

To restore crawl configurations:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin` (for Windows), or `<install path>/CAS/<version>/bin` (for UNIX).
2. If you are restoring into a system that has an older version of the crawl configuration, type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `updateCrawls` task with the `-f` (or `--file`) flag and the name of the XML file that contains crawl configurations.
   For example:
   ```
   C:\Endeca\CAS\11.0.0\bin>cas-cmd updateCrawls -f C:\tmp\backupconfig.xml
   Updated crawl Test
   ```
3. If you are restoring into a system that does not have the crawl configuration, type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `createCrawls` task with the `-f` (or `--file`) flag and the name of the XML file that contains crawl configurations.
   For example:
   ```
   C:\Endeca\CAS\11.0.0\bin>cas-cmd createCrawls -f C:\tmp\backupconfig.xml
   Updated crawl Test
   ```

Restoring the last generation of Endeca records

This task describes restoring one generation of baseline data into a Record Store instance and restoring the corresponding configuration file for the Record Store instance. This task does not describe restoring multiple generations or deltas between generations.

To restore the last generation of Endeca records:

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin` (for Windows), or `<install path>/CAS/<version>/bin` (for UNIX).
2. Create a new empty Record Store instance by typing `component-manager-cmd.bat` (for Windows), or `component-manager-cmd.sh` (for UNIX) and specify the `create-component` task with the `-t` option with an argument of `RecordStore`, and the `-n` option with a Record Store instance name of your choice.

   You need to repeat this step if your crawl configuration contains multiple Record Store instances. Also you should ensure that the name of each Record Store instance coordinates with the `crawlId`. For example, if you have a `crawlId` of `Test`, you create a Record Store instance named `Test`.

   For example:
   ```
   C:\Endeca\CAS\11.0.0\bin>component-manager-cmd.bat create-component
    -n Test -t RecordStore
   ```
3. Restore the configuration file for a Record Store instance by typing `recordstore-cmd.bat` (for Windows), or `recordstore-cmd.sh` (for UNIX) and specify the `set-configuration` task with the `-a` (or

`--instanceName`) flag and the name of a Record Store instance and also the `-f` (or `--file`) flag and the XML file name that contains the configuration settings.
For example:

```
C:\Endeca\CAS\11.0.0\bin>recordstore-cmd.bat set-configuration -a Test -f
C:\tmp\RSIbackup_configfile.xml
Successfully set recordstore configuration.
```

4. Write the data into the Record Store instance by typing `recordstore-cmd.bat` (for Windows), or `recordstore-cmd.sh` (for UNIX) and specify the `write` task with the `-a` (or `--instanceName`) flag and the name of a Record Store instance and also the `-f` (or `--file`) flag and the pathname of the file that contains the Endeca records.
   For example:

```
C:\Endeca\CAS\11.0.0\bin>recordstore-cmd.bat write -a Test -f C:\tmp\RSIback¬
up.xml
Wrote 2190 records.
```

Restoring dimension value Id mappings

You restore dimension value Id mappings using the CAS Server Command line-Utility. The restore process completely replaces all dimension value Id mappings stored in the specified Dimension Value Id Manager.

To restore dimension value Id mappings :

1. Start a command prompt and navigate to `<install path>\CAS\<version>\bin` (for Windows), or `<install path>/CAS/<version>/bin` (for UNIX).
2. Type `cas-cmd.bat` (for Windows), or `cas-cmd.sh` (for UNIX) and specify the `importDimensionValueIdMappings` task with the following:
   - `-f` (or `--file`) flag and the name of the CSV file to read mappings from
   - `-m` (or `--dimension_value_id_manager`) flag and the name of the Dimension Value Id Manager to import into.

   For example:

```
C:\Endeca\CAS\11.0.0\bin>cas-cmd importDimensionValueIdMappings -m dvalmgr -f
 C:\tmp\mappings.csv
```

Offline backup and restore operations

The administration tasks in this section can only be performed while the Endeca CAS Service is stopped. Once you shutdown the service, you can back up CAS to preserve nearly all of its state.

The CAS state includes:

- Record Store instance data and configuration
- Crawl configurations
- Crawl history
- Dimension value Id mappings

The CAS state does not include:

- Alternate data directories for Record Store instance data (configured via the `dataDirectory` property).

- State information for custom data sources or manipulators that write state to other locations. (Extensions developed using the CAS Extension API can write to any location a developer choses.)

Backing up CAS state

This task describes how to back up CAS state. CAS stores its state in `<install path>\CAS\workspace\state`.

To back up CAS state:

1. Stop the Endeca CAS Service.
2. On the machine running the CAS Service, navigate to `<install path>\CAS\workspace\state`.
3. Copy the `state` directory to a location outside the CAS installation.
4. Restart the Endeca CAS Service.

Restoring CAS state

This task describes how to restore CAS state information.

To restore CAS state:

1. Stop the Endeca CAS Service.
2. Locate the `cas` directory that you previously backed up. This is typically in a location outside the CAS installation.
3. Navigate to `<install path>\CAS\Workspace\state`.
4. Copy the`state` directory that you previously backed up into `<install path>\CAS\Workspace`.
5. Restart the Endeca CAS Service.

Chapter 17

Configuring logging

This section describes how to configure logging for CAS.

Configuring logging for CAS components and command-line utilities

You can change the default logging configuration of the Endeca CAS Service and any CAS components you run from the command line.

Log location and rolling

CAS writes log files to the `<install path>\CAS\workspace\logs` directory. If desired, you can reconfigure CAS to write log files to another location.

CAS rolls a log file once it exceeds 100MB, and the CAS Service keeps 10 backups of its log.

Note: If you delete the log file, the Endeca CAS Service recreates the log only when you restart the service. So it is possible to delete the log, run a crawl, and not have logging information if you did not restart the service.

Configuration files for CAS components

The `<install path>\CAS\workspace\conf` directory contains the following logging configuration files:

- `cas-cmd.log4j.properties` - configures logging for the CAS Server Command-line Utility
- `cas-service-log4j.properties` - configures logging for the Endeca CAS Service
- `component-manager-cmd.log4j.properties` - configures logging for the Component Instance Manager Command-line Utility
- `recordstore-cmd.log4j.properties` - configures logging for the Record Store Command-line Utility

Logging options and levels

You can re-configure log locations, log file size, log file encoding, log pattern, and logging message levels.

Logging levels can be set to any of the following :

- `DEBUG` designates fine-grained informational events that are most useful to debug Record Store problems.
- `INFO` designates informational messages that highlight the progress of Record Store operations at a coarse-grained level.

- WARN designates potentially harmful situations.
- ERROR designates error events that might still allow the Record Store to continue running.
- FATAL designates very severe error events that will presumably lead the Record Store to abort.
- OFF has the highest possible rank and is intended to turn off logging.

These levels allow you to monitor events of interest at the appropriate granularity. When you are initially setting up your Record Store implementation, you might want to use the DEBUG level to get all messages, and change to a less verbose level in production.

Log file encoding

CAS produces log files encoded as UTF-8.

Setting log properties for troubleshooting CMS data source issues

You can set logging properties that may help determine the causes of connection, authentication, and request/response time issues between the CAS and the CMS provider.

To configure logging properties for troubleshooting CMS data source issues:

1. In a text editor, open `cas-service-log4j.properties`.
2. Add the following lines to the file:

```
log4j.logger.org.apache.axis.client=DEBUG
log4j.logger.httpclient.wire=DEBUG
log4j.logger.org.apache.commons.httpclient=DEBUG
```

Excluding failed records from the CAS Service log file

If a record fails during a crawl, the CAS Server discards the failed record and writes a truncated version of the record to the `cas-service.log` file. If you do not want the CAS Server to write any information about failed records to the log file, you can disable logging for failed records by uncommenting a setting in the `cas-service-log4j.properties` file.

To exclude failed records from the CAS Service log file:

1. Stop the Endeca CAS Service.
2. Navigate to the `CAS\workspace\conf` directory.
3. In a text editor, open `cas-service-log4j.properties`.
4. Uncomment the line containing the `log4j.logger.com.endeca.itl.executor.ErrorChannelImpl` setting.
5. Save and close the `cas-service-log4j.properties` file.
6. Start the Endeca CAS Service.

Once you uncomment the setting, the CAS Server does not write any information about failed records to the log file. However, failed records are still counted as metrics under `FAILED_RECORDS`.

Enabling log timing information for crawl processing steps

You can enable a logging setting in `cas-service-log4j.properties` that instructs CAS to write log timing information for each processing step of a crawl. This additional logging information is especially useful for troubleshooting performance issues.

1. In a text editor, open `<install path>\CAS\workspace\conf\cas-service-log4j.properties`.
2. Un-comment the following line in the file:
   ```
   log4j.logger.com.endeca.itl.executor.ProcessorTaskTiming=DEBUG
   ```
3. Save and close the file.
4. Restart the Endeca CAS Service.

