MarkLogic Server

mlcp User Guide

MarkLogic 9
May, 2017

Last Revised: 9.0-2, July, 2017

Table of Contents

mlcp User Guide

1.0 Introduction to MarkLogic Content Pump

MarkLogic Content Pump (mlcp) is a command line tool for getting data into and out of a MarkLogic Server database. This chapter covers the following topics:

1.1 Feature Overview

Using mlcp, you can import documents and metadata to a database, export documents and metadata from a database, or copy documents and metadata from one database to another. For example:

- Import content into a MarkLogic Server database from flat files, compressed ZIP and GZIP files, or mlcp database archives.
- Create documents from flat files, delimited text files, Hadoop sequence files, aggregate XML files, and line-delimited JSON files. For details, see “Importing Content Into MarkLogic Server” on page 30.
- Import mixed content types from a directory, using the file suffix and MIME type mappings to determine document type. Unrecognized/missing suffixes are imported as binary documents. For details, see “How mlcp Determines Document Type” on page 36.
- Export the contents of a MarkLogic Server database to flat files, a compressed ZIP file, or an mlcp database archive. For details, see “Exporting Content from MarkLogic Server” on page 95.
- Copy content and metadata from one MarkLogic Server database to another. For details, see “Copying Content Between Databases” on page 122.
- Import or copy content into a MarkLogic Server database, applying a custom server-side transformation before inserting each document. For details, see “Transforming Content During Ingestion” on page 66.
- Extract documents from an archived forest to flat files or a compressed file using Direct Access. For details, see “Using Direct Access to Extract or Copy Documents” on page 135.
- Import documents from an archived forest into a live database using Direct Access. For details, see “Importing Documents from a Forest into a Database” on page 139.

The mlcp tool has two modes of operation:

- Local: mlcp drives all its work on the host where it is invoked. Resources such as import data and export destination must be reachable from that host.
- Distributed: mlcp distributes its workloads across the nodes in a Hadoop cluster. Resources such as import data and export destination must be reachable from the cluster, which usually means via HDFS.

Local mode is the default unless you configure your environment or mlcp command line as described in “Configuring Distributed Mode” on page 15. Distributed mode requires a Hadoop installation.

To understand the difference between the two modes, consider the following: When loading documents in local mode, all the input data must be reachable from the host on whichyou run mlcp, and all communication with MarkLogic Server is through that host. Throughput is limited by resources such as memory and network bandwidth available to the host running mlcp. When loading documents in distributed mode, multiple nodes in a Hadoop cluster communicate with MarkLogic Server, so greater concurrency can be achieved, while placing fewer resource demands on any one host.

You can use mlcp even when a load balancer sits between the client host and the MarkLogic host. The mlcp tool is compatible with AWS Elastic Load Balancer (ELB) and other load balancers.

1.2 Terms and Definitions

You should be familiar with the following terms and definitions when using mlcp:

Term	Definition
aggregate	XML content that includes recurring element names and which can be split into multiple documents with the recurring element as the document root. For details, see “Splitting Large XML Files Into Multiple Documents” on page 42.
line-delimited JSON	A type of aggregate input where each line in the file is a piece of stand-alone JSON content. For details, see “Creating Documents from Line-Delimited JSON Files” on page 48.
archive	A compressed MarkLogic Server database archive created using the mlcp export command. You can use an archive to restore or copy database content and metadata with the mlcp import command. For details, see “Exporting to an Archive” on page 97.
HDFS	The Hadoop Distributed File System, which can be used as an input source or an output destination in distributed mode.

Term	Definition
sequence file	A flat file of binary key-value pairs in one of the Apache Hadoop `SequenceFile` formats. The mlcp tool only supports importing `Text` and `BytesWritable` values from a sequence file.
split	The unit of work for one thread in local mode or one MapReduce task in distributed mode.

1.3 Modifying the Example Commands for Windows

All the examples in this guide use Unix command line syntax. If you are using mlcp with the Windows command interpreter, `Cmd.exe`, use the following guidelines to construct equivalent commands:

- Replace `mlcp.sh` with `mlcp.bat`. You should always use `mlcp.bat` on Windows; using `mlcp.sh` with Cygwin is not supported.
- For aesthetic reasons, long example command lines are broken into multiple lines using the Unix line continuation character “\”. On Windows, remove the line continuation characters and place the entire command on one line, or replace the line continuation characters with the Windows equivalent, “^”.
- Replace option arguments enclosed in single quotes (') with double quotes ("). If the single-quoted string contains embedded double quotes, escape the inner quotes.
- Escape any unescaped characters that have special meaning to the Windows command interpreter.

For example, the following Unix command line:

```
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password passwd -input_file_path /space/bill/data -mode local \
    -output_uri_replace "/space,'',/bill/data/,'/will/'" \
    -output_uri_prefix /plays
```

Corresponds to this Windows command line:

```
C:\Example> mlcp.bat import -host localhost -port 8000 -username user ^
    -password passwd -input_file_path c:\space\bill -mode local ^
    -output_uri_replace "/c:/space,'',/bill/data/,'/will/'" ^
    -output_uri_prefix /plays
```

1.4 Command Line Summary

The mlcp command line has the following structure. Note that you should always use `mlcp.bat` on Windows; using `mlcp.sh` with Cygwin is not supported.

- Linux and OS X: `mlcp.sh` *`command options`*

- Windows: `mlcp.bat` *`command options`*

Where *`command`* is one of the commands in the table below. Each command has a set of command-specific options, which are covered in the chapter that discusses the command.

Command	Description
import	Import data from the file system, the Hadoop Distributed File System (HDFS), or standard input to a MarkLogic Server database. For a list of options usable with this command, see "Import Command Line Options" on page 84.
export	Export data from a MarkLogic Server database to the file system or HDFS. For a list of options usable with this command, see "Export Command Line Options" on page 117.
copy	Copy data from one MarkLogic Server database to another. For a list of options usable with this command, see "Copy Command Line Options" on page 128.
extract	Use Direct Access to extract files from a forest file to documents on the native file system or HDFS. For a list of options usable with this command, see "Extract Command Line Options" on page 140.
version	Report mlcp runtime environment version information, including the mlcp, JRE, and Hadoop versions, as well as the supported MarkLogic version.
help	Display brief help about mlcp.

In addition to the command-specific options, mlcp enables you to pass additional settings to Hadoop MapReduce when using `-mode distributed`. This feature is for advanced users who are familiar with MapReduce. For details, see "Setting Custom Hadoop Options and Properties" on page 16.

> **Note:** If you use Hadoop-specific options such as `-conf` or `-D`, they must appear after `-options_file` (if present) and before any mlcp-specific options.

Options can also be specified in an options file using `-options_file`. Options files and command line options can be used together. For details, see "Options File Syntax" on page 9.

Note the following conventions for command line options to mlcp:

- Prefix options with a single dash (-).
- Option names are case-sensitive.

- If an option has a value, separate the option name and value with whitespace. For example: `mlcp import -username admin`
- If an option has a predefined set of possible values, such as `-mode`, the option values are case-insensitive unless otherwise noted.
- If an option appears more than once on the command line, the first occurrence is used.
- When string option values require quoting, use single quotes. For example: `-output_uri_replace "this,'that '"`.
- The value of a boolean typed option can be omitted. If the value is omitted, true is implied. For example, `-copy_collections` is equivalent to `-copy_collections true`.

1.4.1 Setting Java Virtual Machine (JVM) Options

The mlcp tool is a Java application. You can pass extra parameters to the JVM during an mlcp command using the environment variable `JVM_OPTS`.

For example, the following command passes the setting "-Xmx100M" to the JVM to increase the JVM heap size for a single mclp run:

```
$ JVM_OPTS='-Xmx100M' mclp.sh import ...
```

1.4.2 Regular Expression Syntax

For `-input_file_path`, use the regular expression syntax outlined here:

http://hadoop.apache.org/docs/stable/api/org/apache/hadoop/fs/FileSystem.html#globStatus(org.apache.hadoop.fs.Path)

For all other options that use regular expressions, such as `-input_file_pattern`, use the Java regular expression language. Java's pattern language is similar to the Perl pattern language. For details on the grammar, see the documentation for the Java class `java.util.regex.Pattern`:

http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html

For a tutorial on the expression language, see http://docs.oracle.com/javase/tutorial/essential/regex/.

1.4.3 Options File Syntax

You can specify mlcp options using an options file, in addition to using command line options by using `-options_file`. Using an options file is especially convenient when working with options whose values contain quotes and other special characters that are difficult to escape on the command line.

If you use an options file, it must be the first option on the command line. The mlcp command (import, export, copy) can also go inside the options file. For example:

```
$ mlcp.sh -options_file my_options.txt -input_file_path /example
```

An options file has the following contents:

- Each line contains either a command name, an option, or an option value, ordered as they would appear on the command line.
- Comments begin with “#” and must be on a line by themselves.
- Blank lines, leading whitespace, and trailing whitespace are ignored.

For example, if you frequently use the same MarkLogic Server connection information (host, port, username, and password), you can put the this information into an options file:

```
$ cat my-conn.txt
# my connection info
-host
localhost
-port
8000
-username
me
-password
my_password

# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -options_file my-conn.txt \
    -input_file_path /space/examples/all.zip
```

This is equivalent to the following command line without an options file:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username me \
    -password my_password -input_file_path /space/examples/all.zip
```

You can also include a command name (import, export, or copy) as the first non-comment line in an options file:

```
# my connection info for import
import
-host
localhost
-port
8000
-username
me
-password
my_password
```

1.5 Compatibility of mlcp Across MarkLogic Versions

Unless otherwise noted, mlcp is compatible with a wide range of MarkLogic versions. That is, you can usually use a recent version of mlcp with and older version of MarkLogic and vice versa. However, not all features of mlcp or MarkLogic will work across version boundaries.

For example, MarkLogic 9 and mlcp 9.0 include support for redacting documents as you export them. However, older versions of MarkLogic do not support this feature, so it is not possible to use the `-redaction` option of mlcp with older versions.

Similarly, you can use mlcp to export a database archive from MarkLogic 9 or later that includes documents with the `node-update` security capability. However, this capability did not exist in earlier versions of MarkLogic, so it cannot be preserved if you import the MarkLogic 9 archive into an older MarkLogic, and may even cause errors.

For best results, use the version of mlcp that corresponds to your version of MarkLogic, or limit your jobs to features you know are supported in both.

1.6 Accessing the mlcp Source Code

The mlcp tool is developed and maintained as an open source project on GitHub. To access the sources or contribute to the project, navigate to the following URL in your browser:

http://github.com/marklogic/marklogic-contentpump

2.0 Installation and Configuration

This chapter describes how to install mlcp and configure your client environment and MarkLogic for most effective use of the tool. The following topics are included:

2.1 Supported Platforms

In local mode, mlcp is supported on the same platforms as MarkLogic Server, including 64-bit Linux, 64-bit Windows, and Macintosh OS X. For details, see Supported Platforms in the *Installation Guide*.

Distributed mode is only supported on 64-bit Linux.

2.2 Required Software

The following software is required to use mlcp:

- MarkLogic Server 7.0-1 or later, with an XDBC App Server configured. MarkLogic 8 and later versions come with an XDBC App Server pre-configured on port 8000.
- Oracle/Sun Java JRE 1.8 or later.

  Note: Apache Hadoop only supports the Oracle/Sun JRE, though other JRE's may work. For details, see http://wiki.apache.org/hadoop/HadoopJavaVersions.

In distributed mode, mlcp requires access to an installation of one of the following Hadoop MapReduce distributions. The mlcp tool might work with other distributions based on Apache Hadoop v2.6.

- Cloudera’s Distribution Including Apache Hadoop (CDH) version 5.8
- Hortonworks Data Platform (HDP) version 2.4
- MapR version 5.1. Using mlcp with MapR requires special setup. For details, see “Using mlcp With MapR” on page 16.

2.3 Installing mlcp

After downloading mlcp, follow these instructions to install mlcp.

1. Download mlcp from http://developer.marklogic.com/products/mlcp.

2. Unpack the mlcp distribution to a location of your choice. This creates a directory named `mlcp-`*`version`*, where *version* is the mlcp version. For example, assuming `/space/marklogic` contains zip file for mlcp version 1.3, then the following commands install mclp under `/space/marklogic/mlcp-1.3/`:

```
$ cd /space/marklogic
$ unzip mlcp-1.3-bin.zip
```

3. Optionally, put the mlcp bin directory on your path. For example:

```
$ export PATH=${PATH}:/space/marklogic/mlcp-1.3/bin
```

4. Put the `java` command on your path. For example:

