Caliente 3.9

Common Commands Cheat Sheet & Quick Installation Guide

Effective Date: November 10th, 2017

Approved By: Management Team

Version Number: 0.1.0

Contact Name: Diego Rivera

Responsible Group: Armedia Technology CoP

Asset Type: Reference Document

Table of Contents

Document History

Name	Date	Reason For Changes	Version	Approval
Draft	2017-11-10	Initial Draft	0.1	
Draft	2017-12-18	Added quick installation instructions	0.2	
Draft	2020-02-19	Updated to match 3.9 release	0.3	

1 Introduction

1.1 Document Purpose

The purpose of this document is to provide a set of sample commands to provide guidance for common use-case scenarios with Caliente, and to provide simple installation and deployment steps to get Caliente up and running. Software requirements and Hardware recommendation guidelines are also covered.

1.2 Audience

This document contains information aimed at technical audiences which will be operating Caliente to perform content extractions or migrations, and/or provisioning systems to support a Caliente migration:

- Inform SysAdmins or on-premises operators with regards to common usage patterns for Caliente
- Inform SysAdmins or IT management staff with regards to supported and recommended hardware and software configurations for running Caliente
- Provide a brief overview of Caliente's capabilities and modes of operation

2 Installation and Provisioning

This section covers the basics with regards to Caliente's supported software environments, as well as recommended hardware dimensioning.

2.1 Hardware Recommendations

Caliente's hardware requirements are directly dependent upon both the model of use and the scale of the migration being undertaken. Naturally, smaller migrations require less horsepower, while larger efforts may require significant hardware.

That said, Caliente is fairly efficient with regards to its resource usage, so there's a limit to how big a Caliente server needs to be for migrations to be both performant and successful.

2.1.1 Minimum Hardware

- CPU: 4 Cores, at least at 2.4GHz
- RAM: 16GB DDR RAM
- Networking: 1Gbps full-duplex
- Disk Space: see below

2.1.2 Recommended Hardware

- CPU: 8-12 Cores, 2.4GHz minimium, preferably 3.0GHz or better
- RAM: 24GB to 32GB DDR RAM
- Networking: 10Gbps full-duplex
- Disk Space: see below

Obviously, more hardware can be provisioned as desired, but depending on environmental circumstances (network bandwidth, server capacity), using more powerful hardware configurations will yield limited performance benefits (diminishing returns). This is perhaps the area of configuration where there is the most flexibility with regards to supported configurations since using limited hardware will only result in limiting the size of data sets that Caliente can operate on at any one time, and how long that operation will take to complete.

2.1.3 Disk Space Requirements

Disk space requirements are solely dependent on the size of the migration being attempted. Large migrations will, of course, require more disk space. If feasible, Armedia recommends being able to store the entire extraction set on-disk, local to the Caliente server, plus 50% extra space for overhead (metadata storage, logs, etc).

Furthermore, the use of SSD is heavily favored, though not required.

However, a minimum of 500GB ***dedicated for data storage*** should be assumed for any migration job. Operating system and required software (JVM, extra tools) disk space requirements should not affect this number.

2.1.4 A note on Databases

Caliente utilizes H2 as its internal, default database. H2 has been observed to present acceptable performance up to the millions of objects. For larger counts (tens of millions, for instance), performance may begin to falter. It's an exercise for the reader to benchmark extraction in their own hardware in order to determine whether H2 is sufficient for the intended purpose, or if an alternative database should be used.

Caliente also supports connecting to a PostgreSQL database (local or otherwise), and using PostgreSQL there is no size limitation for extraction jobs: the local disk space for document contents becomes the dominant limitation.

The use of PostgreSQL with Caliente should only be attempted in close coordination with Armedia.

2.2 Software Requirements

2.2.5 Java Virtual Machine Requirements

Caliente is a command-line Java application, compatible with specific versions Oracle Java in 64-bit mode. Caliente is not supported on 32-bit JVM installations.

Supported Java Versions:

- Oracle Java 8 (1.8.*), 64-bit
- OpenJDK 8 (1.8.*), 64-bit

Other hardware platforms may be supported as long as a) a compatible Oracle Java VM is available, and b) Armedia asserts the support and compatibility for the platform in writing (may be outside this document).