The next time you run a crawl you will get additional logging information similar to the following:

```
Processor Task Timing

IncrementalDataSourceProcessor-414611937: (Hits=1, Value=8542.280 ms,
Time=15:11:37,489)
MdexOutputSink-1898864883(processRecord): (Total=7413.427 ms, Avg=0.835 ms,
Hits=8877, StdDev=9.526 ms, Min=0.001 ms, Max=659.003 ms, FirstTimed=15:11:36,802,
 LastTimed=15:11:40,078)
SplittingFilterProcessor-1235020019(processRecord): (Total=3250.518 ms, Avg=0.366
 ms, Hits=8877, StdDev=0.711 ms, Min=0.010 ms, Max=26.920 ms, First¬
Timed=15:11:29,626, LastTimed=15:11:37,457)
ArchiveExpandProcessor-1134860470(processRecord): (Total=2104.446 ms, Avg=0.237
ms, Hits=8877, StdDev=0.843 ms, Min=0.004 ms, Max=44.004 ms, First¬
Timed=15:11:29,595, LastTimed=15:11:37,457)
PropertyRemover-92265517(processRecord): (Total=1849.963 ms, Avg=0.208 ms,
Hits=8877, StdDev=0.716 ms, Min=0.003 ms, Max=27.465 ms, FirstTimed=15:11:29,595,
 LastTimed=15:11:37,457)
MdexOutputSink-1898864883(notifyInputClosed): (Total=598.802 ms, Avg=299.401 ms,
 Hits=2, StdDev=391.645 ms, Min=22.466 ms, Max=576.336 ms, FirstTimed=15:11:38,206,
 LastTimed=15:11:40,094)
PropertyRemover-92265517(notifyInputClosed): (Hits=1, Value=0.544 ms,
Time=15:11:37,489)
ArchiveExpandProcessor-1134860470(notifyInputClosed): (Total=0.330 ms, Avg=0.165
 ms, Hits=2, StdDev=0.141 ms, Min=0.065 ms, Max=0.265 ms, FirstTimed=15:11:37,489,
 LastTimed=15:11:37,489)
SplittingFilterProcessor-1235020019(notifyInputClosed): (Hits=1, Value=0.012 ms,
 Time=15:11:37,489)
```

Examining the Endeca CAS Service log

The Endeca CAS Service logs messages for all CAS components and crawls in the `cas-service.log` file.

Location of the CAS Service log

The Endeca CAS Server has one (and only one) log, regardless of how many crawls have been configured. The log is named `cas-service.log` and is located in the `logs` directory in the CAS workspace directory. If you are using the default workspace directory name, the pathname of the log file is similar to this:

```
C:\Endeca\CAS\workspace\logs\cas-service.log
```

Format of log entries

The log contains two types of log entries:

- CAS component log entries, which are entries that pertain to starting and stopping CAS components.
- crawl log entries, which are entries that pertain to a specific crawl.

By default, crawl log entries have the format:

```
yyyy-MM-dd HH:mm:ss,SSS logLevel [component] [thread name] class: <message>
```

where:

- *yy-MM-dd HH-mm-ss* is the timestamp of the entry. You can change the format by editing the `cas-server.log4j.properties` file.
- *logLevel* is the log level of the entry, such as INFO or FATAL.
- *component* is cas (for the crawl manager), ComponentInstanceManager, or instance name for Record Stores or DVal ID Managers.
- *thread name* is the name of the processing thread for the message.
- *message* is the message returned by a CAS Server module.

Enabling crawl statistics

If a crawl log level is set to INFO, TRACE, or DEBUG, the crawl statistics are entered as INFO entries in the log when the crawl finishes, as in this example (timestamps and log levels are omitted for ease of reading):

```
Crawl Mode = FULL_CRAWL (MetricsReport)
Crawl Stop Cause = Completed (MetricsReport)
Directories Filtered from Archives = 0 (MetricsReport)
Directories Filtered = 0 (MetricsReport)
Total Records Output = 423 (MetricsReport)
Files Filtered from Archives = 124 (MetricsReport)
Directories Crawled Not from Archives = 55 (MetricsReport)
Documents Unsuccessfully Converted = 9 (MetricsReport)
Files Crawled from Archives = 65 (MetricsReport)
Files Crawled Not from Archives = 285 (MetricsReport)
Delete Records Output = 0 (MetricsReport)
Files Filtered Not from Archives = 51 (MetricsReport)
Directories Crawled = 73 (MetricsReport)
Directories Filtered Not from Archives = 0 (MetricsReport)
Documents Converted = 333 (MetricsReport)
Files Crawled = 350 (MetricsReport)
Documents Converted After Retry = 0 (MetricsReport)
New or Updated Records Output = 423 (MetricsReport)
Directories Crawled from Archives = 18 (MetricsReport)
Files Filtered = 175 (MetricsReport)
Crawl Seconds = 71 (MetricsReport)
Start Time = 5/23/08 9:23:59 AM EDT (MetricsReport)
End Time = 5/23/08 9:25:10 AM EDT (MetricsReport)
```

Note that for incremental crawls, the `Delete Records Output` statistic is also included and indicates how many files were deleted from the previous crawl. An Endeca record is created for each deleted file; the record will have the `Endeca.Action` property set to DELETE.

The `Crawl Stop Cause` statistic has one of the following values:

- Completed
- Failed
- Aborted

If a crawl fails, the `Crawl Failure Reason` statistic provides a message from the CAS Server explaining the failure.

Keep in mind that if the log is too verbose (thus making it more difficult to find errors), you can change the log level of the crawl. The default log level is INFO.

The CAS logging configuration file is `cas-service-log4j.properties` and is located in the `<install path>\CAS\workspace\conf` directory. You can also change the log level on a per-crawl basis using the CAS Console, the CAS API, or the CAS command-line utilities.

Chapter 18

Tips and troubleshooting CAS

This section provides tips and miscellaneous troubleshooting information about the Content Acquisition System.

Fixing crawl performance issues

This topic lists performance issues you many encounter when running a CAS crawl and provides ways to address the issues.

Periodic crawl performance problems caused by defragmenting the crawl history database

CAS has an embedded database that stores crawl history. With large crawls, the database can grow until it approaches its maximum file size limit. However, before the database reaches its maximum size, CAS defragments it, in order to reduce the overall file size of the database. Any crawls that are running will slow down while the defragmentation process is running.

If you notice a crawl that is taking a long time, you can determine whether a defragmentation process is causing the issue by checking the CAS service log for the following error:

`com.endeca.itl.ItlRuntimeException: java.sql.SQLException: Data File size limit is reached`.

If you see this error frequently, you can work with Oracle Customer Support to adjust the maximum file size of the crawl history database and adjust the frequency at which CAS runs the defragmentation process. For details, contact Oracle Customer Support.

Modifying the CAS Server connection information for the CAS Console

You first specified this information during the installation procedure. If you need to run the CAS Server on a different machine, you can change the CAS Server connection information through the `casconsole.properties` file.

To modify your CAS Server connection:

1. Navigate to `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF/conf` (on UNIX).
   In a default installation, this is `C:\Endeca\Workbench\workspace` (on Windows) or `usr/local/Endeca/Workbench/workspace` (on UNIX).

2. Open `casconsole.properties`.
3. Modify the entries for `Default cas server host` and `Default cas server port` as needed.
4. Save and close the file.

Modifying the CAS Service temporary directory

By default, the Endeca CAS Service temporary directory is set to `<install path>\CAS\workspace\temp` (on Windows) and `<install path>/CAS/workspace/temp` (on UNIX). If necessary, you can modify this path by changing the `java.io.tmpdir` system property in the Endeca CAS Service script.

To modify the CAS Service temporary directory:

1. Stop the Endeca CAS Service.
2. Navigate to `<install path>\CAS\version\bin`.
3. If you are running the Endeca CAS Service manually, open `cas-service` (either .bat or .sh depending on your platform) in a text editor.
4. If you are running the Endeca CAS Service automatically as a Windows service, open `cas-service-wrapper.conf` in a text editor.
5. Locate the `Djava.io.tmpdir` argument and modify the value of the path as necessary.
6. Save and close the file.
7. Re-start the Endeca CAS Service.

Responding to a "Too many open files" error

On UNIX, you may get a “Too many open files” error if you are crawling several data sources simultaneously.

The relevant line in the error’s stack trace is the following:

```
Caused by: java.io.FileNotFoundException: /localdisk/jsmith/ende¬
ca/CAS/workspace/state/test_data_multiseeds/data/dictionary/seg0/c3a1.dat (Too
many open files)
```

The error occurs because the operating system has reached the per-process limit for the number of files the process can have open at once.

To resolve this problem, you can increase the number of file handles available. For more information about how to increase the number of available file handles, refer to the documentation for your operating system.

Note: There is no single recommended range of file handles values that will fit all situations. File/socket requirements can depend on a number of metrics, such as processes managed, nodes, files transferred, and system status queries. Therefore, determining a new limit experimentally, through trial and error, is the simplest resolution.

Setting the group entry size

You can change the group entry size default setting.

On UNIX systems, the crawler relies on the group and passwd databases to generate native properties for files. Because there is no limit to the size of the entries in these databases, the default sizes may be too small for some systems.

For example, if the size of a group entry is too large, the following message is written to the log:

```
The group's entry in the group database is too large,
consider setting the com.endeca.itl.group.size property.
```

You change group entry size by using the Java `-D` option as a parameter to the Java Virtual Machine (JVM), as follows:

```
-Dcom.endeca.itl.group.size=2048
```

Note that the `2048` parameter is in bytes.