```
$ export PATH=${PATH}:$JAVA_HOME/bin
```

5. If you plan to use mlcp in distributed mode, you must have a Hadoop installation and must configure your environment so mlcp can find your Hadoop installation. For details, see “Configuring Distributed Mode” on page 15.

You might need to configure your MarkLogic cluster before using mlcp for the first time. For details, see “Configuring Your MarkLogic Cluster” on page 13.

On Windows, use the `mlcp.bat` command to run mlcp. On Unix and Linux, use the `mlcp.sh` command. You should not use `mlcp.sh` in the Cygwin shell environment on Windows.

2.4 Configuring Your MarkLogic Cluster

The mlcp tool uses an XDBC App Server to communicate with each host in a MarkLogic Server cluster that has at least one forest attached to a database used in your mlcp job.

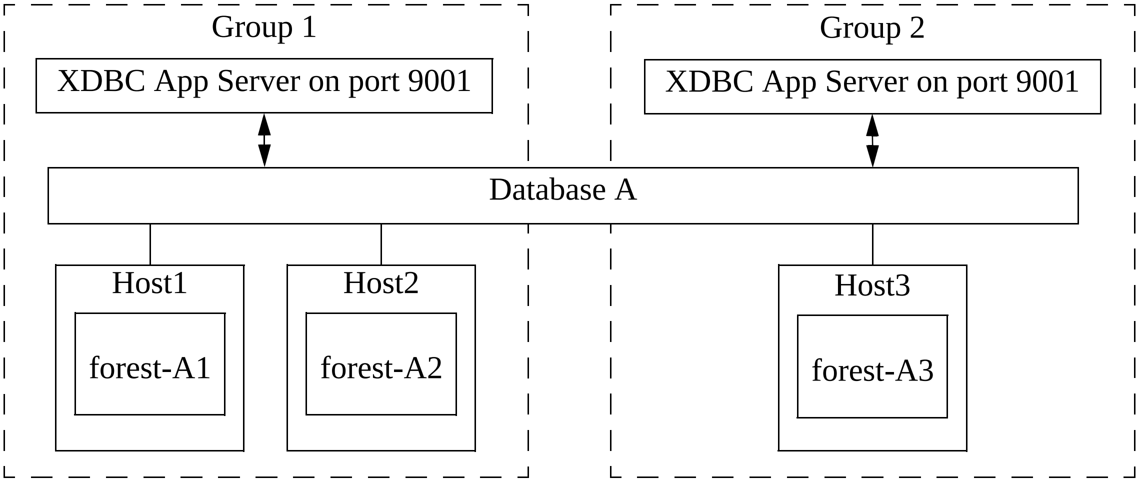
When you use mlcp with MarkLogic 8 or later on the default port (8000), no special cluster configuration is necessary. Port 8000 includes a pre-configured XDBC App Server. The default database associated with port 8000 is the Documents database. To use mlcp with a different database and port 8000, use the `-database`, `-input_database`, or `-output_database` options. For example:

```
mlcp.sh import -host myhost -port 8000 -database mydatabase ...
```

When using MarkLogic 7 or earlier on any port, or when using MarkLogic 8 or later with a port other than 8000, you must make an XDBC App Server available to each host that has at least one forest attached to the database(s) used by your job. Hosts within a group share the same App Server configuration, but hosts in different groups do not.

Therefore, if all your forest hosts are in a single group, you only need to configure one XDBC App Server. If your forests are on hosts in multiple groups, then you must configure an XDBC App Server listening on the same port in each group.

For example, the cluster shown below is properly configured to use Database A as an mlcp input or output source. Database A has 3 forests, located on 3 hosts in 2 different groups. Therefore, both Group 1 and Group 2 must make Database A accessible via XDBC on port 9001.



If the forests of Database A are only located on Host1 and Host2, which are in the same group, then you would only need to configure one XDBC App Server on port 9001.

If you use MarkLogic 8 or later and port 8000 instead of port 9001, then you would not need to explicitly create any XDBC App Servers to support the above database configuration because both group automatically have an XDBC App Server on port 8000. You might need to explicitly specify the database name (Database A) in your mlcp command, though, if it is not the default database associated with port 8000.

2.5 Security Considerations

When you use mlcp, you supply the name of a user(s) with which to interact with MarkLogic Server. If the user does not have admin privileges, then the user must have at least the privileges listed in the table below.

Note: Additional privileges may be required. These roles only enable use of MarkLogic Server as a data source or destination. For example, these roles do not grant read or update permissions to the database.

mlcp Command	Privilege	Notes
import	`hadoop-user-write`	Applies to the user name specified with `-username`. It is recommended that you also set `-output_permissions` to set the permissions on inserted documents.
export	`hadoop-user-read`	Applies to the user name specified with `-username`.
copy	`hadoop-user-read` (input) `hadoop-user-write` (output)	The `-input_username` user have the `hadoop-user-read` privilege on source MarkLogic Server instance. The `-output_username` user must have the `hadoop-user-write` privilege on destination MarkLogic Server instance.

By default, mlcp requires a username and password to be included in the command line options for each job. You can avoid passing a cleartext password between your mlcp client host and MarkLogic Server by using Kerberos for authentication. For details, see “Using mlcp With Kerberos” on page 19.

2.6 Configuring Distributed Mode

Distributed mode enables mlcp to distribute its workload across a Hadoop cluster. Using mlcp in distributed mode requires a Hadoop installation. For information on supported versions, see “Required Software” on page 12.

Hadoop does not have to be installed on the host where you run mlcp, but the Hadoop configuration files must be reachable by mlcp.

This sections covers the following topics related to using mlcp in distributed mode:

- Specifying the Hadoop Configuration File Location
- Setting Custom Hadoop Options and Properties
- Required Hadoop User Privileges
- Using mlcp With MapR

Note: Some versions of Hadoop and HDFS have problems with pathnames that contain spaces, so it is recommended that you do not use mlcp in distributed mode with input or output file pathnames that contain whitespace.

2.6.1 Specifying the Hadoop Configuration File Location

You must tell mlcp where to find the Hadoop configuration files on the host where you run mlcp. Hadoop does not need to be installed on this host, but the Hadoop configuration files must be reachable.

Use one of the following methods to tell mlcp where to find your Hadoop configuration files locally:

- Set the mlcp command line option `-hadoop_conf_dir`. For example:

  ```
  $ mlcp.sh command -hadoop_conf_dir /etc/hadoop/conf
  ```

- Set the environment variable `HADOOP_CONF_DIR`. For example:

  ```
  $ export HADOOP_CONF_DIR=/etc/hadoop/conf
  ```

If your Apache Hadoop installation is on a remote host, you can copy the configuration files locally and set `HADOOP_CONF_DIR` (or `-hadoop_conf_dir`) to that directory.

2.6.2 Setting Custom Hadoop Options and Properties

Use the following mlcp options to pass Hadoop-specific configuration information through mlcp to your Hadoop installation. You can use these options to control mlcp's use of Hadoop in distributed mode.

- `-conf` *`conf_filename`* : Pass in a Hadoop configuration properties file.
- `-D` *`property=value`* : Pass one Hadoop configuration property setting.

The property names and configuration file syntax is as dictated by Hadoop. For details, consult the documentation for your Hadoop distribution.

Note: These Hadoop options must appear on the command line after `-options_file` (if present) and before any other mlcp-specific options.

2.6.3 Required Hadoop User Privileges

When you use distributed mode for import, the user your Hadoop tasks run as must have permission to access the directories or files specified by `-input_file_path`. Similarly, when you use distributed mode for export or extract, the user must have permission to create directories and files in the directory specified by `-output_file_path`.

2.6.4 Using mlcp With MapR

To use MapR as mlcp's Hadoop distribution, you must download the `mlcp-mapr-`*`version`*`-bin` bundle instead of the standard mlcp bundle. For example, download `mlcp-mapr-9.0-bin.zip` from developer.marklogic.com.

You must also separately download the `maprfs` Java JAR file from MapR and make it available to mlcp. You can use the following procedure:

1. Download `maprfs-`*`version`*`-mapr.jar` from `http://repository.mapr.com/nexus/content/groups/mapr-public/com/mapr/hadoop/maprfs/`*`version`*`-mapr/`. Modify *`version`* to match your MapR version.

   For example, download `maprfs-5.1.0-mapr.jar` from `http://repository.mapr.com/nexus/content/groups/mapr-public/com/mapr/hadoop/maprfs/5.1.0-mapr/`.

2. Make the JAR file available to mlcp:

   a. If you have write access to your mlcp installation directory, place the JAR file in *`MLCP_INSTALL_DIR`*`/lib`.

   b. If you do not have write access to your mlcp installation directory, then ensure the JAR file is on your Java classpath when running mlcp.

To avoid warnings about JCE policy files when using mlcp with MapR, you should also install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files compatible with your JRE.

For example, you can download the Java 8 policy files for the Oracle JRE from the following location:

http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html

2.7 Connecting to MarkLogic Using SSL

When you connect to a MarkLogic App Server with mlcp, you can use an SSL-enabled connection to secure the communications. This applies to the `import`, `export`, and `copy` mlcp commands.

- Enabling SSL on Your App Server
- Configuring mlcp to Use SSL
- Advanced SSL Configuration

2.7.1 Enabling SSL on Your App Server

You can only use SSL to connect to MarkLogic through an SSL-enabled App Server. For more details, see Configuring SSL on App Servers in the *Security Guide*.

If you want to use SSL with both the source (input) and destination (output) App Servers during an mlcp `copy` job, both App Servers must be SSL enabled.

2.7.2 Configuring mlcp to Use SSL

By default, mlcp does not connect to MarkLogic using SSL. Use one of the following options to specify that mlcp should connect via SSL:

mlcp Command	Command Line Option	For more information
import	`-ssl`	"Import Command Line Options" on page 84
export	`-ssl`	"Export Command Line Options" on page 117
copy	`-input_ssl` and/or `-output_ssl`	"Copy Command Line Options" on page 128

All these options accept a boolean argument value. As described in "Command Line Summary" on page 7, "true" is assumed if you leave the argument off.

2.7.3 Advanced SSL Configuration

An advanced user can specify detailed SSL configuration options by following the procedure in this section.

Internally, mlcp uses the MarkLogic Connector for Hadoop to connect to MarkLogic and move data as required by your job. If you make an implementation of `com.marklogic.mapreduce.SslConfigOptions` available to the connector (through mlcp), then your mlcp job will use the configuration specified by that implementation.

Use the following procedure to create an SSL config class and use it with mlcp:

1. Create a class that implements `com.marklogc.mapreduce.SslConfigOptions` from the MarkLogic Connector for Hadoop. Your implementation should use one of the techniques described in Accessing SSL-Enabled XDBC App Servers in the *XCC Developer's Guide* to supply an `SSLContext` object through the `getSslContext` method.

2. Create a configuration file that tells the connector to use your `SslConfigOptions` class to configure SSL connections. See the example below.

3. Ensure your `SslConfigOptions` implementation is on the classpath for your mlcp job.

4. Use the `-conf` mlcp command line option to tell mlcp where to find the configuration file from Step 2. For example:

```
mlcp import -conf yourConfFile otherMlcpOptions
```

The following is an example of a MarkLogic Connector for Hadoop configuration for using a custom SSL configuration for an mlcp `import` job. Substitute your class name for *`yourSslOptions`* in the property value.

```
<property>
  <name>mapreduce.marklogic.output.ssloptionsclass</name>
  <value>yourSslOptions.class</value>
</property>
```

The following is an example configuration file for an `export` job. Substitute your class name for *`yourSslOptions`* in the property value.