Specifically, Caliente has been tested with Oracle Java version 9 (1.9.*) and later, and has been found ***incompatible*** due to specific design goals that have yet to be resolved for the new Jigsaw (JSR376) architecture.

2.2.6 Operating System Requirements

Caliente is only supported on the following operating system configurations:

- Windows 7 (or newer), 64-bit
- Windows Server 2008R2 (or newer), 64-bit
- RedHat Enterprise Linux 6 (or newer), Server or Desktop, 64-bit
- CentOS 6 (or newer), Server or Desktop, 64-bit
- Ubuntu 16.04 (or newer), Server or Desktop, 64-bit

In particular, other Linux versions not listed here may also be compatible, but aren't expressly supported.

2.3 Installation

Caliente does not require any specific installation as it's a self-contained executable JAR file. However, other supporting components may require specific installation steps in order to provide the required compatibility layers. Consult the installation and configuration documents for these components as appropriate for your installation:

- Oracle Java VM (***required***, see above for supported versions)
- Documentum Foundation Classes client (optional, version 6.5 and newer)
    - This is only required when Caliente will be interfacing with Documentum
- PostgreSQL (optional, version 9.6 or newer)
    - This is only required as dictated by Armedia
- Armedia's customized Alfresco Bulk Importer (optional, version 2.5.4 or newer)
    - This is only required when importing into Alfresco

3 Caliente Operation and Self-Help

Caliente is built with an internal help mechanism that describes the parameters available for use by operators. This list can be invoked either explicitly by including the ***--help*** parameter (short version: ***-h***):

```
$ java –jar caliente-exe.jar -h
# This will produce Caliente’s help output

$ java –jar caliente-exe.jar --help
# This will also produce Caliente’s help output
```

You can also add the ***--help*** parameter to ***<u>any</u>*** Caliente command line that you’ve already constructed, however complex, to cause Caliente to output its help information and exit. This is convenient when you’ve spent a significant amount of time constructing the command, but need to quickly reference the Caliente command syntax.

Furthermore, Caliente’s command-line help is context-sensitive, and will display help information related to the command-line that has been constructed so far, specifically towards the command that is to be executed and the engine to use for execution.

3.1 General Advice

Caliente’s syntax is meant to be simple, but can become verbose when attempting more complex tasks. As such, the following suggestions may help you reach success more easily:

- Scripts (or batch files) are your friends! Using scripts may help you avoid having to type the same parameters over and over and over again, while also help you avoid headaches due to typos and such.
- Caliente produces a good amount of logging information – learn to make sense of it!
    - The manifest files (*.manifest.csv) are especially useful since, as CSV files, they can be loaded directly into Microsoft Excel, LibreOffice Calc, or some other program that may assist you in analyzing the data contained therein
- Caliente can only be as fast as its surrounding environments
    - In our experience, the single biggest factor for poor export or import performance is the CMS server’s performance. If the server is constrained in terms of disk speed, RAM, or CPU, then Caliente will be unable to extract or ingest content at a high rate of transfer. Also, care should be taken to not run Caliente exports or imports during production hours as the extra load may adversely affect the performance perceived by other users.
    - The next biggest factor is the available network bandwidth between Caliente and the CMS server – if this bandwidth is constrained, Caliente will bottleneck upon

itself because of all the simultaneous downloads it will attempt. This can be throttled with the ***--threads*** parameter (see below), but will still significantly impact the speed of the operation

- Last, but not least, the hardware Caliente runs on should be powerful enough to support "large" numbers of threads, and a good amount of Java Heap memory. We generally recommend no less than 8 CPU cores, 16GB of RAM, and as much disk space as possible to reduce the number of "trips" it will take to migrate all the data over (SSDs are highly preferred – especially for the metadata database)

3.2 Basic Command Structure

All Caliente commands take the following basic structure:

```
$ java –jar caliente-exe.jar [ global-options ] command [ command-options ]  [ <arg#1>
... <arg#N> ]
```

This is a quick description of each of the portions of that command line structure:

- ***[global-options]***: these are options that have global effect and aren't tied to specific engine or command choices. For example: the engine selection parameter, the help parameter, the supplementary library parameter, and logging control parameters