To pass this parameter to the JVM, use the `-JVM` flag when you run the startup script.

Keep in mind that the `-JVM` flag must be the last flag on the command line.

This type of error is more likely to occur with entries in the group database, rather than the passwd database. If, however, your crawl encounters problems with the passwd database, there is also a passwd entry property:

```
com.endeca.itl.passwd.size
```

Appendix A

Sample crawl configuration files

This section provides a group of sample crawl configuration files to use as a starting point for your own crawl configuration files or simply as a reference to better understand configuration options.

Common properties for crawl configurations

The following table lists crawl configuration properties that are common to all crawl configurations, including file system crawls, CMS crawls, custom crawls, and so on.

In addition to the common properties in the table below, a crawl configuration also includes configuration properties specific to its data source type. For example, a JDBC crawl configuration has JDBC connection properties.

To determine configuration properties a specific data source, run the `getModuleSpec` task of the `cas-cmd` utility.

Configuration element	Description
`crawlId`	Specifies a unique name to distinguish the crawl from others in CAS. The `crawlId` can contain alphanumeric characters, underscores, dashes, and periods. All other characters are invalid for a `crawlId`.
`unavailableIncrementalSwitchesToFull¬Crawl`	Specifies a Boolean value to indicate whether CAS should switch from running an incremental crawl to running a full crawl in cases where it is not possible to run an incremental crawl. A value of `true` instructs CAS to run a full crawl if it is not possible to run an incremental crawl. A value of `false` instructs CAS to abort the incremental crawl and throw a `FullCrawlRequiredException` indicating why the incremental crawl could not run. The default value depends on the `outputConfig` type. If the `outputConfig` is set to `Record Store`, then the default value is `true`. If the `outputConfig` is set to `com.endeca.cas.out¬put.Mdex`, then the default value is `false`.

Configuration element	Description
	If the `outputConfig` is set to `File System`, then the default value is `false`.
`crawlThreads`	Specifies the maximum threads available to the CAS Service. The default number of threads is one more than the number of CPUs of the machine running the CAS Service. For example, if the CAS Service is running on a machine with four CPUs, the default number of threads is five.
`textExtractionConfig`	Specifies whether document conversion is enabled. If `textExtractionConfig` has a value of true and contains additional document conversion settings, then CAS performs document conversion and stores the converted text as a property on the Endeca record.
`manipulatorConfigs`	Specifies any number of manipulators within a crawl configuration. If one or more `manipulatorConfig` elements are present, CAS passes each record to each manipulator for processing according to its `manipulator¬Config` settings. Manipulators execute in the order in which they are nested within `manip¬ulatorConfigs`.

Sample configuration for a file system data source

This sample shows a crawl configuration of a file system data source with no manipulators and no filters.

Much of the configuration is specified in key/value pairs within a `moduleProperty` property element. To determine the values for configuration properties, run the `getModuleSpec` task of the `cas-cmd` utility.

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

<configurations xmlns="http://endeca.com/itl/cas/2011-12">
    <crawlConfig>
      <crawlId>
        <id>FileCrawl</id>
      </crawlId>
      <unavailableIncrementalSwitchesToFullCrawl>false</unavailableIncremental¬
SwitchesToFullCrawl>
      <crawlThreads>3</crawlThreads>
      <sourceConfig>
        <moduleId>
          <id>File System</id>
        </moduleId>
        <moduleProperties>
          <moduleProperty>
            <key>expandArchives</key>
            <value>false</value>
          </moduleProperty>
```

```
          <moduleProperty>
            <key>gatherNativeFileProperties</key>
            <value>true</value>
          </moduleProperty>
          <moduleProperty>
            <key>seeds</key>
            <value>C:\tmp\itldocset</value>
            <value>C:\tmp\iapdocset</value>
            <value>C:\tmp\mdexdocset</value>
          </moduleProperty>
        </moduleProperties>
        <excludeFilters/>
        <includeFilters/>
      </sourceConfig>
      <textExtractionConfig>
        <enabled>true</enabled>
        <makeLocalCopy>true</makeLocalCopy>
        <timeout>90</timeout>
      </textExtractionConfig>
      <manipulatorConfigs/>
      <outputConfig>
        <moduleId>
          <id>File System</id>
        </moduleId>
        <moduleProperties>
          <moduleProperty>
            <key>outputXml</key>
            <value>true</value>
          </moduleProperty>
          <moduleProperty>
            <key>outputCompressed</key>
            <value>false</value>
          </moduleProperty>
          <moduleProperty>
            <key>outputPrefix</key>
            <value>CrawlerOutput</value>
          </moduleProperty>
          <moduleProperty>
            <key>outputDirectory</key>
            <value>C:\tmp</value>
          </moduleProperty>
        </moduleProperties>
      </outputConfig>
    </crawlConfig>
</configurations>
```

Sample configuration for a Record Store Merger data source

This sample shows a crawl configuration of a Record Store Merger data source.

Much of the configuration is specified in key/value pairs within a `moduleProperty` property element. To determine the values for configuration properties, run the `getModuleSpec` task of the `cas-cmd` utility.

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

<configurations xmlns="http://endeca.com/itl/cas/2011-12">
  <crawlConfig>
    <crawlId>
```

```
      <id>ebizsampleapp-last-mile-crawl</id>
    </crawlId>
    <unavailableIncrementalSwitchesToFullCrawl>false</unavailableIncremental¬
SwitchesToFullCrawl>
    <crawlThreads>5</crawlThreads>
    <sourceConfig>
      <moduleId>
        <id>com.endeca.cas.source.RecordStoreMerger</id>
      </moduleId>
      <moduleProperties>
        <moduleProperty>
          <key>dataRecordStores</key>
          <value>ebizsampleapp-products</value>
        </moduleProperty>
        <moduleProperty>
           <key>dimensionValueRecordStores</key>
               <value>ebizsampleapp-category-dimension</value>
               <value>ebizsampleapp-trigger-dimensions</value>
        </moduleProperty>
        <moduleProperty>
          <key>isPortSsl</key>
          <value>false</value>
        </moduleProperty>
      </moduleProperties>
    </sourceConfig>
    <manipulatorConfigs/>
    <outputConfig>
      <moduleId>
        <id>com.endeca.cas.output.Mdex</id>
      </moduleId>
      <moduleProperties>
        <moduleProperty>
          <key>configRepositorySite</key>
          <value>ebizsampleapp</value>
        </moduleProperty>
        <moduleProperty>
          <key>inputDirectory</key>
          <value>C:/Endeca/apps/ebizsampleapp/data/complete_index_config</value>

        </moduleProperty>
        <moduleProperty>
          <key>outputDirectory</key>
          <value>C:/Endeca/apps/ebizsampleapp/data/dgidx_input</value>
        </moduleProperty>
        <moduleProperty>
          <key>dimensionValueIdManagerInstanceName</key>
          <value>ebizsampleapp-dimension-value-id-manager</value>
        </moduleProperty>
      </moduleProperties>
    </outputConfig>
  </crawlConfig>
</configurations>
```

Sample configuration for a manipulator

This sample shows a crawl configuration that includes a manipulator.

Much of the configuration is specified in key/value pairs within a `moduleProperty` property element. To determine the values for configuration properties, run the `getModuleSpec` task of the `cas-cmd` utility.

The sample omits the configuration of a data source. In other words, the sample does not contain the `sourceConfig` or `outputConfig` of a `crawlConfig`. The example illustrates the `manipulatorConfigs` elements. In particular, it shows two Substring Manipulators. The first creates a property of 20 characters called `Short.Truncated.Text`, and the second manipulator creates a property of 40 characters called `Medium.Truncated.Text`.

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

<configurations xmlns="http://endeca.com/itl/cas/2011-12">
    <crawlConfig>

    ...

       <manipulatorConfigs>
          <manipulatorConfig>
             <moduleId>
                <id>com.endeca.cas.extension.sample.manipulator.substring.Sub¬
stringManipulator</id>
             </moduleId>
             <moduleProperties>
                <moduleProperty>
                   <key>sourceProperty</key>
                   <value>Endeca.Document.Text</value>
                </moduleProperty>
                <moduleProperty>
                   <key>targetProperty</key>
                   <value>Short.Truncated.Text</value>
                </moduleProperty>
                <moduleProperty>
                   <key>length</key>
                   <value>20</value>
                </moduleProperty>
             </moduleProperties>
             <id>Create short truncated text property</id>
          </manipulatorConfig>
          <manipulatorConfig>
             <moduleId>
                <id>com.endeca.cas.extension.sample.manipulator.substring.Sub¬
stringManipulator</id>
             </moduleId>
             <moduleProperties>
                <moduleProperty>
                   <key>sourceProperty</key>
                   <value>Endeca.Document.Text</value>
                </moduleProperty>
                <moduleProperty>
                   <key>targetProperty</key>
                   <value>Medium.Truncated.Text</value>
                </moduleProperty>
                <moduleProperty>
                   <key>length</key>
                   <value>40</value>
                </moduleProperty>
             </moduleProperties>
             <id>Create_Medium_Text_Property</id>
             <enabled>true</enabled>
         </manipulatorConfig>
       </manipulatorConfigs>
```

```
      ...