```
<property>
  <name>mapreduce.marklogic.input.ssloptionsclass</name>
  <value>yourSslOptions.class</value>
</property>
```

For a copy job, set either or both of `mapreduce.marklogic.output.ssloptionsclass` or `mapreduce.marklogic.input.ssloptionsclass`, depending on which connections you wish to secure with SSL.

> **Note:** If you set the property `mapreduce.marklogic.input.usessl` or `mapreduce.marklogic.output.usessl` in the configuration file as well as using th -ssl mlcp option, the mlcp command line options take precedence over the configuration file property setting.

For more details, see Making a Secure Connection to MarkLogic Server with SSL in the *MarkLogic Connector for Hadoop Developer's Guide* and the *MarkLogic Hadoop MapReduce Connector API*.

For an example of a custom SSL configuration implementation, see the `ContentReader` example for the MarkLogic Connector for Hadoop. For more details, see ContentReader in the *MarkLogic Connector for Hadoop Developer's Guide*. The example source code is available in the following ways:

- Download the MarkLogic Connector for Hadoop distribution from http://developer.marklogic.com/products/hadoop,
- Browse or clone a copy of the sources in the GitHub `marklogic-contentpump` project at http://github.com/marklogic/marklogic-contentpump.

2.8 Using mlcp With Kerberos

You can use mlcp in local mode with Kerberos to avoid sending cleartext passwords between your mlcp client host and MarkLogic Server. You cannot use mlcp with Kerberos in distributed mode.

Before you can use Kerberos with mlcp, you must configure your MarkLogic installation to enable external security, as described in External Security in the *Security Guide*.

If external security is not already configured, you will need to perform at least the following procedures:

- Create a Kerberos external security configuration object. For details, see Creating an External Authentication Configuration Object in the *Security Guide*.
- Create a Kerberos keytab file and install it in your MarkLogic installation. For details, see Creating a Kerberos keytab File in the *Security Guide*.
- Create one or more users associated with an external name. For details, see Assigning an External Name to a User in the *Security Guide*.
- Configure your XDBC App Server to use “kerberos-ticket” authentication. For details, see Configuring an App Server for External Authentication in the *Security Guide*.

The following topics touch on additional details specific to mlcp.

- Creating Users
- Configuring an XDBC App Server for Kerberos Authentication
- Invoking mlcp

2.8.1 Creating Users

Before you can use Kerberos for authentication, you must create at least one MarkLogic user with which mlcp can use Kerberos authentication to connect to MarkLogic Server, as described in Assigning an External Name to a User in the *Security Guide*.

This user must also be assigned roles and privileges required to enable your mlcp operations.

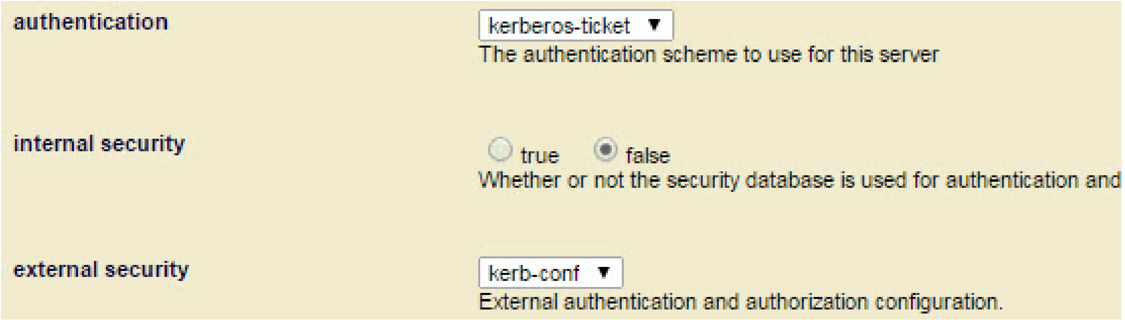
For example, if you’re using mlcp to import documents into a database, then the user must have update privileges on the target database, as well as the minimum privileges required by mlcp. For details on the minimum privileges required by mlcp, see “Security Considerations” on page 14.

2.8.2 Configuring an XDBC App Server for Kerberos Authentication

The mlcp tool communicates with MarkLogic through an XDBC App Server. Configure your XDBC App Server to use Kerberos for external security, as described in Configuring an App Server for External Authentication in the *Security Guide*.

Configure your XDBC App Server to use “kerberos-ticket” authentication.

For example, if you create a configuration named “kerb-conf”, then configure your XDBC App Server with the following values for the “authentication”, “internal security”, and “external security” configuration settings in the Admin Interface:



authentication kerberos-ticket ▼
The authentication scheme to use for this server

internal security ○ true ● false
Whether or not the security database is used for authentication and

external security kerb-conf ▼
External authentication and authorization configuration.

You can use an existing XDBC App Server or create a new one. To create a new XDBC App Server, use the Admin Interface, the Admin API, or the REST Management API. For details, see Procedures for Creating and Managing XDBC Servers in the *Administrator’s Guide*.

Configure the App Server to use “kerberos-ticket” authentication and the Kerberos external security configuration object you created following the instructions in Creating an External Authentication Configuration Object in the *Security Guide*.

> **Note:** When you install MarkLogic, an XDBC App Server and other services are available port 8000. Changing the security configuration for the App Server on port 8000 affects all the MarkLogic services available through this port, including the HTTP App Server and REST Client API instance.

2.8.3 Invoking mlcp

Once you configure your XDBC App Server and user for Kerberos external security, then you can do the following to use Kerberos authentication with mlcp:

- Use `kinit` or a similar program on your mlcp client host to create and cache a Kerberos Ticket to Get Tickets (TGT) for a principal you assigned to a MarkLogic user.
- Invoke mlcp with no `-username` and no `-password` option from the environment in which you cached the TGT.

For example, suppose you configured an XDBC App Server on port 9010 of host “ml-host” to use “kerberos-ticket” authentication. Further, suppose you associated the Kerberos principal name “kuser” with the user “mluser”. Then the following commands result in mlcp authenticating with Kerberos as user “kuser”, and importing documents into the database as “mluser”.

```
kinit kuser
...
mlcp.sh import -host ml-host -port 9010 -input_file_path src_dir
```

You do not necessarily need to run kinit every time you invoke mlcp. The cached TGT typically has a lifetime over which it is valid.

3.0 Getting Started With mlcp

This chapter walks you through a short introduction to mlcp in which you import documents into a database and then export them back out as files in the following steps:

3.1 Prepare to Run the Examples

This section leads you through creating a work area and sample data with the following file system layout:

```
gs/
  import/
    one.xml
    two.json
  export/
```

Follow this procedure to set up the example work area

1. Download and install mlcp according to the instructions in “Installation and Configuration” on page 12.

2. Ensure the mlcp `bin` directory and the `java` commands are on your path. For example, the following example command places the mlcp `bin` directory on your path if mlcp is installed in *`MLCP_INSTALL_DIR`*:

```
Linux: export PATH=${PATH}:MLCP_INSTALL_DIR/bin
Windows: set PATH=%PATH%;MLCP_INSTALL_DIR\bin
```

3. Create a directory to serve as your work area and change directories to this work area. For example:

```
mkdir gs
cd gs
```

4. Create a sub-directory to hold the sample input and output data. For example:

```
mkdir import
```

5. Create the sample input files in the `import/` directory.

   a. Use the following commands on Linux:

```
echo '<data>1</data>' > import/one.xml
echo '{"two": 2}' > import/two.json
```

   b. Use the following commands on Windows:

```
echo ^<data^>1^</data^> > import\one.xml
echo {"two":2} > import\two.json
```

3.2 Optional: Create an Options File

You can encapsulate mlcp command line options in an options file; for details, see "Options File Syntax" on page 9. An options file is convenient for re-use of commonly used options. Also, using an options file can help you avoid command line interpolation of quotes by the shell.

The examples use an options file to save MarkLogic connection related options so that you can easily re-use them across multiple commands. This section describes how to create this file.

If you prefer to pass the connection options directly on the command line instead, add `-username`, `-password`, `-host`, and possibly `-port` options to the example mlcp commands in place of `-options_file`.

Use the following procedure to create the example options file.

1. If you are not already at the top level of your work area, change directory to this location. That is, the `gs` folder created in "Prepare to Run the Examples" on page 23.

```
cd gs
```

2. Create a file named `conn.txt` with the following contents. Each line is either an option name or a value for the preceding option.

```
-username
your_username
-password
your_password
-host
localhost
-port
8000
```

3. Edit `conn.txt` and modify the values of the `-username` and `-password` options to match your environment.

4. Optionally, modify the `-host` and/or `-port` option values. The host and port must identify a MarkLogic Server App Server that supports the XDBC protocol. MarkLogic Server

comes with an App Server pre-configured on port 8000 that supports XDBC, attached to the Documents database. You can choose a different App Server.

You should now have the following file structure:

```
gs/
  conn.txt
  import/
    one.xml
    two.json
```

3.3 Load Documents

Load documents into a MarkLogic Server database using the mlcp `import` command. The examples in this section load documents from flat files into the default database associated with the App Server on port 8000 (the Documents database).

Other input options include compressed files, delimited text files, aggregate XML data, line-delimited JSON data, and Hadoop sequence files; for details, see "Importing Content Into MarkLogic Server" on page 30. You can also load document into a different database using the `-database` option.

To load a single file, specify the path to the file as the value of `-input_file_path`. For example:

```
-input_file_path import
```

When you load documents, a default URI is generated based on the type of input data. For details, see "Controlling Database URIs During Ingestion" on page 33.

We will import documents from flat files, so the default URI is the absolute pathname of the input file. For example, if your work area is `/space/gs` on Linux or `C:\gs` on Windows, then the default URI when you import documents from `gs/import` is as follows:

```
Linux: /space/gs/import/filename
Windows: /c:/gs/import/filename
```

You can use a `-output_uri_replace` option to the command line to strip off the portion of the URI that comes from the path steps before "`gs`". The option argument is of the form "*pattern,replacement_text*". For example, given the default URIs shown above, we'll add the following option to create URIs that begin with "/gs":

```
Linux: -output_uri_replace "/space,''"
Windows: -output_uri_replace "/c:,''"
```

Run the following command from the root of your work area (`gs`) to load all the files in the `import` directory. Modify the argument to `-output_uri_replace` to match your environment.

```
Linux:
  mlcp.sh import -options_file conn.txt \
      -output_uri_replace "/space,''" -input_file_path import

Windows:
  mlcp.bat import -options_file conn.txt ^
    -output_uri_replace "/c:,''" -input_file_path import
```

The output from mlcp should look similar to the following (but with a timestamp prefix on each line). "`OUTPUT_RECORDS_COMITTED: 2`" indicates mlcp loaded two files. For more details, see "Understanding mlcp Output" on page 27.

```
INFO contentpump.LocalJobRunner: Content type is set to MIXED.  The format of
  the  inserted documents will be determined by the MIME  type specification
  configured on MarkLogic Server.
INFO input.FileInputFormat: Total input paths to process : 2
INFO contentpump.LocalJobRunner:  completed 100%
INFO contentpump.LocalJobRunner: com.marklogic.mapreduce.ContentPumpStats:
INFO contentpump.LocalJobRunner: INPUT_RECORDS: 2
INFO contentpump.LocalJobRunner: OUTPUT_RECORDS: 2
INFO contentpump.LocalJobRunner: OUTPUT_RECORDS_COMMITTED: 2
INFO contentpump.LocalJobRunner: OUTPUT_RECORDS_FAILED: 0
INFO contentpump.LocalJobRunner: Total execution time: 0 sec
```

Optionally, use Query Console's Explore feature to examine the contents of the Documents database and see that the documents were created. You should see documents with the following URIs:

```
/gs/import/one.xml
/gs/import/two.json
```

You can also create documents from files in a compressed file and from other types of input archives. For details, see "Importing Content Into MarkLogic Server" on page 30.

3.4 Export Documents

Use the mlcp `export` command to export documents from a MarkLogic Server database into files on your filesystem. You can export documents to several formats, including files, compressed files, and database archives. For details, see "Exporting Content from MarkLogic Server" on page 95.

You can identify the documents to export in several ways, including by URI, by directory, by collection, and by XPath expression. This example uses a directory filter. Recall that the input documents were loaded with URIs of the form `/gs/import/`*`filename`*. Therefore we can easily extract the files by database directory using `-directory_filter /gs/import/`.

This example exports documents from the default database associated with the App Server on port 8000. Use the `-database` option to export documents from a different database.

Use the following procedure to export the documents inserted in “Load Documents” on page 25.

1. If you are not already at the top level of your work area, change directory to this location. That is, the gs folder created in “Prepare to Run the Examples” on page 23. For example:

```
cd gs
```

2. Extract the previously inserted documents into a directory named export. The export directory must not already exist.

```
Linux:
  mlcp.sh export -options_file conn.txt -output_file_path export \
    -directory_filter /gs/import/

Windows:
  mlcp.bat export -options_file conn.txt -output_file_path export ^
    -directory_filter /gs/import/
```

You should see output similar to the following, but with a timestamp prefix on each line. The “OUTPUT_RECORDS: 2” line indicates mlcp exported 2 files.

```
INFO mapreduce.MarkLogicInputFormat: Fetched 1 forest splits.
INFO mapreduce.MarkLogicInputFormat: Made 1 splits.
INFO contentpump.LocalJobRunner:  completed 100%
INFO contentpump.LocalJobRunner: com.marklogic.mapreduce.ContentPumpStats:
INFO contentpump.LocalJobRunner: INPUT_RECORDS: 2
INFO contentpump.LocalJobRunner: OUTPUT_RECORDS: 2
INFO contentpump.LocalJobRunner: Total execution time: 0 sec
```

The exported documents are in gs/export. A filesystem directory is created for each directory step in the original document URI. Therefore, you should now have the following directory structure:

```
gs/
  export/
    gs/
      import/
        one.xml
        two.json
```

3.5 Understanding mlcp Output

The output from mlcp varies depending on the operation (import, export, copy, extract), but usually looks similar to the following (with a timestamp prefix on each line).