- ***command***: this is the operation that Caliente is meant to execute in this invocation. An abbreviated version of the command is also acceptable here, where available. For example: decrypt (dec), encrypt (enc), export (ex, exp), import (im, imp)

- ***[command-options]***: these are options specific to the selected command, and may vary between commands, while also taking into account the engine choice

- ***[<arg#1> … <arg#N>]***: positional command-line parameters that may be useful to the given command

The Caliente help system takes into account both the selected command as well as the engine choice in order to parse the entire command line. In particular, these settings are also taken into account when displaying the help message such that you can add the help parameter at any point to obtain more information regarding the options and parameters available for the operation you're trying to execute.

3.3 Token Files

Caliente has a feature to facilitate both the specification of command-line parameters for scripting, as well as dynamic generation of command-line parameters. This feature is also used for some specific parameters (notably, the ***--from*** parameter during extraction) due to its flexibility.

If you wish to manage parameters as a group, you can put them all in a parameter file, and the file will be parsed and processed just as if it were inline in the command line. Importantly, multiple lines are ignored as separators, end-of-line comments are supported (with #), as is line continuation (the last character of the line is \, and means to append the next line directly there).

Here is an example of a token file’s contents for Caliente parameters:

```
# This is an end-of-line comment and will be ignored
--engine dctm export
--data ${target-directory}
--from “/some-path”
--from “/another path”
--from “dm_document where condition = something”
#--from “dm_document where state = disabled”

# Read more parameters from another token file relative to this one
--@ extra-parameters.txt

# Read more parameters from another token file using an absolute path
--@ /backups/backup-parameters.txt

# Read more parameters from a URL instead of a file
--@ http://server.com/generated-parameters.txt
```

Note the following:

- Empty lines are ignored
- Unquoted/unescaped spaces are ignored
- The ***#*** character marks an end-of-line comment: anything after that in any given line is ignored
- You can reference other files using --@, and if their path is relative it will be calculated relative to the file referencing them. You can also use any URL that the default Java libraries support
- Recursive reference loops result in a parsing error (i.e. --@ a -> --@ b -> --@ a == error)
- The engine is smart enough to not read files twice, keeping them cached in memory as it goes, which helps performance by avoiding unnecessary work repetition
- If a --@ reference points to a missing or unreadable file, or URL that can’t be fetched, this results in a parsing error

So, for instance, a valid invocation could be:

```
$ java –jar caliente-exe.jar --@ parameter-details.txt
```

Such that all parameters are stored and managed inside the token file.

These types of token files are also supported for some specific parameters, most notably the ***--from*** extraction parameter (see below). In this case, the syntax is slightly different:

```
# Extract the stuff from Quality Control, selected by path
/Quality Control

# Extract the Monthly Reports based on a CMS-specific query
Select * from dm_document \
    where folder('/Monthly Reports', DESCEND) \
      and is_final = 1

# Extract specific documents by ID
%09de75d18002250c
%09de75d18002a07c
%09de75d18002986d
%09de75d18002a0f6
# %09de75d18002a0f5 # this one is disabled by comment
%09de75d180020797
%09de75d18002ad0a
%09de75d18002078f
%09de75d18002078d
%09de75d18002ad0b
%09de75d180029878

# Include more listings from other files

# This file will be located relative to the path of the current file
@more-extractions.txt

# This file will be located via its absolute path
@/home/billy/yet-more-extractions.txt

# Include more extractions from a URL
@http://server.com/extraction-list-extras.txt
```

The same rules apply as for the other token file, with the following distinctions:

- Instead of --@ *target*, the syntax is @*target* (no space in between)
- An additional URL type is supported for classpath resources using one of the following URL schemas: classpath, cp, resource, res
- Line continuation is supported such that using \ as the last character in a line causes the next line to be appended in place
- Each line is interpreted as a single value for the --from parameter, which is why line continuations may be important where longer language-specific queries and readability are concerned

4 Content Export

4.1 Basic Export Format

Caliente's extraction engines all use a similar syntax for extraction:

```
$ java –jar caliente-exe.jar --engine ${ENGINE} export \
   --data ${target} \
   --from ${source-content-spec} \
   --threads ${thread-count} ...
```