    </crawlConfig>
</configurations>
```

Sample configuration for writing output to a Record Store instance

This sample shows a crawl configuration that writes output to a Record Store instance.

The configuration change is on a per crawl basis. A crawl configuration file used with the command-line utility can have many crawls defined and each crawl can have different output configuration options.

The crawl configuration file requires `<outputConfig>` settings that specify `Record Store` as the `<moduleId>`. The `Record Store` value is the main difference between configuring a data source to write to an output file versus writing to a Record Store instance.

The other sub-elements of `<outputConfig>` configure additional details of a Record Store instance, such as the host and port of the Endeca CAS Service running the Record Store instance, whether to use SSL when connecting to the Record Store instance, and so on. This XML snippet shows the `<moduleId>` for a Record Store instance.

```
...
     <outputConfig>
        <moduleId>
          <id>Record Store</id>
        </moduleId>
...
```

The following example configuration file shows the required configuration for one crawl named `itldocset` that writes records to a Record Store instance named `itldocset`. It is a simple example that does not contain any filters. You can use this example as the basis for your own configuration file in the procedure below.

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

<configurations xmlns="http://endeca.com/itl/cas/2011-12">
    <crawlConfig>
      <crawlId>
        <id>itldocset</id>
      </crawlId>
      <unavailableIncrementalSwitchesToFullCrawl>true</unavailableIncremental¬
SwitchesToFullCrawl>
      <crawlThreads>3</crawlThreads>
      <sourceConfig>
        <moduleId>
          <id>File System</id>
        </moduleId>
        <moduleProperties>
          <moduleProperty>
            <key>expandArchives</key>
            <value>true</value>
          </moduleProperty>
          <moduleProperty>
            <key>gatherNativeFileProperties</key>
            <value>true</value>
          </moduleProperty>
          <moduleProperty>
```

```
            <key>seeds</key>
            <value>C:\tmp\itldocset</value>
          </moduleProperty>
        </moduleProperties>
        <excludeFilters/>
        <includeFilters/>
      </sourceConfig>
      <textExtractionConfig>
        <enabled>true</enabled>
        <makeLocalCopy>false</makeLocalCopy>
        <timeout>90</timeout>
      </textExtractionConfig>
      <manipulatorConfigs/>
      <outputConfig>
        <moduleId>
          <id>Record Store</id>
        </moduleId>
        <moduleProperties>
          <moduleProperty>
            <key>isPortSsl</key>
            <value>false</value>
          </moduleProperty>
          <moduleProperty>
            <key>host</key>
            <value>hostname.domainname.com</value>
          </moduleProperty>
          <moduleProperty>
            <key>port</key>
            <value>8500</value>
          </moduleProperty>
          <moduleProperty>
            <key>instanceName</key>
            <value>itldocset</value>
          </moduleProperty>
          <moduleProperty>
            <key>isManaged</key>
            <value>true</value>
          </moduleProperty>
        </moduleProperties>
      </outputConfig>
    </crawlConfig>
</configurations>
```

The sub-elements of `outputConfig` define the Record Store instance configuration options. Much of the configuration is specified in <`moduleProperty`> key/value pairs. In your configuration file, configure the following elements.

Configuration element	Description
`moduleId`	Specifies an `id` of a `Record Store`.
`isPortSsl`	Specifies whether to use SSL when connecting to the Record Store instance. A value of `true` uses HTTPS and treats the `port` property as an SSL port. A value of `false` uses HTTP and treats `port` as a non-SSL port. Specify `false` if you enabled redirects from a non-SSL port to an SSL port.
`host`	Specifies the fully qualified name of the host running the Record Store instance. The default value is `localhost`

Configuration element	Description
port	Specifies the port of the Endeca CAS Service running the Record Store instance. The default value is 8500.
instanceName	Specifies the Record Store instance name to write to.
isManaged	Specifies whether the Record Store is managed. If you disable the isManaged property for a data source by setting it to false, a Record Store instance is not created when you configure the data source.The default value is true.

Sample configuration for writing output to an MDEX compatible format

This sample shows a crawl configuration that writes output to an MDEX compatible format.

The configuration change is on a per crawl basis. A crawl configuration file used with the command-line utility can have many crawls defined and each crawl can have different output configuration options.

The procedure requires a crawl configuration file with <outputConfig> settings that specify com.ende¬ca.cas.output.Mdex as the <moduleId>. The other sub-elements of <outputConfig> configure additional details of the MDEX target, such as the input directory for the instance configuration files and the output directory for the CAS records that are passed to Dgidx. This XML snippet shows the <moduleId> for an MDEX compatible output.

```
...
     <outputConfig>
        <moduleId>
          <id>com.endeca.cas.output.Mdex</id>
        </moduleId>
...
```

The following example configuration file shows the configuration for one crawl named ebizsampleapp-last-mile-crawl that writes Dgidx input files to C:/Endeca/apps/ebizsampleapp/data/dgidx_input. You can use this example as the basis for your own configuration file in the procedure below.

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

<configurations xmlns="http://endeca.com/itl/cas/2011-12">
  <crawlConfig>
    <crawlId>
      <id>ebizsampleapp-last-mile-crawl</id>
    </crawlId>
    <unavailableIncrementalSwitchesToFullCrawl>false</unavailableIncremental¬
SwitchesToFullCrawl>
    <crawlThreads>5</crawlThreads>
    <sourceConfig>
      <moduleId>
        <id>com.endeca.cas.source.RecordStoreMerger</id>
      </moduleId>
      <moduleProperties>
        <moduleProperty>
          <key>dataRecordStores</key>
          <value>ebizsampleapp-products</value>
        </moduleProperty>
```

```
        <moduleProperty>
          <key>taxonomyRecordStores</key>
          <value>ebizsampleapp-category-dimension</value>
          <value>ebizsampleapp-trigger-dimensions</value>
        </moduleProperty>
        <moduleProperty>
          <key>isPortSsl</key>
          <value>false</value>
        </moduleProperty>
      </moduleProperties>
    </sourceConfig>
    <manipulatorConfigs/>
    <outputConfig>
      <moduleId>
        <id>com.endeca.cas.output.Mdex</id>
      </moduleId>
      <moduleProperties>
        <moduleProperty>
          <key>inputDirectory</key>
          <value>C:/Endeca/apps/Discover/config/mdex</value>
        </moduleProperty>
        <moduleProperty>
          <key>outputDirectory</key>
          <value>C:/Endeca/apps/Discover/data/dgidx_input</value>
        </moduleProperty>
        <moduleProperty>
          <key>dimensionValueIdManagerInstanceName</key>
          <value>Discover-dimension-value-id-manager</value>
        </moduleProperty>
        <moduleProperty>
          <key>configRepositoryHost</key>
          <value>TSMITH-WIN7</value>
        </moduleProperty>
        <moduleProperty>
          <key>configRepositoryPort</key>
          <value>8006</value>
        </moduleProperty>
        <moduleProperty>
          <key>configRepositoryUserName</key>
          <value>admin</value>
        </moduleProperty>
        <moduleProperty>
          <key>configRepositoryPassword</key>
          <value>admin</value>
        </moduleProperty>
        <moduleProperty>
          <key>configRepositorySite</key>
          <value>Discover</value>
        </moduleProperty>
      </moduleProperties>
    </outputConfig>
  </crawlConfig>
</configurations>
```

The sub-elements of `outputConfig` define the MDEX configuration options. The configuration is specified in `<moduleProperty>` key/value pairs. In your configuration file, configure the following elements.

Configuration element	Description
`moduleId`	Specifies that `id` is `com.endeca.cas.output.Mdex`.

Configuration element	Description
inputDirectory	Specifies a path to the directory containing Developer Studio instance configuration files.
outputDirectory	Specifies a path to the directory where CAS writes output in an MDEX compatible format (i.e. as Dgidx input files). The CAS output is consumed by Dgidx.
dimensionValueIdManagerInstanceName	Specifies the name of the Dimension Value Id Manager for the application.
configRepositoryHost	Specifies the name of the host that is running the Endeca Configuration Repository. (This is the host where Endeca Workbench was installed. The Endeca Configuration Repository is a component of the Workbench installation.)
configRepositoryPort	Specifies the port for the Endeca Configuration Repository. (This is the port for Endeca Workbench.)
configRepositoryUserName	Specifies the username of the application.
configRepositoryPassword	Specifies the corresponding password for the user.
configRepositorySite	Specifies the application name that this crawl is associated with.

Sample configuration for writing output to a file

This sample shows a crawl configuration that writes output to an output file.

The configuration change is on a per crawl basis. A crawl configuration file used with the command-line utility can have many crawls defined and each crawl can have different output configuration options.

The procedure requires a crawl configuration file with <outputConfig> settings that specify File System as the <moduleId>. That File System value constitues the main difference between configuring a crawl to write to an output file versus writing to a Record Store. The other sub-elements of <outputConfig> are additional configuration about the output file itself such as whether compression is enabled, a file prefix name, the path to the output file, and so on. This XML snippet shows the <moduleId> that you change to reconfigure the data source.