```
INFO contentpump.LocalJobRunner: Content type is set to MIXED.  The format of
  the  inserted documents will be determined by the MIME  type specification
  configured on MarkLogic Server.
INFO input.FileInputFormat: Total input paths to process : 2
INFO contentpump.LocalJobRunner:  completed 100%
INFO contentpump.LocalJobRunner: com.marklogic.mapreduce.ContentPumpStats:
INFO contentpump.LocalJobRunner: INPUT_RECORDS: 2
INFO contentpump.LocalJobRunner: OUTPUT_RECORDS: 2
INFO contentpump.LocalJobRunner: OUTPUT_RECORDS_COMMITTED: 2
INFO contentpump.LocalJobRunner: OUTPUT_RECORDS_FAILED: 0
INFO contentpump.LocalJobRunner: Total execution time: 0 sec
```

The following table summarizes the purpose of key pieces of information reported by mlcp:

Message	Description
`Content type is set to format X.`	Import only. This indicates the type of documents mlcp will create. The default is MIXED, which means mlcp will base the type on the input file suffix. For details, see “How mlcp Determines Document Type” on page 36.
`Total input paths to process : N`	Import only. Found N candidate input sources. If this number is 0, then the pathname you supplied to `-input_file_path` does not contain any data that meets your import criteria. If you’re unable to diagnose the cause, refer to “Troubleshooting” on page 143.
`INPUT_RECORDS: N`	The number of inputs mlcp actually tried to process. For an import operation, this is the number of documents mlcp attempted to create. For an export operation, this is number of documents mlcp attempted to export. If there are errors, this number may not correspond to the actual number of documents imported, exported, copied, or extracted. This number can be larger or smaller than the total input paths. For example, if you import from a compressed file that includes directories, the directories count towards total inputs paths, but mlcp will only attempt to create documents from the file entries, so total paths will be larger than the attempted records. Similarly, if you’re loading aggregate XML files and splitting them into multiple documents, then total input paths reflects the number of aggregate files, while the attempted records reflects the number of documents created from the aggregates, so total paths is less than attempted records.

Message	Description
`OUTPUT_RECORDS: N`	On import, the number of documents (records) sent to MarkLogic for insertion into the database. This number can be smaller than INPUT_RECORDS if errors are detected on the client that cause a record to be skipped. On export, the number of output files mlcp successfully created.
`OUTPUT_RECORDS_COMMITTED: N`	Import only. The number of documents committed to the database. This number can be larger or smaller than OUTPUT_RECORDS. For example, it will be smaller if an error is detected on MarkLogic Server or larger if a server-side transformation creates multiple documents from a single input document.
`OUTPUT_RECORDS_FAILED: N`	Import only. The number of documents (records) rejected by MarkLogic Server. This number does not include failures detected by mlcp on the client.

3.6 Stopping an mclp Job Prematurely

Note that if you stop a job prematurely, some work might continue.

When you use mlcp in distributed mode, mlcp distributes its work across a Hadoop cluster. Interrupting the local mlcp client does not cause work to stop on the Hadoop cluster. In local mode, an interrupted job will shutdown gracefully as long as it can finish withint 30 seconds. If that time period expires, mlcp prints a warning.

4.0 Importing Content Into MarkLogic Server

You can use mlcp to insert content into a MarkLogic Server database from flat files, compressed ZIP and GZIP files, aggregate XML files, Hadoop sequence files, and MarkLogic Server database archives. The input data can be accessed from the native filesystem or HDFS.

For a list of import related options, see “Import Command Line Options” on page 84.

This chapter covers the following topics:

4.1 Supported Input Format Summary

Use the `-input_file_type` option to tell mlcp the format of the data in each input file (or each entry inside a compressed file). This option controls if/how mlcp converts the content into database documents.

The default input type is `documents`, which means each input file or ZIP file entry creates one database document. All other input file types represent composite input formats which can yield multiple database documents per input file.

The following table provides a quick reference of the supported input file types, along with the allowed document types for each, and whether or not they can be passed to mlcp as compressed files.

-input_file_type	Document Type	-input_compressed permitted
documents	XML, JSON, text, or binary; controlled with -document_type.	Yes
archive	As in the database: XML, JSON, text, and/or binary documents, plus metadata. The type is not under user control.	No (archives are already in compressed format)
delimited_text	XML or JSON	Yes
delimited_json	JSON	Yes
sequencefile	XML, text or binary; controlled with these options: -input_sequencefile_value_class -input_sequencefile_value_type.	No. However, the contents can be compressed when you create the sequence file. Compression is bound up with the value class you use to generate and import the file.
aggregates	XML	Yes
rdf	Serialized RDF triples, in one of several formats. For details, see Supported RDF Triple Formats in the *Semantics Developer's Guide*. RDF/JSON is not supported.	Yes
forest	As in the database: XML, JSON, text, and/or binary documents. The type is not under user control.	No

When the input file type is documents or sequencefile you must consider both the input format (-input_file_type) and the output document format (-document_type). In addition, for some input formats, input can come from either compressed or uncompressed files (-input_compressed).

The -document_type option controls the database document format when -input_file_type is documents or sequencefile. MarkLogic Server supports text, JSON, XML, and binary documents. If the document type is not explicitly set with these input file types, mlcp uses the input file suffix to determine the type. For details, see "How mlcp Determines Document Type" on page 36.

Note: You cannot use mlcp to perform document conversions. Your input data should match the stated document type. For example, you cannot convert XML input into a JSON document just by setting `-document_type json`.

To illustrate how the `-input_file_type` and `-document_type` fit together, consider a Hadoop sequence file that contains binary values. You would set the following options:

- `-input_file_type sequencefile`
- `-document_type binary`

If the sequence file contained text rather than binary values, then `-input_file_type` is unchanged, but `-document_type` becomes `text`:

- `-input_file_type sequencefile`
- `-document_type text` (or `xml`, if the values are valid XML)

4.2 Understanding Input File Path Resolution

If you do not explicitly include a URI scheme prefix such as `file:` or `hdfs:` on the input file path, mlcp uses the following rules to locate the input path:

- In local mode, mlcp defaults to the local file system (`file`).
- In distributed mode, mlcp defaults to the Hadoop default scheme, which is usually HDFS. The Hadoop default scheme is configurable through the Hadoop configuration parameter `fs.default.name`.

Note: In distributed mode, the file scheme (`file:`) refers to the local filesystem of the Hadoop cluster nodes to which the job is distributed. For example, if you perform an import in distributed mode with an input file path that uses the `file:` prefix, the input files must be reachable along that path from all nodes in your Hadoop cluster.

The following example loads files from the local filesystem directory `/space/bill/data`:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password passwd -input_file_path /space/bill/data -mode local
```

The following example loads files from the native filesystem of each host in a Hadoop cluster, assuming `/space/bill/data` is a shared network path on all hosts in the Hadoop cluster:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password passwd -input_file_path file:/space/bill/data \
    -mode distributed
```

4.3 Controlling Database URIs During Ingestion

By default, the document URIs created by mlcp during ingestion are determined by the input source. The tool supports several command line options for modifying this default behavior.

- Default Document URI Construction
- Transforming the Default URI
- Character Encoding of URIs

4.3.1 Default Document URI Construction

The default database URI assigned to ingested documents depends on the input source. Loading content from the local filesystem or HDFS can create different URIs than loading the same content from a ZIP file or archive. Command line options are available for you to modify this behavior. You can use options to generate different URIs; for details, see “Transforming the Default URI” on page 35.

The following table summarizes the default behavior with several input sources:

Input Source	Default URI	Example
documents in a native or HDFS directory	`/path/filename` Note that on Windows, the device (“c:”) becomes a path step, so `c:\path\file` becomes `/c:/path/file`.	`/space/data/bill/dream.xml` `/c:/data/bill/dream.xml`
documents in a ZIP or GZIP file	`/compressed-file-path/path/inside/zip/filename`	If the input file is `/space/data/big.zip` and it contains a directory entry `bill/`, then the document URI for `dream.xml` in that directory is: `/space/data/big.zip/bill/dream.xml`
a GZIP compressed document	`/path/filename-without-gzip-suffix`	If the input is `/space/data/big.xml.gz`, the result is `/space/data/big.xml`.
delimited text file	The value in the column used as the id. (The first column, by default).	For a record of the form “first,second,third” where Column 1 is the id: `first`

Input Source	Default URI	Example
archive or forest	The document URI from the source database.	
sequence file	The key in a key-value pair	

aggregate XML
line delimited JSON
	`/path/filename-split_start-seqnum`
	Where */path/filename* is the full path to the input file, *split_start* is the byte position from the beginning of the split, and *seqnum* begins with 1 and increments for each document created.
		For input file `/space/data/big.xml`:
		`/space/data/big.xml-0-1`
		`/space/data/big.xml-0-2`
		For input file `/space/data/big.json`:
		`/space/data/big.json-0-1`
		`/space/data/big.json-0-2`

RDF	A generated unique name	`c7f92bccb4e2bfdc-0-100.xml`

For example, the following command loads all files from the file systemdirectory `/space/bill/data` into the database attached to the App Server on port 8000. The documents inserted into the database have URIs of form `/space/bill/data/`*`filename`*.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password passwd -input_file_path /space/bill/data -mode local
```

If the `/space/bill/data` directory is zipped up into `bill.zip`, such that `bill/` is the root directory in zip file, then the following command inserts documents with URIs of the form `bill/data/`*`filename`*:

```
# Windows users, see Modifying the Example Commands for Windows
$ cd /space; zip -r bill.zip bill
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password passwd -input_file_path /space/bill.zip \
    -mode local -input_compressed true
```

When you use the `-generate_uri` option to have mlcp generate URIs for you, the generated URIs follow the same pattern as for aggregate XML and line delimited JSON:

```
/path/filename-split_start-seqnum
```

The generated URIs are unique across a single import operation, but they are not globally unique. For example, if you repeatedly import data from some file `/tmp/data.csv`, the generated URIs will be the same each time (modulo differences in the number of documents inserted by the job).

4.3.2 Transforming the Default URI

Use the following options to tailor the database URI of inserted documents:

- `-output_uri_replace` performs one or more string substitutions on the default URI.
- `-output_uri_prefix` prepends a string to the URI after substitution.
- `-output_uri_suffix` appends a string to the URI after substitution.

The `-output_uri_replace` option accepts a comma delimited list of regular expression and replacement string pairs. The string portion must be enclosed in single quotes:

```
-output_uri_replace pattern,'string',pattern,'string'
```

For details on the regular expression language supported by `-output_uri_replace`, see “Regular Expression Syntax” on page 9.

> **Note:** These options are applied after the default URI is constructed and encoded, so if the option values contain characters not allowed in a URI, you must encode them yourself. See “Character Encoding of URIs” on page 35.

The following example loads documents from the filesystem directory `/space/bill/data`. The default output URIs would be of the form `/space/bill/data/`*`filename`*. The example uses `-output_uri_replace` to replace “bill/data” with “will” and strip off “/space/”, and then adds a “/plays” prefix using `-output_uri_prefix`. The end result is output URIs of the form `/plays/will/`*`filename`*.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password passwd -input_file_path /space/bill/data -mode local \
    -output_uri_replace "/space,'',/bill/data/,'/will/'" \
    -output_uri_prefix /plays
```

Character Encoding of URIs

If a URI constructed by mlcp contains special characters that are not allowed in URIs, mlcp automatically encodes them. This applies to the special characters “ ” (space), “%”, “?” or “#”. For example, “foo bar.xml” becomes “foo%20bar.xml”.

If you supply a URI or URI component, you are responsible for ensuring the result is a legitimate URI. No automatic encoding takes place. This applies to `-output_uri_replace`, `-output_uri_prefix`, and `-output_uri_suffix`. The changes implied by these options are applied after mlcp encodes the default URI.

When mlcp exports documents from the database to the file system (or HDFS) such that the output directory and/or file names are derived from the document URI, the special symbols are decoded. That is, “foo%bar.xml” becomes “foo bar.xml” when exported. For details, see “How URI Decoding Affects Output File Names” on page 99.

4.4 How mlcp Determines Document Type

The document type determines what kind of database document mlcp inserts from input content: Text, XML, JSON, or binary. Document type is determined in the following ways:

- Document type can be inherent in the input file type. For example, `aggregates` and `rdf` input files always insert XML documents. For details, see "Supported Input Format Summary" on page 30.
- You can specify a document type explicitly with `-document_type`. For example, to load documents as XML, use `-input_file_type documents -document_type xml`. You cannot set an explicit type for all input file types.
- Mlcp can determine document type dynamically from the output document URI and the MarkLogic Server MIME type mappings when you use `-input_file_type documents -document_type mixed`.

If you set `-document_type` to an explicit type such as `-document_type json`, then mlcp inserts all documents as that type.

If you use `-document_type mixed`, then mlcp determines the document type from the output URI suffix and the MIME type mapping configured into MarkLogic Server. Mixed is the default behavior for `-input_file_type documents`.

Note: You can only use `-document_type mixed` when the input file type is `documents`.

Note: If an unrecognized or unmapped file extension is encountered when loading mixed documents, mlcp creates a binary document.

The following table contains examples of applying the default MIME type mappings to output URIs with various file extensions, an unknown extension, and no extension. The default mapping includes many additional suffixes. You can examine and create MIME type mappings under the Mimetypes section of the Admin Interface. For more information, see Implicitly Setting the Format Based on the MIME Type in the *Loading Content Into MarkLogic Server Guide*.

URI	Document Type
`/path/doc.xml`	XML
`/path/doc.json`	JSON

URI	Document Type
`/path/doc.jpg`	binary
`/path/doc.txt`	text
`/path/doc.unknown`	binary
`/path/doc-nosuffix`	binary

The MIME type mapping is applied to the final output URI. That is, the URI that results from applying the URI transformation options described in “Controlling Database URIs During Ingestion” on page 33. The following table contains examples of how URI transformations can affect the output document type in `mixed` mode, assuming the default MIME type mappings.

Input Filename	URI Options	Output URI	Doc Type
`/path/doc.1`	None	`/path/file.1`	binary
`/path/doc.1`	Add a `.xml` suffix: `-output_uri_suffix ".xml"`	`/path/file.xml`	XML
`/path/doc.1`	Replace the unmapped suffix with .txt: `-output_uri_replace "\.\d+,'.txt'"`	`/path/file.txt`	text

Document type determination is completed prior to invoking server side transformations. If you change the document type in a transformation function, you are responsible for changing the output document to match. For details, see “Transforming Content During Ingestion” on page 66.

4.5 Loading Documents from a Directory

This section discusses importing documents stored as flat files on the native filesystem or HDFS. The following topics are covered:

- Loading a Single File
- Loading All the Files in a Directory
- Filtering Documents Loaded From a Directory

4.5.1 Loading a Single File

Use the following procedure to load all the files in a native or HDFS directory and its sub-directories. To load selected files, see “Filtering Documents Loaded From a Directory” on page 39.

1. Set `-input_file_path` to the path to the input file.

2. Set `-input_file_type` if your input files are not documents. For example, if loading from delimited text files, sequence files, aggregate XML files, RDF triples files, or database archives.

3. Set `-document_type` if `-input_file_type` is not `documents` and the content type cannot be accurately deduced from the file suffixes as described in “How mlcp Determines Document Type” on page 36.

4. Set `-mode`:

   - If Hadoop is available and you want to distribute the workload across a Hadoop cluster, set `-mode` to `distributed`.
   - If Hadoop is not installed or you want mlcp to perform the work locally, set `-mode` to `local`. (This is the default mode unless you set the `HADOOP_CONF_DIR` variable).

Note: If you are loading from the native filesystem in distributed mode or from HDFS in local mode, you might need to qualify the input file path with a URI scheme of `file:` or `hdfs:`. See “Understanding Input File Path Resolution” on page 32.

By default, the imported document has a database URI based on the input file path. For details, see “Controlling Database URIs During Ingestion” on page 33.

The following example command loads a single XML file:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password password -input_file_path /space/bill/data/hamlet.xml
```

4.5.2 Loading All the Files in a Directory

Use the following procedure to load all the files in a native or HDFS directory and its sub-directories. To load selected files, see “Filtering Documents Loaded From a Directory” on page 39.