- --engine ${engine} ***(required)***
    - ${engine} = The CMS engine to use. ${engine} may be one of dctm (Documentum), shpt (SharePoint via JShare), local (Local FS), cmis (CMIS 1.1), or ucm (WebCenter 11G+)
- export ***(required)***
    - Enable extraction mode – common for all engines
- --data ${target}
    - ${target} = The target directory to use as the root for all Caliente data when alternate locations aren't specified. This path will always be created as some control files may need to be created there.
- --from ${source-content-spec}
    - ${source-content-spec} = An engine-specific specification for the source content to be extracted. Some engines support path-based searching, others support searching by object ID, and yet others support query syntax. Some engines even support specifying multiple extraction sources for a single operation. You'll need to consult with each engine's help information for more details. This parameter supports referencing token files by having the ***source-content-spec*** start with @ (i.e. @/some/file/path.txt or @http://server.com/extraction-source.list)
- --threads ${thread-count}
    - ${thread-count} = The number of threads to use for extraction. The default is calculated as twice the number of active CPU cores, the minimum is 1 (not recommended except for test purposes), and the maximum allowed is four times the number of CPU cores. This is because of diminishing returns: at some point adding threads becomes a bottleneck rather than a benefit.
- Other Parameters
    - Engine dependent

4.2 Documentum Usage

Each specific engine has its own set of parameters it accepts for a given operation, as well as custom rules that govern them. These settings are fairly extensive, but we'll try to summarize the more common ones here for convenience. This section will offer up examples for extracting from Documentum.

The Documentum engine is by far the most mature component in the Caliente suite. It requires an external DFC installation, and supports interacting with any Documentum CS that the DFC instance is compatible with. Caliente is compatible with DFC versions 6.7SP1 up to 7.1. Newer versions should also be compatible but are untested.

Documentum Parameters:

- --from ${source-spec}
  - ${source-spec} = May be a path specification (if it starts with /), an object ID (preceded with a %), or a DQL predicate (any other starting character). Multiple source specs, using multiple --from parameter instances, are supported. A file containing one search spec per line (supporting EOL comments with #) can also be specified by preceding its path with @ (i.e. @/home/extract-contents.list)
  - For the predicate extraction, you can assume the predicate will be used as part of a DQL query like so: ***select distinct r_object_id from ${predicate}***
- --dctm-unified
  - Enable the use of Documentum Unified Mode login (raises an error if a password is specified using --password)
- --dctm ${path}
  - ${path} = The path to use as an alternative user Documentum folder, in case the default one (identified via the ***DOCUMENTUM*** environment variable) is not to be used. If there is no ***DOCUMENTUM*** environment variable defined, this parameter is required.
- --dfc ${path}
  - ${path} = The path where the DFC is installed that should be used, instead of the default one pointed to by the ***DOCUMENTUM_SHARED*** environment variable. If there is no ***DOCUMENTUM_SHARED*** environment variable, this parameter is required.
- --dfc-prop ${path}
  - ${path} = The path to the ***dfc.properties*** file that should be used, in case the default one (using the DFC-defined default search algorithm) is not to be used.
- --server ${server-connection-info}
  - ${server-connection-info} = Information that allows Caliente to select which Documentum server to connect to, usually a Docbase name

- --user ${username}
  - ${username} = The username with which to authenticate to the Documentum Content Server
- --password ${password}
  - ${password} = The password with which to authenticate to the Documentum Content Server (see Advanced Usage to learn how you may encrypt this value so it may be safely stored in scripts and runbooks)

4.2.1 Sample Exports

- Export all documents in the ***CalienteTests*** cabinet using a predicate

```
$ java –jar caliente-exe.jar –engine dctm export --server TestCS \
   --user dmadmin --password XZ6ZkRzrHEgzZkX= \
   --from “dm_document where folder(‘/CalienteTests’, DESCEND)” \
   --data extract-test
```

  - This will connect to the docbase ***TestCS*** as the user ***dmadmin***, using the (decrypted) given password
  - Caliente will then run the DQL query: ***select r_object_id from*** *dm_document where folder(‘/CalienteTests’, DESCEND)*
  - The extraction will be stored in the directory ***extract-test***, relative to the directory Caliente was executed from (i.e. the Java system property ***user.dir***)