```
...
     <outputConfig>
        <moduleId>
          <id>File System</id>
        </moduleId>
...
```

The following example configuration file shows the required configuration for one crawl that writes records to an output file. It is a simple example that does not contain any filters. You can use this example as the basis for your own configuration file in the procedure below.

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

<configurations xmlns="http://endeca.com/itl/cas/2011-12">
    <crawlConfig>
      <crawlId>
        <id>FileCrawl</id>
      </crawlId>
```

```
      <unavailableIncrementalSwitchesToFullCrawl>true</unavailableIncremental¬
SwitchesToFullCrawl>
      <crawlThreads>3</crawlThreads>
      <sourceConfig>
        <moduleId>
          <id>File System</id>
        </moduleId>
        <moduleProperties>
          <moduleProperty>
            <key>expandArchives</key>
            <value>false</value>
          </moduleProperty>
          <moduleProperty>
            <key>gatherNativeFileProperties</key>
            <value>true</value>
          </moduleProperty>
          <moduleProperty>
            <key>seeds</key>
            <value>C:\Endeca\SourceDataDocPOC</value>
          </moduleProperty>
        </moduleProperties>
        <excludeFilters/>
        <includeFilters/>
      </sourceConfig>
      <textExtractionConfig>
        <enabled>true</enabled>
        <makeLocalCopy>false</makeLocalCopy>
        <timeout>90</timeout>
      </textExtractionConfig>
      <manipulatorConfigs/>
      <outputConfig>
        <moduleId>
          <id>File System</id>
        </moduleId>
        <moduleProperties>
          <moduleProperty>
            <key>outputXml</key>
            <value>true</value>
          </moduleProperty>
          <moduleProperty>
            <key>outputCompressed</key>
            <value>false</value>
          </moduleProperty>
          <moduleProperty>
            <key>outputPrefix</key>
            <value>CrawlerOutput</value>
          </moduleProperty>
          <moduleProperty>
            <key>outputDirectory</key>
            <value>C:\tmp</value>
          </moduleProperty>
        </moduleProperties>
      </outputConfig>
    </crawlConfig>
</configurations>
```

The sub-elements of `outputConfig` define the output file configuration options. Much of the configuration is specified in `<moduleProperty>` key/value pairs. In your file, configure the following elements.

Configuration element	Description
`moduleId`	Specifies an `id` of `File System`.
`<key>outputXml</key>`	Specifies whether to write the records as XML or binary. A value of `true` writes a single XML output file of records. A value of `false` writes binary files of records.
`<key>outputCompressed</key>`	Specifies whether to compress the output file or not. Specifying `true` compresses the output. The default is `false` (not compressed).
`<key>outputPrefix</key>`	Specifies an output prefix to the file name. The default prefix is `CrawlerOutput`.
`<key>outputDirectory</key>`	Specifies an output directory for the output file using `<key>outputDirectory</key>`. The default value of `outputDirectory` is `output`. The default name of *`crawlID`* is used to create a subdirectory for each data source. This ensures each crawl has a unique subdirectory for its output. For example, if you use the default value for `outputDirectory` and have a *`crawlID`* of `FileSystemCrawl`, the resulting directory structure is `\`*`CASServerWorkspace`*`\output\FileSystemCrawl\`.

Appendix B

File Formats Supported by the CAS Document Conversion Module

This section lists the binary file formats that the CAS Document Conversion Module can convert to text during a crawl. The CAS Document Conversion Module is installed by default as part of the CAS installation.

Archive formats

The following table lists supported archive formats:

Format	Version (if applicable)
7z (BZIP2 and split archives not supported)	
7z Self Extracting exe (BZIP2 and split archives not supported)	
LZA Self Extracting Compress	
LZH Compress	
Microsoft Binder	95, 97
RAR	1.5, 2.0, 2.9
Self-extracting .exe	
UNIX Compress	
UNIX GZip	
UNIX TAR	
Uuencode	
ZIP	PKZip
ZIP	WinZip

Database formats

The following table lists supported database formats:

Format	Version
DataEase	4.x
DBase	III, IV, and V
First Choice DB	Through 3.0
Framework DB	3.0
Microsoft Access	1.0, 2.0
Microsoft Access Report Snapshot (File ID only)	2000 - 2003
Microsoft Works DB for DOS	1.0, 2.0
Microsoft Works DB for Macintosh	2.0
Microsoft Works DB for Windows	3.0, 4.0
Paradox (DOS)	2.0 - 4.0
Paradox (Windows)	1.0
Q & A	Through 2.0
R:Base	R:Base 5000 and R:Base System V
Reflex	2.0
SmartWare II	1.02

E-mail formats

The following table lists supported e-mail formats:

Format	Version
Apple Mail Message (EMLX)	2.0
Encoded mail messages	MHT
Encoded mail messages	Multi Part Alternative
Encoded mail messages	Multi Part Digest
Encoded mail messages	Multi Part Mixed
Encoded mail messages	Multi Part News Group
Encoded mail messages	Multi Part Signed
Encoded mail messages	TNEF
IBM Lotus Notes Domino XML Language DXL	8.5
IBM Lotus Notes NSF (File ID only)	7.x, 8.x
IBM Lotus Notes NSF (Windows, Linux x86-32 and Oracle Solaris 32-bit only with Notes Client or Domino Server)	8.x

Format	Version
MBOX Mailbox	RFC 822
Microsoft Outlook MSG	97 - 2007
Microsoft Outlook Express (EML)	
Microsoft Outlook Forms Template (OFT)	97 - 2007
Microsoft Outlook OST	97 - 2007
Microsoft Outlook PST	97 - 2007
Microsoft Outlook PST (Mac)	2001

Multimedia formats

The following table lists supported e-mail formats:

Format	Version
AVI (Metadata extraction only)	
Flash (text extraction only)	6.x, 7.x, Lite
Flash (File ID only)	9, 10
Real Media - (File ID only)	
MP3 (ID3 metadata only)	
MPEG-1 Audio layer 3 V ID3 v1 (File ID only)	
MPEG-1 Audio layer 3 V ID3 v2 (File ID only)	
MPEG-1 Video V 2 (File ID only)	
MPEG-1 Video V 3 (File ID only)	
MPEG-2 Audio (File ID only)	
MPEG-4 (Metadata extraction only)	
MPEG-7 (Metadata extraction only)	
QuickTime (Metadata extraction only)	
Windows Media ASF (Metadata extraction only)	
Windows Media DVR-MS (Metadata extraction only)	
Windows Media Audio WMA (Metadata extraction only)	

Format	Version
Windows Media Playlist (File ID only)	
Windows Media Video WMV (Metadata extraction only)	
WAV (Metadata extraction only)	

Other formats

The following table lists other supported formats:

Format	Version (if applicable)
AOL Messenger (File ID only)	7.3
Microsoft InfoPath (file ID only)	2007
Microsoft Live Messenger (via XML filter)	10.0
Microsoft OneNote (file ID only)	2007
Microsoft Project (table view only)	98 - 2003
Microsoft Project (table view only)	2007 - 2010
Microsoft Windows Compiled Help (File ID only)	.chm
Microsoft Windows DLL	
Microsoft Windows Executable	
Microsoft Windows Explorer Command (File ID only)	.scf
Microsoft Windows Help (File ID only)	.hlp
Microsoft Windows Shortcut (File ID only)	.lnk
Trillian Text Log File (via text filter)	4.2
Trillian XML Log File (File ID only)	4.2
TrueType Font (File ID only)	ttf, ttc
vCalendar	2.1
vCard	2.1
Yahoo! Messenger	6.x - 8.0

Presentation formats

The following table lists supported presentation formats:

Format	Version (if applicable)
Corel Presentations	6.0 - X3
Harvard Graphics (DOS)	3.0
IBM Lotus Symphony Presentations	1.x
Kingsoft WPS Presentation	2010
Lotus Freelance	1.0 - Millennium 9.6
Lotus Freelance (OS/3)	2.0
Lotus Freelance for Windows	95, 97
Microsoft PowerPoint for Macintosh	4.0 - 2008
Microsoft PowerPoint for Windows	3.0 - 2010
Microsoft PowerPoint for Windows Slideshow	2007 - 2010
Microsoft PowerPoint for Windows Template	2007 - 2010
Novell Presentations	3.0, 7.0
OpenOffice Impress	1.1, 3.0
Oracle Open Office Impress	3.x
StarOffice Impress	5.2 - 9.0
WordPerfect Presentations	5.1 - X4

Raster image formats

The following table lists supported raster image formats:

Format	Version
CALS Raster (GP4)	Type I and Type II
Computer Graphics Metafile	ANSI, CALS, NIST
Encapsulated PostScript (EPS)	TIFF header only
GEM Image (Bitmap)	
Graphics Interchange Format (GIF)	
IBM Graphics Data Format (GDF)	1.0
IBM Picture Interchange Format (PIF)	1.0
JBIG2	graphic embeddings in PDF files
JFIF (JPEG not in TIFF format)	
JPEG	