1. Set `-input_file_path` to the input directory.

2. Set `-input_file_type` if your input files are not documents. For example, if loading from delimited text files, sequence files, aggregate XML files, or database archives.

3. Set `-document_type` if `-input_file_type` is not `documents` and the content type cannot be accurately deduced from the file suffixes as described in “How mlcp Determines Document Type” on page 36.

4. Set `-mode`:

   - If Hadoop is available and you want to distribute the workload across a Hadoop cluster, set `-mode` to `distributed`.
   - If Hadoop is not installed or you want mlcp to perform the work locally, set `-mode` to `local`. (This is the default mode unless you set the `HADOOP_CONF_DIR` variable).

   Note: If you are loading from the native filesystem in distributed mode or from HDFS in local mode, you might need to qualify the input file path with a URI scheme of `file:` or `hdfs:`. See “Understanding Input File Path Resolution” on page 32.

By default, the imported documents have database URIs based on the input file path. For details, see “Controlling Database URIs During Ingestion” on page 33.

The following example command loads all the files in `/space/bill/data`:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password password -input_file_path /space/bill/data
```

4.5.3 Filtering Documents Loaded From a Directory

If `-input_file_path` names a directory, mlcp loads all the documents in the input directory and subdirectories by default. Use the `-input_file_pattern` option to filter the loaded documents based on a regular expression.

Note: Input document filtering is handled differently for `-input_file_type forest`. For details, see “Filtering Forest Contents” on page 106.

For example, the following command loads only files with a “.xml” suffix from the directory `/space/bill/data`:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password password -input_file_path /space/bill/data \
    -mode local -input_file_pattern '.*\.xml'
```

The mlcp tool uses Java regular expression syntax. For details, see “Regular Expression Syntax” on page 9.

4.6 Loading Documents From Compressed Files

You can load content from one or more compressed files. Filtering of compressed file content is not supported; mlcp loads all documents in a compressed file.

Follow this procedure to load content from one or more ZIP or GZIP compressed files.

1. Set `-input_file_path`:
   - To load from a single file, set `-input_file_path` to the path to the compressed file.
   - To load from multiple files, set `-input_file_path` to a directory containing the compressed files.
2. If the content type cannot be accurately deduced from suffixes of the files inside the compressed file as described in “How mlcp Determines Document Type” on page 36, set `-document_type` appropriately.
3. Set `-input_compressed` to `true`.
4. If the compressed file suffix is not “.zip” or “.gzip”, specify the compressed file format by setting `-input_compression_codec` to `zip` or `gzip`.

If you set `-document_type` to anything but `mixed`, then the contents of the compressed file must be homogeneous. For example, all XML, all JSON, or all binary.

The following example command loads binary documents from the compressed file `/space/images.zip` on the local filesystem.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password password -mode local -document_type binary \
    -input_file_path /space/images.zip -input_compressed
```

The following example loads all the files in the compressed file `/space/example.jar`, using `-input_compression_codec` to tell mlcp the compression format because of the “.jar” suffix:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password passwd -mode local -input_file_path /space/example.jar \
    -input_compressed true -input_compression_codec zip
```

If `-input_file_path` is a directory, mlcp loads contents from all compressed files in the input directory, recursing through subdirectories. The input directory must not contain other kinds of files.

By default, the URI prefix on documents loaded from a compressed file includes the full path to the input compressed file and mirrors the directory hierarchy inside the compressed file. For example, if a ZIP file `/space/shakespeare.zip` contains `bill/data/dream.xml` then the ingested document URI is `/space/shakespeare.zip/bill/data/dream.xml`. To override this behavior, see “Controlling Database URIs During Ingestion” on page 33.

4.7 Loading Content and Metadata From an Archive

Follow this procedure to import content and metadata from a database archive created by the mlcp export command. A database archive is stored in one or more compressed files that contain documents and metadata.

1. Set -input_file_path:
    - To load a single archive file, set -input_file_path to that file.
    - To load multiple archive files, set -input_file_path to a directory containing the compressed archive files.
2. Set -document_type to mixed, or leave it unset since mixed is the default setting.
3. Set -input_compressed to true.
4. Set -input_file_type to archive.
5. If the input archive was created without any metadata, set -archive_metadata_optional to true. If this is not set, an exception is thrown if the archive contains no metadata.
6. If you want to exclude some or all of the document metadata in the archive:
    - Set -copy_collections to false to exclude document collections metadata.
    - Set -copy_permissions to false to exclude document permissions metadata.
    - Set -copy_properties to false to exclude document properties.
    - Set -copy_quality to false to exclude document quality metadata.
    - Set -copy_metadata to false to exclude key-value metadata.

An archive is assumed to contain metadata. However, it is possible to create archives without metadata by setting all the metadata copying options (-copy_collections, -copy_permissions, etc.) to false during export. If an archive does not contain metadata, you must set -archive_metadata_optional to tell mlcp to proceed in the absence of metadata.

> **Note:** When you import properties from an archive, you should disable the "maintain last modified" configuration option on the destination database during the import. Otherwise, you can get an XDMP-SPECIALPROP error if the import operation tries to update the last modified property. To disable this setting, use the Admin Interface or the library function admin:set-maintain-last-modified.

The following example command loads the database archive in /space/archive_dir:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password password -mode local -input_file_type archive \
    -input_file_path /space/archive_dir
```

4.8 Splitting Large XML Files Into Multiple Documents

Very large XML files often contain *aggregate* data that can be disaggregated by splitting it into multiple smaller documents rooted at a recurring element. Disaggregating large XML files consumes fewer resources during loading and improves performance when searching and retrieving content. For aggregate JSON handling, see "Creating Documents from Line-Delimited JSON Files" on page 48.

The following mlcp options support creating multiple documents from aggregate data:

- `-aggregate_record_element`
- `-uri_id`
- `-aggregate_record_namespace`

You can disaggregate XML when loading from either flat or compressed files. For more information about working with compressed files, see "Loading Documents From Compressed Files" on page 39.

Follow this procedure to create documents from aggregate XML input:

1. Set `-input_file_path`:
   - To load from a single file, set `-input_file_path` to the path to the aggregate XML file.
   - To load from multiple files, set `-input_file_path` to a directory containing the aggregate files. The directory must not contain other kinds of files.
2. If you are loading from a compressed file, set `-input_compressed`.
3. Set `-input_file_type` to `aggregates`.
4. Set `-aggregate_record_element` to the element QName of the node to use as the root for all inserted documents. See the example below. The default is the first child element under the root element.

   Note: The element QName should appear at only one level. You cannot specify the element name using a path, so disaggregation occurs everywhere that name is found.

5. Optionally, override the default document URI by setting `-uri_id` to the name of the element from which to derive the document URI.
6. If the aggregate record element is in a namespace, set `-aggregate_record_namespace` to the input namespace.

The default URI is *`hashcode-seqnum`* in local mode and *`taskid-seqnum`* in distributed mode. If there are multiple matching elements, the first match is used.

If your aggregate URI id's are not unique, you can overwrite one document in your input set with another. Importing documents with non-unique URI id's from multiple threads can also cause deadlocks.

The example below uses the following input data:

```
$ cat > example.xml
<?xml version="1.0" encoding="UTF-8"?>
<people>
  <person>
    <first>George</first>
    <last>Washington</last>
  </person>
  <person>
    <first>Betsy</first>
    <last>Ross</last>
  </person>
</people>
```

The following command breaks the input data into a document for each `<person>` element. The `-uri_id` and other URI options give the inserted documents meaningful names. The command creates URIs of the form "/people/lastname.xml" by using the `<last/>` element as the aggregate URI id, along with an output prefix and suffix:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password password -mode local -input_file_path example.xml \
    -input_file_type aggregates -aggregate_record_element person \
    -uri_id last -output_uri_prefix /people/ \
    -output_uri_suffix .xml
```

The command creates two documents: `/people/Washington.xml` and `/people/Ross.xml`. For example, `/people/Washington.xml` contains:

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <first>George</first>
    <last>Washington</last>
</person>
```

If the input data is in a namespace, set `-aggregate_record_namespace` to that namespace. For example, if the input data is modified to include a namespace:

```
$ cat > example.xml
<?xml version="1.0" encoding="UTF-8"?>
<people xmlns="http://marklogic.com/examples">...</people>
```

Then mlcp ingests no documents unless you set `-aggregate_record_namespace`. Setting the namespace creates two documents in the namespace "http://marklogic.com/examples". For example, after running the following command:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password password -mode local -input_file_path example.xml \
    -input_file_type aggregates -aggregate_record_element person \
    -uri_id last -output_uri_prefix /people/ \
    -output_uri_suffix .xml \
    -aggregate_record_namespace "http://marklogic.com/examples"
```

The document with URI “/people/Washington.xml” contains :

```
<?xml version="1.0" encoding="UTF-8"?>
<person xmlns="http://marklogic.com/examples">
    <first>George</first>
    <last>Washington</last>
</person>
```

4.9 Creating Documents from Delimited Text Files

Use the `delimited_text` input file type to import content from a delimited text file and create an XML or JSON document corresponding to each line. For line-delimited JSON data, see “Creating Documents from Line-Delimited JSON Files” on page 48.

The following options are commonly used in the generation of documents from delimited text files:

- `-input_file_type delimited_text`
- `-document_type xml` or `-document_type json`
- `-delimiter`
- `-uri_id`
- `-delimited_root_name` (XML output only)
- `-data_type` (JSON output only)

The use of these and other supporting options is covered in the following topics:

- Example: Generating Documents From a CSV File
- Expected Input Format
- Customizing XML Output
- Controlling Data Type in JSON Output
- Controlling the Output Document URI
- Specifying the Field Delimiter
- Optimizing Ingestion of Large Files

4.9.1 Example: Generating Documents From a CSV File

When you import content from delimited text files, mlcp creates an XML or JSON document for each line of input after the initial header line.

The default document type is XML. To create JSON documents, use `-document_type json`.

When creating XML documents, each document has a root node of `<root>` and child elements with names corresponding to each column title. You can override the default root element name using the `-delimited_root_name` option; for details, see “Customizing XML Output” on page 46.

When creating JSON documents, each document is rooted at an unnamed object containing JSON properties with names corresponding to each column title. By default, the values for JSON are always strings. Use `-data_type` to override this behavior; for details, see “Controlling Data Type in JSON Output” on page 46.

For example, if you have the following data and mlcp command:

```
# Windows users, see Modifying the Example Commands for Windows
$ cat example.csv
first,last
george,washington
betsy,ross

$ mlcp.sh ... -mode local -input_file_path /space/mlcp/data \
    -input_file_type delimited_text ...
```

Then mlcp creates the XML output shown in the table below. To generate the JSON output, add `-document_type json` to the mlcp command line.

XML Output	JSON Output

`<root>`
`  <first>george</first>`
`  <last>washington</last>`
`</root>`
 	
`<root>`
`  <first>betsy</first>`
`  <last>ross</last>`
`</root>`
	`{`
	`  "first": "george",`
	`  "last": "washington"`
	`}`
	
	`{`
	`  "first": "betsy",`
	`  "last": "ross"`
	`}`

4.9.2 Expected Input Format

A delimited text input file must have the following characteristics:

- The first line in the input file contains "column" names that are used to create the XML element or JSON property names of each document created from the file.
- The same delimiter is used to separate each value, as well as the column names. The default separator is a comma; use `-delimiter` to override it; for details, see "Specifying the Field Delimiter" on page 48.
- Every line has the same number of fields (values). Empty fields are represented as two delimiters in a row, such as "a,b,,d".

For example, the following data meets the input format requirements:

```
first,last
george,washington
betsy,ross
```

This data produces documents with XML elements or JSON properties named "first" and "last".

4.9.3 Customizing XML Output

When creating XML documents, each document has a root node of `<root>` and child elements with names corresponding to each column title. You can override the default root element name using the `-delimited_root_name` option. You can use the `-namespace` option to specify a root namespace.

The following example produces documents with root element `<person>` in the namespace `http://my.namespace`.

```
$ mlcp.sh ... -mode local -input_file_path /space/mlcp/data \
    -input_file_type delimited_text -namespace http://my.namespace \
    -delimited_root_name person
...
<person xmlns="http://my.namespace">
  <first>george</first>
  <last>washington</last>
</person>
...
```

4.9.4 Controlling Data Type in JSON Output

When creating JSON documents, the default value type is string. You can use the `-data_type` option to specify explicit data types for some or all columns. The options accepts comma-separated list of *`columnName`*,*`typeName`* pairs, where the *`typeName`* can be one of `number`, `boolean`, or `string`.