- Export only documents of the custom subtype arm_accounting_doc for which the “archive_ready” attribute is true

```
$ java –jar caliente-exe.jar –engine dctm export --server Accounting \
   --user account_master --password XZ6ZkRzrHEgzZkX= \
   --from “arm_accounting_doc where archive_ready = 1” \
   --data accounting-archive
```

  - This will connect to the docbase ***Accounting*** as the user ***account_master***, using the (decrypted) given password
  - Caliente will then run the DQL query: ***select r_object_id from*** *arm_accounting_doc where archive_ready = 1*
  - The extraction will be stored in the directory ***accounting-archive***, relative to the directory Caliente was executed from (i.e. the Java system property ***user.dir***)

- Export all documents from the cabinets “Quality Control”, “Monthly Reports”, and from the folder “Audit Details/2019”

```
$ java –jar caliente-exe.jar –engine dctm export --server AuditRepo \
   --user bob.jenkins --password XZ6ZkRzrHEgzZkX= \
   --from “/Quality Control” --from “/Monthly Reports” --from “/Audit Details/2019” \
   --data audit-sample
```

  - This will connect to the docbase ***AuditRepo*** as the user ***bob.jenkins***, using the (decrypted) given password
  - Caliente will then find each of the given folders by path (using *IDfSession.getObjectByPath()*)
  - The extraction will be stored in the directory ***audit-sample***, relative to the directory Caliente was executed from (i.e. the Java system property ***user.dir***)

- You can also mix extraction sources (--from) by asking to extract by a given path, by ID, and by predicate all at once – Caliente will not export objects twice regardless of how many times the object is encountered as part of the source extraction set
- Finally, a “source list” file can be used to specify multiple different types of sources without having to include them piecemeal through the command line, like so:

```
$ java –jar caliente-exe.jar –engine dctm export --server AuditRepo \
   --user bob.jenkins --password XZ6ZkRzrHEgzZkX= \
   --from @extraction-list.txt --data audit-sample
```

  - These would be the contents of *extraction-list.txt*, and would result in the exact same extraction as the previous example:

```
# Extract the stuff from Quality Control
/Quality Control

# Extract the Monthly Reports
/Monthly Reports

# Extract the Audit Details for FY2019
/Audit Details/2019
```

  - See the section on Token Files, above, for more details on the format and features supported by this type of file

5 Content Import

5.1 Basic import format

Caliente's import engines all use a similar syntax for extraction:

```
$ java –jar caliente-exe.jar --engine ${ENGINE} import \
--data ${source} \
--target ${target-location} \
--threads ${thread-count} ...
```

- --engine ${engine} ***(required)***
  - ${engine} = The CMS engine to use. ${engine} may be one of dctm (Documentum), alfrescobi (Alfresco Bulk Importer), local (Local FS), cmis (CMIS 1.1), or xml (Sharepoint 2013 via the .Net XML importer)
- import ***(required)***
  - Enable ingestion mode – common for all engines
- --data ${source}
  - ${source} = The source directory to use as the root for all Caliente data when alternate locations aren't specified. This path will always be created as some control files may need to be created there.
- --target ${target-location}
  - ${target-location} = The location within the target environment where the imported data is to be stored. Supported by most, but not all engine
- --threads ${thread-count}
  - ${thread-count} = The number of threads to use for extraction. The default is calculated as twice the number of active CPU cores, the minimum is 1 (not recommended except for test purposes), and there is no enforced maximum, although operators should seek to be cautious when specifying thread counts higher than the default as this may result in significant overhead.
- Other Parameters
  - Engine dependent

5.2 Documentum Usage

Each specific engine has its own set of parameters it accepts for a given operation, as well as custom rules that govern them. These settings are fairly extensive, but we'll try to summarize the more common ones here for convenience. This section will offer up examples for ingesting into Documentum.

The Documentum engine is by far the most mature component in the Caliente suite. It requires an external DFC installation, and supports interacting with any Documentum CS that the DFC instance is compatible with. Caliente is compatible with DFC versions 6.7SP1 up to 7.1. Newer versions should also be compatible but are untested.