Format	Version
JPEG 2000	JP2
Kodak Flash Pix	
Kodak Photo CD	1.0
Lotus PIC	
Lotus Snapshot	
Macintosh PICT1 and PICT2	BMP only
MacPaint	
Microsoft Windows Bitmap	
Microsoft Windows Cursor	
Microsoft Windows Icon	
OS/2 Bitmap	
OS/2 Warp Bitmap	
Paint Shop Pro (Win32 only)	5.0, 6.0
PC Paintbrush (PCX)	
PC Paintbrush DCX (multi-page PCX)	
Portable Bitmap (PBM)	
Portable Graymap (PGM)	
Portable Network Graphics (PNG)	
Portable Pixmap (PPM)	
Progressive JPEG	
StarOffice Draw	6.x - 9.0
Sun Raster	
TIFF	Group 5 and Group 6
TIFF CCITT Fax	Group 3 and Group 4
Truevision TGA (Targa)	2.0
WBMP wireless graphics format	
Word Perfect Graphics	1.0
X-Windows Bitmap	x10 compatible
X-Windows Dump	x10 compatible
X-Windows Pixmap	x10 compatible
WordPerfect Graphics	2.0, 7.0, 8.0, 9.0, 10.0

Spreadsheet formats

The following table lists supported spreadsheet formats:

Format	Version
Enable Spreadsheet	3.0 - 4.5
First Choice SS	Through 3.0
Framework SS	3.0
IBM Lotus Symphony Spreadsheets	1.x
Kingsoft WPS Spreadsheets	2010
Lotus 1-2-3	Through Millennium 9.6
Lotus 1-2-3 Charts (DOS and Windows)	Through 5.0
Lotus 1-2-3 (OS/2)	2.0
Microsoft Excel Charts	2.x - 2007
Microsoft Excel for Macintosh	98 - 2008
Microsoft Excel for Windows	3.0 - 2010
Microsoft Excel for Windows (xslb)	2007 - 2010 Binary
Microsoft Multiplan	4.0
Microsoft SS Works for DOS	2.0
Microsoft Works for Macintosh	2.0
Microsoft SS Works for Windows	3.0, 4.0
Novell PerfectWorks	2.0
OpenOffice Calc	1.1 - 3.0
Oracle Open Office Calc	3.x
PFS: Professional Plan	1.0
Quattro for DOS	Through 5.0
QuattroPro for Windows	Through X4
SmartWare Spreadsheet	
SmartWare II SS	1.02
StarOffice Calc	5.2 - 9.0
SuperCalc	5.0
Symphony	Through 2.0
VP Planner	1.0

Text and markup formats

The following table lists supported text and markup formats:

Notes:

- CAS 2.3.0 and later supports converting XML content contained in both PCDATA and CDATA elements.
- In the case of XHTML, "file ID only" means that the conversion process produces an Endeca property for the file format type but nothing else.

Format	Version (if applicable)
ANSI Text	7 bit and 8 bit
ASCII Text	7 bit and 8 bit
DOS character set	
EBCDIC	
HTML (CSS rendering not supported)	1.0 - 4.0
IBM DCA/RFT	
Macintosh character set	
Rich Text Format (RTF)	
Unicode Text	3.0, 4.0
UTF-8	
Wireless Markup Language	1.0
XML	text only
XHTML (file ID only)	1.0

Vector image formats

The following table lists supported vector image formats:

Format	Version (if applicable)
Adobe Illustrator	4.0 - 7.0, 9.0
Adobe Illustrator (XMP only)	11 - 13 (CS 1 - 3)
Adobe InDesign (XMP only)	3.0 - 5.0 (CS 1 - 3)
Adobe InDesign Interchange (XMP only)	
Adobe Photoshop (XMP only)	8.0 -10.0 (CS 1 - 3)
Adobe PDF	1.0 - 1.7 (Acrobat 1 - 9)
Adobe PDF Package	1.7 (Acrobat 8 - 9)
Adobe PDF Portfolio	1.7 (Acrobat 8 - 9)
Adobe Photoshop	4.0

Format	Version (if applicable)
Ami Draw	SDW
AutoCAD Drawing	2.5, 2.6
AutoCAD Drawing	9.0 - 14.0
AutoCAD Drawing	2000i - 2010
AutoShade Rendering	2.0
Corel Draw	2.0 - 9.0
Corel Draw Clipart	5.0, 7.0
Enhanced Metafile (EMF)	
Escher graphics	
FrameMaker Vector and Raster Graphics (FMV)	3.0 - 5.0
Gem File (Vector)	
Harvard Graphics Chart (DOS)	2.0 - 3.0
Harvard Graphics for Windows	
HP Graphics Language	2.0
Initial Graphics Exchange Specification (IGES) Drawing	5.1 - 5.3
Micrografx Designer	Through 3.1
Micrografx Designer	6.0
Micrografx Draw	Through 4.0
Microsoft XPS (Text only)	
Novell PerfectWorks Draw	2.0
OpenOffice Draw	1.1 - 3.0
Oracle Open Office Draw	3.x
Visio (Page Preview mode only WMF/EMF)	4
Visio	5.0 - 2007
Visio XML VSX (File ID only)	2007
Windows Metafile	

Notes on Adobe PDF text extraction

The CAS Document Conversion Module works as follows when processing Adobe PDF files with security settings:

- The CAS Document Conversion Module will respect the no-copy option of a PDF. That is, if a PDF publishing application has a no-copy option (which prohibits the copying or extraction of text within the PDF), the Document Conversion Module will not extract text from that PDF.
- The CAS Document Conversion Module does not support text extraction from password-protected files.

- The CAS Document Conversion Module does not support text extraction from PDFs with encrypted content.

To extract the text from these types of PDFs, you must re-create them without setting the appropriate security option.

In addition, text added with the Sticky Note tool is not extracted.

Word processing formats

The following table lists supported word processing formats:

Format	Version (if applicable)
Adobe FrameMaker (MIF)	Versions 3.0 - 6.0
Adobe Illustrator Postscript	Level 2
Ami	
Ami Pro for OS2	
Ami Pro for Windows	2.0, 3.0
DEC DX	Through 4.0
DEC DX Plus	4.0, 4.1
Enable Word Processor	3.0 - 4.5
First Choice WP	1.0, 3.0
Framework WP	3.0
Hangul	97 - 2007
IBM DCA/FFT	
IBM DisplayWrite	2.0 - 5.0
IBM Writing Assistant	1.01
Ichitaro	5.0, 6.0, 8.0 - 13.0, 2004
JustWrite	Through 3.0
Kingsoft WPS Writer	2010
Legacy	1.1
Lotus Manuscript	Through 2.0
Lotus WordPro	9.7, 96, - Millennium 9.6
Lotus WordPro (non-Win32)	97 - Millennium 9.6
MacWrite II	1.1
Mass 11	All versions through 8.0
Microsoft Publisher (File ID only)	2003 - 2007
Microsoft Word for DOS	4.0 - 6.0
Microsoft Word for Macintosh	4.0 - 6.0, 98 - 2008

Format	Version (if applicable)
Microsoft Word for Windows	1.0 - 2007
Microsoft Word for Windows	98-J
Microsoft WordPad	
Microsoft Works WP for DOS	2.0
Microsoft Works WP for Macintosh	2.0
Microsoft Works WP for Windows	3.0, 4.0
Microsoft Write for Windows	1.0 - 3.0
MultiMate	Through 4.0
MultiMate Advantage	2.0
Navy DIF	
Nota Bene	3.0
Novell Perfect Works	2.0
Office Writer	4.0 - 6.0
OpenOffice Writer	1.1 - 3.0
Oracle Open Office Writer	3.x
PC File Doc	5.0
PFS:Write	Versions A, B
Professional Write (DOS)	1.0, 2.0
Professional Write Plus (Windows)	1.0
Q&A Write (Windows)	2.0, 3.0
Samna Word IV	1.0 - 3.0
Smna Work IV+	
Samsung JungUm Global (File ID only)	
Signature	1.0
SmartWare II WP	1.02
Sprint	1.0
StarOffice Writer	5.2 - 9.0
Total Word	1.2
Wang PC (IWP)	Versions through 2.6
WordMarc Composer	
WordMarc Composer+	
WordMarc Word Processor	
WordPerfect for DOS	4.2
WordPerfect for Macintosh	1.02 - 3.1

Format	Version (if applicable)
WordPerfect for Windows	5.1 - X4
WordStar 2000 for DOS	1.0 - 3.0
WordStar 2000 for DOS	2.0, 3.0
WordStar for DOS	3.0 - 7.0
WordStar for Windows	1.0
XyWrite	Through III+

Appendix C

Record properties generated by crawling

During a crawl, the CAS Server produces record properties according to certain naming schemes. You can map any of these properties to Endeca properties or dimensions by the property mapper component in a pipeline.

Common record properties

The CAS Server generates certain properties whether you crawl a file system, CMS, or custom data source extension.

The CAS Server generates record properties and assigns each property a qualified name, with a period (.) to separate qualifier terms. The CAS Server constructs the qualified name as follows:

- The first term is always `Endeca` and is followed by one or more additional terms.
- The second term describes a property category, for example: `CMS` or `FileSystem`. The term `File` may be added to files from either file system or content management system data sources.
- The third and fourth terms, if present, fully qualify the property, for example: `Endeca.CMS.ItemId` or `Endeca.FileSystem.Path`.