For example, if you have an input file called "catalog.csv" that looks like the following:

```
id, price, in-stock
12345, 8.99, true
67890, 2.00,false
```

Then the default output documents look similar to the following. Notice that all the property values are strings.

```
{ "id": "12345",
  "price": "8.99",
  "in-stock: "true"
}
```

The following example command uses the `-data_type` option to make the "price" property a number value and the "in-stock" property a boolean value. Since the "id" field is not specified in the `-data_type` option, it remains a string.

```
$ mlcp.sh ... -mode local -input_file_path catalog.csv \
    -input_file_type delimited_text -document_type json \
    -data_type "price,number,in-stock,boolean"
...
{ "id": "12345",
  "price": 8.99,
  "in-stock: true
}
```

4.9.5 Controlling the Output Document URI

By default, the document URIs use the value in the first column. For example, if your input data looks like the following:

```
first,last
george,washington
betsy,ross
```

Then importing this data with no URI related options creates two documents with name corresponding to the "first" value. The URI will be "george" and "betsy".

Use `-uri_id` to choose a different column or `-generate_uri` to have MarkLogic Server automatically generate a unique URI for each document. For example, the following command creates the documents "washington" and "ross":

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh ... -mode local -input_file_path /space/mlcp/data \
    -input_file_type delimited_text -uri_id last
```

Note that URIs generated with `-generate_uri` are only guaranteed to be unique across your import operation. For details, see "Default Document URI Construction" on page 33.

You can further tailor the URIs using `-output_uri_prefix` and `-output_uri_suffix`. These options apply even when you use `-generate_uri`. For details, see "Controlling Database URIs During Ingestion" on page 33.

If your URI id's are not unique, you can overwrite one document in your input set with another. Importing documents with non-unique URI id's from multiple threads can also cause deadlocks.

4.9.6 Specifying the Field Delimiter

The default delimiter between fields is a comma (","). You can override this using the `-delimiter` option. If the delimiter is a character that is difficult to specify on the command line, specify the delimiter in an options file instead. For details, see "Options File Syntax" on page 9.

For example, the Linux bash shell parser makes it difficult to specify a tab delimiter on the command line, so you can put the options in a file instead. In the example options file below, the string literal after `-delimiter` should contain a tab character.

```
$ cat delim.opt
-input_file_type
delimited_text
-delimiter
"tab"

$ mlcp.sh import ... -mode local -input_file_path /space/mlcp/data \
    -options_file delim.opt
```

4.9.7 Optimizing Ingestion of Large Files

If your delimited text files are very large, consider using the `-split_input` option. When this option is true, mlcp attempts to break each input file into multiple splits, enabling more documents to be loaded in parallel. For details, see "Improving Throughput with -split_input" on page 65.

4.10 Creating Documents from Line-Delimited JSON Files

Use the `delimited_json` input file type to import content from a line-delimited JSON file and create a JSON document corresponding to each line.

This section covers the following topics:

- Line-Delimited JSON Overview
- Controlling the Output Document URI

To create JSON documents from delimited text files such as CSV files, see "Creating Documents from Delimited Text Files" on page 44. For aggregate XML input, see "Splitting Large XML Files Into Multiple Documents" on page 42.

4.10.1 Line-Delimited JSON Overview

A line-delimited JSON file is a type of aggregate file where each line is a self-contained piece of JSON data, such as an object or array.

Usually, each line of input has similar structure, such as the following:

```
{"id": "12345","price":8.99, "in-stock": true}
{"id": "67890","price":2.00, "in-stock": false}
```

However, the JSON data on each line is independent of the other lines, so the lines do not have to contain JSON data of the same “shape”. For example, the following is a valid input file:

```
{"first": "george", "last": "washington"}
{"id": 12345, "price": 8.99, "in-stock": true}
```

Given the input shown below, the following command creates 2 JSON documents. Each document contains the data from a single line of input.

```
$ cat example.json
{"id": "12345","price":8.99, "in-stock": true}
{"id": "67890","price":2.00, "in-stock": false}

# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password password -mode local -input_file_path example.json \
    -input_file_type delimited_json
```

The example command creates documents whose contents precisely mirror each of input:

```
{"id": "12345","price":8.99, "in-stock": true}

{"id": "67890","price":2.00, "in-stock": false}
```

4.10.2 Controlling the Output Document URI

The default document URI is generated from the input file name, the split number, and a sequence number within the split, as described in “Default Document URI Construction” on page 33. For example, if the input file absolute path is `/space/data/example.json`, then the default output document URIs have the following form:

```
/space/data/example.json-0-1
/space/data/example.json-0-2
...
```

You can base the URI on values in the content instead by using the `-uri_id` option to specify the name of a property found in the data. You can further tailor the URIs using `-output_uri_prefix` and `-output_uri_suffix`. For details, see “Controlling Database URIs During Ingestion” on page 33.

For example, the following command uses the value in the “id” field as the base of the URI and uses `-output_uri_suffix` to add a “.json” suffix to the URIs:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh ... -mode local -input_file_path /space/data/example.json \
    -input_file_type delimited_json
    -uri_id id -output_uri_suffix ".json"
```

Given these options, an input line of the form shown below produces a document with the URI “12345.json” instead of “/space/data/example.json-0-1”.

```
{"id": "12345","price":8.99, "in-stock": true}
```

If the property name specified with -uri_id is not unique in your data, mlcp will use the first occurrence found in a breadth first search. The value of the specified property should be a valid number or string.

If you use `-uri_id`, any records (lines) that do not contain the named property are skipped. If the property is found but the value is null or not a number or string, the record is skipped.

4.11 Creating Documents from Hadoop Sequence Files

A Hadoop sequence file is a flat binary file of key-value pairs. You can use mlcp to create a document from each key-value pair. The only supported value types are `Text` and `BytesWritable`. This section covers the following topics:

- Basic Steps
- Implementing the Key and Value Interfaces
- Deploying your Key and Value Implementation
- Loading Documents From Your Sequence Files
- Running the SequenceFile Example

4.11.1 Basic Steps

You must implement a Hadoop `SequenceFile` reader and writer that also implements 2 special mlcp interfaces. To learn more about Apache Hadoop `SequenceFile`, see http://wiki.apache.org/hadoop/SequenceFile/.

1. Implement `com.marklogic.contentpump.SequenceFileKey` and `com.marklogic.contentpump.SequenceFileValue`.

2. Generate one or more sequence files using your classes.

3. Deploy your classes into *`mlcp_install_dir`*`/lib`.

4. Use the mlcp import command to create documents from your sequence files.

The source distribution of mlcp, available from http://developer.marklogic.com, includes an example in `com.marklogic.contentpump.examples`.

4.11.2 Implementing the Key and Value Interfaces

You must read and write your sequence files using classes that implement `com.marklogic.contentpump.SequenceFileKey` and `com.marklogic.contentpump.SequenceFileValue`. These interfaces are included in the mlcp jar file:

```
mlcp_install_dir/lib/mlcp-version.jar
```

Where *version* is your mlcp version. For example, if you install mlcp version 1.3 to `/opt/mlcp-1.3`, then the jar file is:

```
/opt/mlcp-1.3/lib/mlcp-1.3.jar
```

Source and an example implementation are available in the mlcp source distribution on developer.marklogic.com.

Your key class must implement the following interface:

```
package com.marklogic.contentpump;

import com.marklogic.mapreduce.DocumentURI;

public interface SequenceFileKey {
    DocumentURI getDocumentURI();
}
```

Your value class must implement the following interface:

```
package com.marklogic.contentpump;

public interface SequenceFileValue<T> {
    T getValue();
}
```

For an example, see `com.marklogic.contentpump.example.SimpleSequenceFileKey` and `com.marklogic.contentpump.example.SimpleSequenceFileValue`.

These interfaces depend on Hadoop and the MarkLogic Connector for Hadoop. The connector library is included in the mlcp distribution as:

```
mlcp_install_dir/lib/marklogic-mapreduceN-version.jar
```

where *N* is the Hadoop major version and *version* is the connector version. The Hadoop major version will correspond to the Hadoop major version of your mlcp distribution. For example, if you install the Hadoop v2 compatible version of mlcp, then the connector jar file name might be:

```
marklogic-mapreduce2-2.1.jar
```

For details, see the *MarkLogic Connector for Hadoop Developer's Guide* and the *MarkLogic Hadoop MapReduce Connector API*.

You must implement a sequence file creator. You can find an example in `com.marklogic.contentpump.examples.SimpleSequenceFileCreator`.

When compiling your classes, include the following on the Java class path:

- `*mlcp_install_dir*/lib/mlcp-*version*.jar`
- `*mlcp_install_dir*/lib/marklogic-mapreduce*N*-*version*.jar`
- `*mlcp_install_dir*/lib/hadoop-common-*version*.jar`

For example:

```
$ javac -cp
$MLCP_DIR/lib/mlcp-1.3.jar:$MLCP_DIR/lib/marklogic-mapreduce2-2.1.jar:
$MLCP_DIR/lib/hadoop-mapreduce-client-core-2.6.0.jar \
MyKey.java MyValue.java

$ jar -cf myseqfile.jar *.class
```

4.11.3 Deploying your Key and Value Implementation

Once you compile your `SequenceFileKey` and `SequenceFileValue` implementations into a JAR file, copy your JAR file and any dependent libraries into the mlcp `lib/` directory so that mlcp can find your classes at runtime. For example:

```
$ cp myseqfile.jar /space/mlcp-1.3/lib
```

4.11.4 Loading Documents From Your Sequence Files

Once you have created one or more sequence files using your implementation, you can create a document from each key-value pair using the following procedure:

1. Set `-input_file_path`:
    - To load from a single file, set `-input_file_path` to the path to the file.
    - To load from multiple files, set `-input_file_path` to a directory containing the sequence files.
2. Set `-sequencefile_key_class` to the name of your `SequenceFileKey` implementation.
3. Set `-sequencefile_value_class` to the name of your `SequeneFileValue` implementation.
4. Set `-sequencefile_value_type` to either `Text` or `BytesWritable`, depending on the contents of your sequence files.
5. Set `-input_file_type` to `sequencefile`.

By default, the key in each key-value pair is used as the document URI. You can further tailor the URI using command line options, as described in “Controlling Database URIs During Ingestion” on page 33.

For an example, see “Running the SequenceFile Example” on page 53.

4.11.5 Running the SequenceFile Example

This section walks you through creating a sequence file and loading its contents as documents.

Create an input text file from which to create a sequence file. The file should contain pairs of lines where the first line is a URI that acts as the key, and the second line is the value. For example:

```
$ cat > seq_input.txt
/doc/foo.xml
<foo/>
/doc/bar.xml
<bar/>
```

To use the example classes provided with mlcp, put the following libraries on your Java classpath:

- *mlcp_install_dir*/lib/mlcp-*version*.jar
- *mlcp_install_dir*/lib/hadoop-mapreduce-client-core-*version*.jar
- *mlcp_install_dir*/lib/commons-logging-1.1.3.jar
- *mlcp_install_dir*/lib/marklogic-mapreduce*N*-*version*.jar

For example:

- *mlcp_install_dir*/lib/mlcp-1.3.jar
- *mlcp_install_dir*/lib/hadoop-mapreduce-client-core-2.6.0.jar
- *mlcp_install_dir*/lib/commons-logging-1.1.3.jar
- *mlcp_install_dir*/lib/marklogic-mapreduce2-2.1.jar

Generate a sequence file from your test data using `com.marklogic.contentpump.examples.SimpleSequenceFileCreator`. The first argument to program is the output sequence file name. The second argument is the input data file name. The following command generates `seq_output` from `seq_input.txt`.

```
$ java com.marklogic.contentpump.examples.SimpleSequenceFileCreator
seq_output seq_input.txt
```

Load the contents of the sequence file into MarkLogic Server:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -username user -password password -host localhost \
    -port 8000 -input_file_path seq_output -mode local \
    -input_file_type sequencefile -sequencefile_key_class \
    com.marklogic.contentpump.examples.SimpleSequenceFileKey \
    -sequencefile_value_class \
    com.marklogic.contentpump.examples.SimpleSequenceFileValue \
    -sequencefile_value_type Text -document_type xml
```

Two documents are created in the database with URIs /doc/foo.xml and /doc/bar.xml.

4.12 Loading Triples

This section provides a brief overview of loading semantic data into MarkLogic Server. For more details, see the *Semantics Developer's Guide*. The following topics are covered in this section:

- Basics of Triple Loading
- Graph Selection When Loading Quads
- Graph Selection for Other Triple Types

4.12.1 Basics of Triple Loading

To load semantic triples, use -input_file_type rdf and follow the instructions for loading a single file, all files in a directory, or a compressed file. For example, the following command loads triples files from the directory /my/data.