Documentum Parameters:

- --dctm-unified
  - Enable the use of Documentum Unified Mode login (raises an error if a password is specified using --password)
- --dctm ${path}
  - ${path} = The path to use as an alternative user Documentum folder, in case the default one (identified via the ***DOCUMENTUM*** environment variable) is not to be used. If there is no ***DOCUMENTUM*** environment variable defined, this parameter is required.
- --dfc ${path}
  - ${path} = The path where the DFC is installed that should be used, instead of the default one pointed to by the ***DOCUMENTUM_SHARED*** environment variable. If there is no ***DOCUMENTUM_SHARED*** environment variable, this parameter is required.
- --dfc-prop ${path}
  - ${path} = The path to the ***dfc.properties*** file that should be used, in case the default one (using the DFC-defined default search algorithm) is not to be used.
- --server ${server-connection-info}
  - ${server-connection-info} = Information that allows Caliente to select which Documentum server to connect to, usually a Docbase name
- --user ${username}
  - ${username} = The username with which to authenticate to the Documentum Content Server
- --password ${password}
  - ${password} = The password with which to authenticate to the Documentum Content Server (see Advanced Usage to learn how you may encrypt this value so it may be safely stored in scripts and runbooks)

5.2.2 Sample Ingestions

- Import the extraction in ***extracted-data*** into the ***IngestionTest*** docbase, inside the ***CalienteTests*** cabinet (***NOTE***: *most* of the time you'll want to run ingestions as a Documentum superuser like ***dmadmin***):

```
$ java –jar caliente-exe.jar –engine dctm import --server IngestionTest \
    --user dmadmin --password XZ6ZkRzrHEgzZkX= \
    --target "/Imported Data/Test Ingestion" \
    --data extracted-data
```

  - This will connect to the docbase ***IngestionTest*** as the user ***dmadmin***, using the (decrypted) given password
  - Caliente will then begin scanning all the extracted data within the ***extracted-data*** directory, and re-creating them within Documentum to the best of its ability
  - The ingested data will be stored within the ***/Imported Data/Test Ingestion*** folder (which will be created if it does not exist), with all paths being transparently adjusted to be relative to that location, including the paths for any top-level objects contained in the extraction.

6 Advanced Usage

6.1 Password encryption/decryption

Each engine can support its own encryption/decryption schemes in order to facilitate password management without compromising security. The mode used is ***encrypt*** or ***decrypt***, as follows:

```
$ java –jar caliente-exe.jar --engine ${engine} ${mode}
```

Where ***${mode}*** can be either **encrypt** or **decrypt**, and ***${engine}*** is the engine you plan to use for import or extraction. Caliente will then present you with a screen similar to this one:

```
2020-02-20 14:50:16,288 [main            ] Launching Caliente v3.9.5 encrypt mode for
engine dctm


2020-02-20 14:50:16,288 [main            ] Current heap size: 960 MB

2020-02-20 14:50:16,288 [main            ] Maximum heap size: 14233 MB

2020-02-20 14:50:16,289 [main            ] Caliente CLI v3.9.5

2020-02-20 14:50:16,289 [main            ] Classpath addition:
[file:/home/diego/documentum/config]

2020-02-20 14:50:16,289 [main            ] Classpath addition: [file:/apps/documentum-
6.7-SP2/dctm.jar]

Enter the password that you would like to ${mode} (it will not be shown):

Encrypted Value (in brackets) = [${THE_ENCRYPTED_VALUE}]

2020-02-20 14:50:18,286 [main            ] Closing up all internal stores...

2020-02-20 14:50:18,288 [main            ] The internal data stores were closed
cleanly.
```

In encryption mode, the password is read without echoing to the console, and the encrypted value is output at the end. Alternatively Caliente will read the passwords to be encrypted from STDIN, with each line representing a password to encrypt (no trimming is done, EOL characters are ignored). The passwords will be output in the order that they're read from STDIN.

In decryption mode, the encrypted password is echoed to the console as you type, and the decrypted value is output to the screen as well. Similarly to the encyrptor Caliente may also read encrypted values from STDIN, one per line, and will output the decrypted values one by one.

Use this mode to better secure your usage of Caliente within scripts, while also enjoying the convenience of storing passwords for quick access.