The CAS Server may generate the following properties for all records:

Endeca Property Name	Property Value
`Endeca.Action`	The action that was taken with the document. Values are `UPSERT` (the file or folder has been added or modified) or `DELETE` (the document or directory has been deleted since the last crawl).
`Endeca.SourceType`	Indicates the source type of the crawl. Values are `FILESYSTEM` (for file system data sources), `WEB` (for Web servers), `CMS` (for Content Management System data sources), or `EXTENSION` (for data source extensions).
`Endeca.Id`	Provides a unique identifier for each record. For file system crawls, `Endeca.Id` is the same as `Ende¬ca.FileSystem.Path`. It is the full path to the file including the file name. For archive files, this is a string pointing to a file within `Endeca.FileSystem.Path` a container. This property also includes the `PathWithinSourceArchive` (if present). For Web crawls, `Endeca.Id` is the same as `Endeca.Web.Url`.

Endeca Property Name	Property Value
	For CMS crawls, `Endeca.Id` is the concatenation of the `Endeca.CMS.RepositoryId` and `Endeca.CMS.ItemId` properties, and the `Endeca.CMS.ContentPieceId` (if present). This property also includes the `PathWithinSourceArchive` (if present) For data source extensions, a plug-in developer must add `Endeca.Id` to each record and assign it a value appropriate for the data source.
`Endeca.SourceId`	Indicates the name of the data source. This is the same as the `id` value of `crawlId` in a crawl configuration.
`Endeca.File.IsArchive`	A boolean that, if set to a value of `true`, indicates that the document is an archive file, such as a Zip file. If the file is not identified as an archive, the property is absent. Note that archives are identified by their file extension or Mime type. It is possible for a document to have both `Endeca.File.IsArchive` and `Endeca.File.IsInArchive` properties set, as archive files may contain other archive files nested within.
`Endeca.File.IsInArchive`	A boolean that, if set to a value of `true`, indicates that the document is extracted from an archive file. If the file is not an archived document, the property is absent.
`Endeca.File.Size`	The size of the document in bytes, as reported by the native file system, CMS, or an archive entry.
`Endeca.File.SourceArchiveId`	This property is added to all records that have the `Endeca.File.IsInArchive` property. It is intended to provide a reference to the original archive that was encountered in the file system or CMS. The value is the original archive's `Endeca.FileSystem.Path` or `Endeca.Id` property. In the case of nested archives, it is the top-level archive, because that is the original source in the file system or CMS being crawled.
`Endeca.File.PathWithinSourceArchive`	This property is added to all records that have the `Endeca.File.SourceArchiveId` property. The value of this property is the path to the current record within the source archive file. In the case of nested archive entries, it includes the path to the nested archive, appended with the path to the current record within the nested archive.

Record properties generated by file system crawls

During a file system crawl, the CAS Server produces record properties according to a standardized naming scheme.

Windows file example

The following example shows the properties returned from a Windows crawl for a Windows text file named `TestFile.txt`, which is owned by user fsmith from the DEVGROUP domain:

```
...

<RECORD>

...

  <PROP NAME="Endeca.FileSystem.Owner">
      <PVAL>DEVGROUP\fsmith</PVAL>
  </PROP>
  <PROP NAME="Endeca.FileSystem.Group">
      <PVAL>DEVGROUP\Domain Users</PVAL>
  </PROP>
  <PROP NAME="Endeca.FileSystem.IsHidden">
      <PVAL>false</PVAL>
  </PROP>
  <PROP NAME="Endeca.FileSystem.IsTemporary">
      <PVAL>false</PVAL>
  </PROP>
  <PROP NAME="Endeca.FileSystem.Path">
      <PVAL>c:\endecafiles\TestFile.txt</PVAL>
  </PROP>
  <PROP NAME="Endeca.FileSystem.ParentPath">
      <PVAL>c:\endecafiles</PVAL>
  </PROP>
  <PROP NAME="Endeca.FileSystem.ACL.AllowRead">
      <PVAL>BUILTIN\Administrators</PVAL>
  </PROP>
  <PROP NAME="Endeca.FileSystem.ACL.AllowRead">
      <PVAL>NT AUTHORITY\SYSTEM</PVAL>
  </PROP>
  <PROP NAME="Endeca.FileSystem.ACL.AllowRead">
      <PVAL>DEVGROUP\fsmith</PVAL>
  </PROP>
  <PROP NAME="Endeca.FileSystem.ACL.AllowRead">
      <PVAL>BUILTIN\Users</PVAL>
  </PROP>
  <PROP NAME="Endeca.FileSystem.IsDirectory">
      <PVAL>false</PVAL>
  </PROP>
  <PROP NAME="Endeca.FileSystem.ModificationDate">
      <PVAL>1182453853873</PVAL>
  </PROP>
  <PROP NAME="Endeca.FileSystem.CreationDate">
      <PVAL>1182453827530</PVAL>
  </PROP>
  <PROP NAME="Endeca.Action">
      <PVAL>UPSERT</PVAL>
  </PROP>
  <PROP NAME="Endeca.FileSystem.IsSystem">
      <PVAL>false</PVAL>
  </PROP>
  <PROP NAME="Endeca.File.Size">
      <PVAL>16</PVAL>
  </PROP>
  <PROP NAME="Endeca.Document.Type">
```

```
      <PVAL>Unknown (ASCII 8)</PVAL>
  </PROP>
  <PROP NAME="Endeca.Document.Text">
      <PVAL>This is a test.</PVAL>
  </PROP>
</RECORD>

...
```

Common File System properties

The CAS Server produces some common properties from records crawled in either a Windows or UNIX file system.

The following record properties are common to documents fetched from both Windows and UNIX file systems.

Endeca Property Name	Property Value
`Endeca.FileSystem.Extension`	The file extension of the document, which is the string after the last dot in the file name. If the document has no dot in the name, this property will not be generated.
`Endeca.FileSystem.Group`	The name of a group for which permissions have been set for the document. For UNIX files, the property value is the `groupname`. For Windows files, the name is prepended with the domain to which the group name belongs, in the format: `DOMAIN\principal`.
`Endeca.FileSystem.IsDirectory`	A Boolean that indicates whether the document is a directory (a value of `true`) or a file (a value of `false`). The value is set to `true` even if the directory is in a container.
`Endeca.FileSystem.IsHidden`	A Boolean that indicates whether the document is a hidden file (a value of `true`) or not (a value of `false`).
`Endeca.FileSystem.Modification¬Date`	The date when the file was last modified. Modifications include changing permissions on the document. The date format is in milliseconds since midnight January 1, 1970 UTC (Coordinated Universal Time).
`Endeca.FileSystem.Name`	The name of the file.
`Endeca.FileSystem.Owner`	The name of a user or other principal who is the owner of the file. For UNIX files, the property value is the `ownername`. For Windows files, the name is prepended with the domain to which the name belongs, in the format: `DOMAIN\principal`.
`Endeca.FileSystem.Path`	The identifier of the full path to the file, including the file name. For archive files, this is a string pointing to a file within a container. This property also includes the `PathWithin¬SourceArchive` (if present).
`Endeca.FileSystem.ParentPath`	The identifier of the path to the directory containing the file. This does not include the file name. For archive files, this is a string pointing to a container.

Record properties for file system crawls on Windows

The CAS Server produces certain properties from records crawled on a Windows file system.

The following table lists the record file properties that are specific to Windows file systems.

Endeca Property Name	Property Value
`Endeca.FileSystem.ACL.AllowRead`	The name of a user, group, or other principal who has the right to read the document. The name is prepended with the domain to which the name belongs, in the format: `DOMAIN\principal` .
`Endeca.FileSystem.ACL.DenyRead`	The name of a user, group, or other principal who is denied the right to read the document. The name is prepended with the domain to which the name belongs, in the format: `DOMAIN\principal` .
`Endeca.FileSystem.CreationDate`	The date when the document was created. The date format is in milliseconds since midnight January 1, 1970 UTC (Coordinated Universal Time).
`Endeca.FileSystem.IsSystem`	A Boolean that indicates whether the document is a system file (a value of `true`) or not (a value of `false`).
`Endeca.FileSystem.IsTemporary`	A Boolean that indicates whether the document is a temporary file (a value of `true`) or not (a value of `false`).

Record properties for file system crawls on UNIX

The CAS Server produces certain properties from records crawled in a UNIX file system.

The following table lists the record file properties that are specific to UNIX file systems.

Endeca Property Name	Property Value
`Endeca.FileSystem.IsGroupReadable`	A Boolean that indicates whether the group (the `Endeca.FileSystem.Group` value) has read rights to the document.
`Endeca.FileSystem.IsOwnerReadable`	A Boolean that indicates whether the file owner (the `Endeca.FileSystem.Owner` value) has read rights to the document.
`Endeca.FileSystem.IsSymbolicLink`	A Boolean that indicates whether the document is a symbolic link that refers to another file or directory (a value of `true` indicates that the document is a symbolic link).
`Endeca.FileSystem.IsWorldReadable`	A Boolean that indicates whether everyone on the system (`world`) has read rights to the document.
`Endeca.FileSystem.LinkTarget`	The name of the document to which a symbolic link refers. This property is present only if the `Endeca.FileSystem.IsSymbolicLink` property is set to `true`.

Limitations with ACL properties

The Content Acquisition System on Windows cannot get ACL properties for seeds that represent a root folder. However, the Content Acquisition System successfully gets ACL properties for all children of the root.

This limitation only occurs in the following scenario:

- The machine running the Endeca CAS Service is a Windows machine.
- The crawl is a file crawl with the **Retrieve ACLs** option enabled. (By default **Retrieve ACLs** is enabled.)
- The seed specified represents a root folder (for example, `C:\` or `\\machinename\folder`).

The Content Acquisition System produces an Endeca record for a root folder, and the record is tagged with other generated record properties. Only the ACL properties are missing.

Document Conversion properties

The CAS Document Conversion Module generates certain properties for records crawled with document conversion enabled.

The CAS Document Conversion Module generates `Document` properties that contain information (including the text) of the document or metadata about the document.

Endeca Property Name	Property Value
`Endeca.Document.Metadata.`*`attribute`*	Metadata information in the document. The metadata attributes depend on which ones were added by the authoring tool used to create the document. For example, an Adobe Acrobat PDF document could have such metadata attributes as `Endeca.Document.Metadata.title` and `Endeca.Document.Metadata.primary_author`.
`Endeca.Document.Metadata.Misc`	Properties that are returned from the CAS Document Conversion Module but that do not have a type attribute are mapped to this property.
`Endeca.Document.Text`	The text (content) of the source document. Note that the CAS Document Conversion Module typically does not preserve line break information.
`Endeca.Document.TextExtraction.Error`	An error returned by the CAS Document Conversion Module. Note that a `no filter available for this file type` error indicates that you should modify the document conversion module to exclude files of this type.
`Endeca.Document.Type`	The type of document, such as Microsoft Word 2003/2004, Adobe Acrobat (PDF), JPEG File Interchange, and Extensible Markup Language (XML).

Note:

- You should not use these properties for filters. These properties are created after the files are accessed, and therefore cannot be used to filter out files.
- If you crawl a data source without text conversion enabled (a probe crawl), none of these properties are generated.

Record properties generated by CMS crawls

The CAS Server produces certain CMS properties regardless of whether document conversion is enabled or not. The following record properties are common to CMS data sources.

Endeca Property Name	Property Value
Endeca.CMS.Uri	The URI of the object which, if defined, allows an application to access an object as a web resource.
Endeca.CMS.UpdatedBy	The user name of the person who updated the content item.
Endeca.CMS.RepositoryType	The type of CMS repository.
Endeca.CMS.RepositoryVersion	The version of the CMS repository, such as 7.1.
Endeca.CMS.RepositoryId	The ID of the repository.
Endeca.CMS.ItemId	The unique ID of the item in the repository.
Endeca.CMS.ContentPieceId	The unique ID of the item's content piece.
Endeca.CMS.Path	The path to the item in the repository, including the name of item, such as /dctm65/test. In cases where an item resides in several locations, the property value for both Endeca.CMS.Path and Endeca.CMS.ParentPath depends on which location the CAS Server encounters first when crawling the repository. This means that subsequent crawls of the repository may produce different property values. For example, if an item named doc1 resides in root/folder1/doc1 and root/doc1, the CAS Server can produce a property value of either root/folder1/doc1 or root/doc1.
Endeca.CMS.ParentPath	The path to the parent of the item in the repository, not including the name of the item, such as /dctm65.
Endeca.CMS.Name	The name of the item.
Endeca.CMS.Author	The author of the item.
Endeca.CMS.IsFolder	true if the item is a folder, false otherwise.
Endeca.CMS.NumContentPieces	The number of pieces of content associated with the item in the repository.
Endeca.CMS.ContentLength	The length in bytes of the content as reported by the CAS Server.
Endeca.CMS.CreationDate	The creation date of the item.
Endeca.CMS.ModificationDate	The last modified date of the item.
Endeca.CMS.MimeType	The MIME type of the item.
Endeca.CMS.AllowReadContent	An ACL entry for a user or group that can read the content of the item.
Endeca.CMS.DenyReadContent	An ACL entry for a user or group that cannot read the content of the item.
Endeca.CMS.AllowReadProper¬ties	An ACL entry for a user or group that can read the properties of the item.
Endeca.CMS.DenyReadProper¬ties	An ACL entry for a user or group that cannot read the properties of the item.

Here are additional notes concerning record properties produced by CMS crawls:

- In addition to the properties listed above, an Endeca record may also contain properties that are specific to a CMS repository that are passed through to CAS. Such properties have a prefix of `Endeca.CMS.Misc`.
- CMS data sources may be inconsistent in displaying the format of ACL property values. For example, property values could contain: user display name, user display name@domain, domain\user name, domain\group name, or domain\role name.

  CAS makes its best effort to return names in the form [domain\](user name), [domain\](group name), and [domain\](role name), but CAS is limited by the capabilities of the underlying CMS and the values the CMS returns in ACLs.

How CMS crawls handle multiple pieces of content

Some CMS repositories support items with multiple pieces of content. In these cases the CAS Server outputs a record for the item and records for each piece of content.

For example, an item from a repository could contain an attached PDF and an Excel file.

After the crawl, the records for each piece of content will contain:

- All properties of the original item record, such as ACL user and group permission entries of type `Ende¬ca.CMS.AllowReadContent`
- A content piece identifier property `Endeca.CMS.ContentPieceId`
- An identifier of a specific record `Endeca.Id`. It is the concatenation of the `Endeca.CMS.RepositoryId` and `Endeca.CMS.ItemId` properties, and also the `Endeca.CMS.ContentPieceId` (if present).

Index

A

archive files, support for 43
archived output files 46
automatically generating dimension values 89

B

baseline updates, running 51, 53

C

CAS Document Conversion Module
    options for 24
CAS Server Command-line Utility
    creating a Dimension Value Id Manager 112
    creating crawls 103
    deleting a crawl 104
    deleting a Dimension Value Id Manager 113
    getting a crawl 107
    getting all crawls 105
    getting metrics for all crawls 118
    getting specification of a module 101
    getting the metrics of a crawl 120
    getting the status of a data source acquisition 121
    listing all module specifications 100, 108
    listing crawls 109
    listing data sources and manipulators 102
    saving passwords for crawls 99
    starting acquisition from a data source 110
    stopping acquisition from a data source 111
    updating crawls 111
    exporting dimension value Ids 114
    generating dimension value Ids 115
    getting dimension value Ids 116
    getting dimension value specifications 116
    importing dimension value Ids 117
    sample crawl configuration file of a data source 170
    sample crawl configuration file of a manipulator 173
changing logging levels 159
CIM Command-line Utility
    creating components 124
    deleting components 125
    listing components 126
    overview of 123
cleaner
    interval property 33
client state
    overview 30
configuring a Record Store instance 33
crawls
    output filename 46

D

deleted files, properties of 31
Dimension Value Id Manager 17
Document Conversion module
    other supported formats 184
    supported compressed formats 181
    supported database formats 182
    supported e-mail formats 182
    supported multimedia formats 183
    supported presentation formats 185
    supported raster image formats 185
    supported text and markup formats 188
    supported vector image formats 188
    supported word processing formats 190

E

Endeca CAS Server
    changing host and port 165
    CAS Document Conversion Module options 24
    creating a Forge pipeline 80
    flags for startup scripts 151
    output files 46
    overview 15
    properties of deleted files 31
    recommended file filters 22
    record properties 193
    specifying JVM arguments 151
    starting 150
    stopping 152
Endeca CAS Service log 161
Endeca Crawler
    Document Conversion module
        supported spreadsheet formats 187
Endeca Document Conversion Module
    properties generated by 198
Endeca Record Store instance configuration 33
Endeca Web Crawler, running sample 54
errors
    too many open files 166

F

file system properties 196
filters, overview of 22
flags for startup script 151
Forge sample applications
    reading from Record Store 51, 53
    writing to Record Store 50

G

generated record properties
    for multiple pieces of content 200
group entry size, setting 167
GZIP Tar files, support for 43

J

Jar files, support for 44
JVM arguments for crawls, specifying 151

L

log, CAS Service 161
logging configuration files 159

O

output records file
    archived 46
    naming format 46

P

partial updates, running 51, 53
pipeline
    creating a record adapter 82
    overview 80
pipeline, creating a Forge 79

R

record adapter for pipeline, creating 82
record properties
    CMS crawls 199
    for deleted files 31
Record Store Command-line Utility
    committing transactions 136
    getting client state 139
    getting configuration 137
    getting last-committed generation ID 138
    getting last-read generation 139
    getting write generation ID 140
    listing active transactions 140
Record Store Command-line Utility *(continued)*
    listing generations 141
    overview of 129
    reading baselines 132
    reading delta records 133
    reading records by ID 134
    rolling back transactions 142
    running the cleaner 135
    setting client state 136, 144
    setting configuration 143
    setting last-read generation 136, 144
    starting transactions 145
    writing records 131

S

sample applications
    Forge reading from Record Store 51, 53
    Forge writing to Record Store 50
    Web Crawler 54
security 19
sourceConfig properties 169
starting the CAS Server 150
stopping the CAS Server 152

T

Tar files, support for 44
too many open files error 166
transactions
    overview 29

U

UNIX record properties 197

W

Windows record properties 197
workspace directory output files 46

Z

Zip files, support for 43