```
    # Windows users, see Modifying the Example Commands for Windows
    $ mlcp.sh import -username user -password password -host localhost \
        -port 8000 -input_file_path /my/data -mode local \
        -input_file_type rdf
```

You can use mlcp to load triples files in several formats, including RDF/XML, Turtle, and N-Quads. For a full list of supported formats, see Supported RDF Triple Formats in *Semantics Developer's Guide*.

Note: Each time you load triples from a file, mlcp inserts new documents into the database. That is, multiple loads of the same input inserts new triples each time, rather than overwriting. Only the XQuery and REST API allow you replace triples.

Load triples data embedded within other content according to the instructions for the enclosing input file type, rather than with -input_file_type rdf. For example, if you have an XML input document that happens to have some triples embedded in it, load the document using -input_file_type documents.

You cannot combine loading triples files with other input file types.

If you do not include any graph selection options in your mlcp command, Quads are loaded into the graph specified in the data. Quads with no explicit graph specification and other kinds of triple data are loaded into the default graph. You can change this behavior with options. For details, see "Graph Selection When Loading Quads" on page 55 or "Graph Selection for Other Triple Types" on page 57.

For details, see Loading Triples with mlcp in *Semantics Developer's Guide*.

4.12.2 Graph Selection When Loading Quads

When loading quads, you can use the following command line options to control the graph into which your quads are loaded:

- `-output_graph`
- `-output_override_graph`
- `-output_collections`

You can use `-output_collections` by itself or with the other two options. You cannot use `-output_graph` and `-output_override_graph` together.

If your semantic data is not in a quad format like N-Quads, see “Graph Selection for Other Triple Types” on page 57.

Quads interact with these options differently than other triple formats because quads can include a graph IRI in each quad. The following table summarizes the affect of various option combinations when importing quads with mlcp:

Graph Options	Description
none	For quads that contain an explicit graph IRI, load the triple into that graph. For quads with no explicit graph IRI, load the triple into the default graph. The default graph URI is `http://marklogic.com/semantics#default-graph`.
`-output_graph`	For quads that contain an explicit graph IRI, load the triple into that graph. For quads with no explicit graph IRI, load the triple into the graph specified by -output_graph.
`-output_override_graph`	Load all triples into the graph specified by `-output_override_graph`. This graph overrides any graph IRIs contained in the quads.
`-output_collections`	Similar to `-output_override_graph`, but you can specifiy multiple collections. Load triples into the graph specified as the first (or only) collection; also add triples to any additional collections on the list. This overrides any graph IRIs contained in the quads.

Graph Options	Description
-output_graph with -output_collections	For quads that contain an explicit graph IRI, load the triple into that graph. For quads with no explicit graph IRI, load the triple into the graph specified by -output_graph. Also add triples to the specified collections.
-output_override_graph with -output_collection	Load all triples into the graph specified by -output_override_graph. This graph overrides any graph IRIs contained in the quads. Also add triples to the specified collections.

For more details, see Loading Triples with mlcp in the *Semantics Developer's Guide*.

For example, suppose you load the following N-Quad data with mlcp. There are 3 quads in the data set. The first and last quad include a graph IRI, the second quad does not.

```
<http://one.example/subject1> <http://one.example/predicate1>
    <http://one.example/object1> <http://example.org/graph3> .
_:subject1 <http://an.example/predicate1> "object1"  .
_:subject2 <http://an.example/predicate2> "object2"
    <http://example.org/graph5> .
```

If you use a command similar to the following load the data:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -username user -password password -host localhost \
    -port 8000 -input_file_path /my/data.nq -mode local \
    -input_file_type rdf
```

Then the table below illustrates how the various graph related options affect how the triples are loaded into the database:

Graph Options	Result
none	Graphs: http://example.org/graph3 http://marklogic.com/semantics#default-graph http://example.org/graph5
-output_graph /my/graph	Graphs: http://example.org/graph3 /my/graph http://example.org/graph5

Graph Options	Result
`-output_override_graph /my/graph`	Graphs: `/my/graph` for all triples
`-output_collections "aa,bb,cc"`	Graphs: `aa` for all triples All triples also added to collections `bb` and `cc`

`-output_graph /my/graph`
`-output_collections "bb,cc"`
	Graphs:
	`http://example.org/graph3`
	`/my/graph`
	`http://example.org/graph5`
	All triples also added to collections `bb` and `cc`

`-output_override_graph /my/graph`
`-output_collections "bb,cc"`
	Graphs:
	`/my/graph` for all triples
	All triples also added to collections `bb` and `cc`

4.12.3 Graph Selection for Other Triple Types

When loading triples (rather than quads), you can use the following command line options to control the graph into which your triples are loaded:

- `-output_graph`
- `-output_collections`

The following table summarizes the affect of various option combinations when importing triples with mlcp. For quads, see "Graph Selection When Loading Quads" on page 55.

Graph Options	Description
none	Load triples into the default graph (`http://marklogic.com/semantics#default-graph`).
`-output_graph`	Load triples into the specified graph.
`-output_collections`	Load triples into the graph specified as the first (or only) collection; also add triples to any additional collections on the list.
`-output_graph` with `-output_collections`	Load triples into the graph specified by `-output_graph` and also add them to the specified collections.

For more details, see Loading Triples with mlcp in the *Semantics Developer's Guide*.

For example, if you use a command similar to the following load triples data:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -username user -password password -host localhost \
    -port 8000 -input_file_path /my/data.nt -mode local \
    -input_file_type rdf
```

Then the table below illustrates how the various graph related options affect how the triples are loaded into the database:

Graph Options	Result
none	Graph: `http://marklogic.com/semantics#default-graph`
`-output_graph /my/graph`	Graph: `/my/graph`
`-output_collections "aa,bb,cc"`	Graph: `aa` All triples also added to collections `bb` and `cc`

`-output_graph /my/graph`
`-output_collections "bb,cc"`
	Graph: `/my/graph`
	All triples also added to collections `bb` and `cc`

4.13 Loading Documents from a Forest With Direct Access

Direct Access enables you to extract documents directly from an offline or read-only forest without using MarkLogic Server instance for input. Direct Access is primarily intended for accessing archived data that is part of a tiered storage deployment.

For details, see “Importing Documents from a Forest into a Database” on page 139.

4.14 Performance Considerations for Loading Documents

MarkLogic Content Pump comes configured with defaults that should provide good performance under most circumstances. This section presents some performance tradeoffs to consider if you want to try to optimize throughput for your workload.

This section covers the following topics:

- Time vs. Space: Configuring Batch and Transaction Size
- Time vs. Correctness: Understanding -fastload Tradeoffs
- How Assignment Policy Affects Optimization
- Tuning Split Size and Thread Count for Local Mode
- Tuning Split Size for Distributed Mode

- Reducing Memory Consumption With Streaming
- Improving Throughput with -split_input

4.14.1 Time vs. Space: Configuring Batch and Transaction Size

You can tune the document insertion throughput and memory requirements of your job by configuring the batch size and transaction size of the job.

- `-batch_size` controls the number of updates per request to the server.
- `-transaction_size` controls the number of requests to the server per transaction.

The default batch size is 100 and the maximum batch size is 200. (However, some options can affect the default). The default transaction size is 10 and the maximum transaction size is 4000/*actualBatchSize*. This means that the default maximum number of updates per transaction is 1000, and updates per transaction can range from 20 to 4000.

Selecting a batch size is a speed vs. memory tradeoff. Each request to the server introduces overhead because extra work must be done. However, unless you use `-streaming` or `-document_type mixed`, all the updates in a batch stay in memory until a request is sent, so larger batches consume more more memory.

Transactions introduce overhead on MarkLogic Server, so performing multiple updates per transaction can improve insertion throughput. However, an open transaction holds locks on fragments with pending updates, potentially increasing lock contention and affecting overall application performance.

It is also possible to overwhelm MarkLogic Server if you have too many concurrent sessions active.

4.14.2 Time vs. Correctness: Understanding -fastload Tradeoffs

The `-fastload` option can significantly speed up ingestion during `import` and `copy` operations, but it can also cause problems if not used properly. This section describes how `-fastload` affects the behavior of mlcp and some of the tradeoffs associated with enabling it.

The optimizations described by this section are only enabled if you explicitly specify the `-fastload` or `-output_directory` options. (The `-output_directory` option implies `-fastload`).

> **Note:** The `-fastload` option work slightly different when used with `-restrict_hosts`. For details, see “How -restrict_hosts Affects -fastload” on page 84. The limitations of `-fastload` described in this section still apply.

By default, mlcp inserts documents into the database by distributing work across the e-nodes in your MarkLogic cluster. Each e-node inserts documents into the database according to the configured document assignment policy.

This means the default insertion process for a document is similar to the following:

1. mlcp selects Host A from the available e-nodes in the cluster and sends it the document to be inserted.

2. Using the document assignment policy configured for the database, Host A determines the document should be inserted into Forest F on Host B.

3. Host A sends the document to Host B for insertion.

When you use `-fastload` (or `-output_directory`), mlcp attempts to cut out the middle step by applying the document assignment policy on the client. The interaction becomes similar to the following:

1. Using the document assignment policy, mlcp determines the document should be inserted into Forest F on Host B.

2. mlcp sends the document to Host B for insertion, with instructions to insert it into a specific forest.

Pre-determining the destination host and forest can always be done safely and consistently if the all of the following conditions are met:

- Your forest topology is stable.
- You are creating rather than updating documents.

To make forest assignment decisions locally, mlcp gathers information about the database assignment policy and forest topology at the beginning of a job. If you change the assignment policy or forest topology while an mlcp `import` or `copy` operation is running, mlcp might make forest placement decisions inconsistent with those MarkLogic Server would make. This can cause problems such as duplicate document URIs and unbalanced forests.

Similar problems can occur if mlcp attempts to update a document already in the database, and the forest topology or assignment policy changes between the time the document was originally inserted and the time mlcp updates the document. Using user-specified forest placement when initially inserting a document creates the same conflict.

Therefore, it is not safe to enable `-fastload` optimizations in the following situations:

- A document mlcp inserts already exists in the database and any of the following conditions are true:
    - The forest topology has changed since the document was originally inserted.
    - The assignment policy has changed since the document was originally inserted.
    - The assignment policy is not Legacy (default) or Bucket. For details, see “How Assignment Policy Affects Optimization” on page 61.

    - The document was originally inserted using user-specified forest placement.
- A document mlcp inserts does not already exist in the database and any of the following conditions are true:
    - The forest topology changes while mlcp is running.
    - The assignment policy changes while mlcp is running.

Assignment policy is a database configuration setting that affects how MarkLogic Server selects what forest to insert a document into or move a document into during rebalancing. For details, see Rebalancer Document Assignment Policies in *Administrator's Guide*.

> **Note:** Assignment policy was introduced with MarkLogic 7 and mlcp v1.2. If you use an earlier version of mlcp with MarkLogic 7 or later, the database you import data into with `-fastload` or `-output_directory` must be using the legacy assignment policy.

Any operation that changes the forests available for updates changes your forest topology, including the following:

- Adding or an employing a new forest
- Removing or retiring an existing forest
- Changing the `updates-allowed` state of forest. For example, calling `admin:forest-set-updates-allowed`
- Changing the database assignment policy

In most cases, it is your responsibility to determine whether or not you can safely use `-fastload` (or `-output_directory`, which implies `-fastload`). In cases where mlcp can detect `-fastload` is unsafe, it will disable it or give you an error.

4.14.3 How Assignment Policy Affects Optimization

This section describes how your choice of document assignment policy can introduce additional limitations and risks. Assignment policy is a database configuration setting that affects how MarkLogic Server selects what forest to insert a document into or move a document into during rebalancing. For details, see Rebalancer Document Assignment Policies in *Administrator's Guide*.

> **Note:** Assignment policy was introduced with MarkLogic 7 and mlcp v1.2. If you use an earlier version of mlcp with MarkLogic 7 or later, the database you import data into with `-fastload` or `-output_directory` must be using the legacy assignment policy.

The following table summarizes the limitations imposed by each assignment policy. If you do not explicitly set assignment policy, the default is Legacy or Bucket.

Assignment Policy	Notes
Legacy (default) Bucket	You can safely use `-fastload` if: • there are no pre-existing documents in the database with the same URIs; or • you use `-output_directory`; or • the URIs may be in use, but the forest topology has not changed since the documents were created, and the documents were not initially inserted using user-specified forest placement.
Statistical	You can only use `-fastload` to create new documents; updates are not supported. You should use `-output_directory` to ensure there are no updates. You cannot use `-fastload` in distributed mode with this assignment policy. All documents in a batch are inserted into the same forest. The rebalancer may subsequently move the documents if the batch size is large enough to cause the forest to become unbalanced. If you set `-fastload` to `true` and mlcp determines database rebalancing is occurring or needs to be done at the start of a job, an error occurs.
Range	You can only use `-fastload` to create new documents; updates are not supported. You should use `-output_directory` to ensure there are no updates. You cannot use `-fastload` in distributed mode with this assignment policy. You should use `-output_partition` to tell mlcp which partition to insert documents into. The partition you specify is used even if it is not the correct partition according to your configured partition policy. You can only use `-fastload` optimizations with range policy if you are licensed for Tiered Storage. If you set `-fastload` to `true` and mlcp determines database rebalancing is occurring or needs to be done at the start of a job, an error occurs.

Assignment Policy	Notes
Query	You can only use -fastload to create new documents; updates are not supported. You should use -output_directory to ensure there are no updates. You cannot use -fastload in distributed mode with this assignment policy. You should use -output_partition to tell mlcp which partition to insert documents into. The partition you specify is used even if it is not the correct partition according to your configured partition policy. You can only use -fastload optimizations with range policy if you are licensed for Tiered Storage. If you set -fastload to true and mlcp determines database rebalancing is occurring or needs to be done at the start of a job, an error occurs.

4.14.4 Tuning Split Size and Thread Count for Local Mode

You can tune split size only when importing documents in local mode from one of the following input file types:

- Whole documents (-input_file_type documents), whether from flat or compressed files.
- Composite file types that support -split_input, such as delimited_text.

You cannot tune split size when creating documents from composite files that do not support -split_input, such as sequence files and aggregate XML files.

You can tune thread count for both whole documents and all composite files types. Thread count and split size can interact to affect job performance.

In local mode, a split defines the unit of work per thread devoted to a session with MarkLogic Server. The ideal split size is one that keeps all mlcp's session threads busy. The default split size is 32M for local mode. Use the -max_split_size, -thread_count, and -thread_count_per_split options to tune your load.

By default, threads are assigned to splits in a round-robin fashion. For example, consider a loading 120 small documents of length 1M. Since the default split size is 32M, the load is broken into 4 splits. If -thread_count is 10, each split is assigned to at least 2 threads (10 / 4 = 2). The remaining 2 threads are each assigned to a split, so the number of threads per split are distributed as follows:

```
Split 1: 3 threads
Split 2: 3 threads
Split 3: 2 threads
Split 4: 2 threads
```

This distribution could result in two of the splits completing faster, leaving some threads idle. If you set `-max_split_size` to 12M, the loads has 10 splits, which can be evenly distributed across the threads and may result in better thread utilization.

If `-thread_count` is less than the number of splits, the default behavior is one thread per split, up to the total number of threads. The remaining splits must wait until a thread becomes available.

If MarkLogic Server is not I/O bound, then raising the thread count, and possibly threads per split, can improve throughput when the number of splits is small but each split is very large. This is often applicable to loading from zip files, aggregate files, and delimited text files. Note that if MarkLogic Server is already I/O bound in your environment, increasing the concurrency of writes will not necessarily improve performance.

4.14.5 Tuning Split Size for Distributed Mode

This topic applies only to importing documents in distributed mode using one of of the following input file types:

- Whole documents (`-input_file_type documents`), whether from flat or compressed files.
- Composite file types that support `-split_input`, such as `delimited_text`.

This topic does not apply to creating documents from composite files that do not support `-split_input`, such as sequence files and aggregate XML files. Split size for such composite files is not tunable.

Distributed mode uses Hadoop to import documents from multiple tasks, running on the nodes of a Hadoop cluster. A *split* is a unit of work for one Hadoop task.

Tuning Hadoop performance, including split size, is a complex topic outside the scope of this document. However, for best performance, tune split size to maximize parallelism across your Hadoop cluster, with each task taking at least a couple minutes. If your split size is too small and each task only runs for a very short time, the task overhead can degrade overall performance.

In distributed mode, split size is determined by the following formula:

```
max(min_split_size, min(max_split_size, block_size))
```

The default `min_split_size` is 0 and the default `max_split_size` is `Long.MAX` (the maximum signed long integer). The `block_size` depends on your Hadoop configuration, but the default is 64M. You can configure the min and max split size using the mlcp options `-min_split_size` and `-max_split_size`. You can only tune block size through Hadoop configuration settings.

The Hadoop configuration parameter `mapred.tasktracker.map.tasks.maximum` controls the maximum number of map tasks assigned to each worker node in your Hadoop cluster. You can use the mlcp option `-thread_count_per_split` to tune the maximum number of threads assigned to each task. If you set the value so high that the JVM running a task runs out of memory, the task will fail and can end up in a re-try loop.

In addition to balancing the workload across MapReduce tasks, you must also consider the load on MarkLogic Server. Too many concurrent sessions can overtax CPU and memory resources.

4.14.6 Reducing Memory Consumption With Streaming

The streaming protocol allows you to insert a large document into the database without holding the entire document in memory. Streaming uploads documents to MarkLogic Server in 128k chunks.

Streaming content into the database usually requires less memory on the host running mlcp, but ingestion can be slower because it introduces additional network overhead. Streaming also does not take advantage of mlcp's builtin retry mechanism. If an error occurs that is normally retryable, the job will fail.

> **Note:** Streaming is only usable when `-input_file_type` is `documents`. You cannot use streaming with delimited text files, sequence files, or archives.

To use streaming, enable the `-streaming` option. For example:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -username user -password password -host localhost \
    -port 8000 -input_file_path /my/dir -streaming
```

4.14.7 Improving Throughput with -split_input

If you are loading documents from very large files, you might be able to improve throughput using the `-split_input` option. When `-split_input` is true, mlcp attempts to break large input files that would otherwise by processed in a single split into multiple splits. This enables portions of the input file to be loaded by multiple tasks (distributed mode) or threads (local mode). You will usually see more benefit from using `-split_input` in distributed mode than local mode.

> **Note:** This option can only be applied to composite input file types that logically produce multiple documents and for which mlcp can efficiently identify document boundaries, such as `delimited_text`. Not all composite file types are supported; for details, see "Import Command Line Options" on page 84.

In local mode, `-split_input` is false by default. In distribute mode, `-split_input` is true by default.

The `-split_input` option affects local mode as follows: Suppose you are importing a very large delimited text file in local mode with `-split_input` set to false and the data processed as a single split. The work might be performed by multiple threads (depending on the job configuration), but these threads read records from the input file synchronously. This can cause some read contention. If you set `-split_input` to true, then each thread is assigned its own chunk of input, resulting in less contention and greater concurrency.

The `-split_input` option affects distributed mode as follows: Suppose you are importing a very large delimited text file in distributed mode with `-split_input` set to false. By default, your file gets assigned to a single Hadoop task. If you set `-split_input` to true, then your file can be split among multiple tasks and distributed across your Hadoop cluster. This can result in better utilization of your Hadoop cluster and improved throughput.

The number of subdivisions is determined by the formula *`file-size / max-split-size`*, so you should also consider tuning split size to match your input data characteristics. For example, if your data consists 1 delimited text file containing 16M of data, you can observe the following interactions between `-split_input` and `-max_split_size`:

Input File Size	`-split_input`	Split Size	Number of Splits
16M	false	32M	1
16M	true	32M	1
16M	true	1M	16

Tuning the split size in this case potentially enables greater concurrency because the multiple splits can be assigned to different threads or tasks.

Split size is tunable using `-max_split_size`, `-min_split_size`, and block size. For details, see “Tuning Split Size and Thread Count for Local Mode” on page 63 and “Tuning Split Size for Distributed Mode” on page 64.

4.15 Transforming Content During Ingestion

You can create an XQuery or Server-Side JavaScript function and install it on MarkLogic Server to transform and enrich content before inserting it into the database. Your function runs on MarkLogic Server. You can use such functions with the mlcp `import` and `copy` commands.

- Creating a Custom XQuery Transformation
- Creating a Custom JavaScript Transformation
- Implementation Guidelines
- Installing a Custom Transformation
- Using a Custom Transformation
- Example: Server-Side Content Transformation
- Example: Changing the URI and Document Type

4.15.1 Creating a Custom XQuery Transformation

The following topics describe how to implement a server-side content transformation function in XQuery:

- Function Signature
- Input Parameters
- Expected Output
- Example Implementation

4.15.1.1 Function Signature

A custom transformation is an XQuery function module that conforms to the following interface. Your function receives a single input document, described by `$content`, and can generate zero, one, or many output documents.

```
declare function yourNamespace:transform(
  $content as map:map,
  $context as map:map)
as map:map*
```

4.15.1.2 Input Parameters

The table below describes the input parameters to a transform function:

Parameter	Description
`$content`	Data about the original input document. The map contains the following keys: • `uri` - The URI of the document being inserted into the database. • `value` - The contents of the input document, as a document node, binary node, or text node.
`$context`	Additional context information about the insertion, such as tranformation-specific parameter values. The map can contain the following keys when your transform function is invoked: • `transform_param` : The value passed by the client through the `-transform_param` option, if any. Your function is responsible for parsing and validation. • `collections` : Collection URIs specified by the `-output_collections` option. Value format: A sequence of strings. • `permissions` : Permissions specified by the `-output_permissions` option. Value format: A sequence of `sec:permission` elements, as produced by `xdmp:permission`. • `quality` : The document quality specified by the `-output_quality` parameter. Value format: An integer value. • `temporalCollection` : The temporal collection URI specified by the `-temporal-collection` parameter. Value format: A string.

The type of node your function receives in the “value” property of `$content` depends on the input document type, as determined by mlcp from the `-document_type` option or URI extension. For details, see “How mlcp Determines Document Type” on page 36. The type of node your function returns in the “value” property should follow the same guidelines.

The table below outlines the relationship between document type and the node type your transform function should expect.

Document Type	“value” node type
XML	`document-node`
JSON	`document-node`
BINARY	`binary-node`
TEXT	`text-node`

The collections, permissions, quality, and temporal collection metadata from the mlcp command line is made available to your function so that you can modify or replace the values. If a given metadata category is not specified on the command line, the key will not be present in the input map.

4.15.1.3 Expected Output

Your function can produce more than one output document. For each document, your function should return a `map:map`. The `map:map` for an output document must use the same keys as the `$content` map (`uri` and `value`).

> **Note:** Modifying the document URI in a transformation can cause duplicate URIs when combined with the `-fastload` option, so you should not use `-fastload` or `-output_directory` with a transformation module that changes URIs. For details, see “Time vs. Correctness: Understanding -fastload Tradeoffs” on page 59.

The documents returned by your transformation should be exactly as you want to insert them into the database. No further transformations are applied by the mlcp infrastructure. For example, a transform function cannot affect document type just by changing the URI. Instead, it must convert the document node. For details, see “Example: Changing the URI and Document Type” on page 81.

You can use the context parameter to specify collections, permissions, quality, and values metadata for the documents returned by your transform. Use the following keys and data formats for specifying various categories of metadata:

Context Map Key	Expected Value Format
collections	A sequence of strings containing collection URIs.
permissions	A sequence of sec:permission elements, each representing a capability and a role id. For details, see xdmp:permission.
quality	An integer value (or a string that can be converted to an integer).
metadata	A map:map containing key-value metadata.
temporalCollection	A string containing a temporal document collection URI.

For a description of the meaning of the keys, see “Input Parameters” on page 68.

If your function returns multiple documents, they will all share the metadata settings from the context parameter.

4.15.1.4 Example Implementation

The following example adds an attribute to incoming XML documents and leaves non-XML documents unmodified. The attribute value is specified on the mlcp command line, using the -transform_param option.

```
declare function example:transform(
  $content as map:map,
  $context as map:map
) as map:map*
{
  let $attr-value :=
   (map:get($context, "transform_param"), "UNDEFINED")[1]
  let $the-doc := map:get($content, "value")
  return
    if (fn:empty($the-doc/element()))
    then $content
    else
      let $root := $the-doc/*
      return (
        map:put($content, "value",
          document {
            $root/preceding-sibling::node(),
            element {fn:name($root)} {
              attribute { fn:QName("", "NEWATTR") } {$attr-value},
              $root/@*,
              $root/node()
            },
            $root/following-sibling::node()
          }
        ), $content
      )
};
```

For an end-to-end example of using this transform, see “Example: Server-Side Content Transformation” on page 76.

4.15.2 Creating a Custom JavaScript Transformation

The following topics describe how to implement a server-side content transformation function in Server-Side JavaScript:

- Function Signature
- Input Parameters
- Expected Output
- Example Implementation

4.15.2.1 Function Signature

A custom transformation is a JavaScript function module that conforms to the following interface. Your function receives a single input document, described by `$content`, and can generate zero, one, or many output documents.

```
function yourTransform(content, context)
```

4.15.2.2 Input Parameters

The `content` parameter is an object containing data about the original input document. The `content` parameter has the following form:

```
{ uri: string,
  value: node
}
```

The type of node your function receives in `content.value` depends on the input document type, as determined by mlcp from the `-document_type` option or URI extension. For details, see “How mlcp Determines Document Type” on page 36. The type of node your function returns in the `value` property should follow the same guidelines.

The table below outlines the relationship between document type and the node type your transform function should expect (or return).

Document Type	“value” node type
XML	`document-node`
JSON	`document-node`
BINARY	`binary-node`
TEXT	`text-node`

The `context` parameter can contain context information about the insertion, such as any transform-specific parameters passed on the mlcp command line. The `context` parameter has the following form:

```
{ transform_param: string,
  collections: [ string, ... ],
  permissions: [ object, ... ],
  quality: number,
  temporalCollection: string}
```

The following table describes the properties of the input parameters in more detail:

Parameter	Description
`content`	• `uri` - The URI of the document being inserted into the database. • `value` - The contents of the input document, as a document node, binary node, or text node; see below.
`context`	• `transform_param` - The value passed by the client through the `-transform_param` option, if any. Your function is responsible for parsing and validation of the input string. • `collections` : Collection URIs specified by the `-output_collections` option. Value format: An array of strings. • `permissions` : Permissions specified by the `-output_permissions` option. Value format: An array of permissions objects, as produced by `xdmp.permission`. • `quality` : The document quality specified by the `-output_quality` parameter. Value format: A number. • `temporalCollection` : The temporal collection URI specified by the `-temporal-collection` parameter. Value format: A string.

The collections, permissions, quality, and temporal collection metadata from the mlcp command line is made available to your function so that you can modify or replace the values. If a given metadata category is not specified on the command line, the property will not be present in the `context` object.

4.15.2.3 Expected Output

Your function can produce more than one output document. For each document, your function should return a JavaScript object containing the same properties as the content input parameter (`uri` and `value`). When returning multiple document objects, put them in a Sequence.

The document content returned by your transformation should be exactly as you want to insert them into the database. No further transformations are applied by the mlcp infrastructure. For example, a transform function cannot affect document type just by changing the URI. Instead, it must convert the document node. For details, see "Example: Changing the URI and Document Type" on page 81.

You can modify the `context` input parameter to specify collections, permissions, quality, and values metadata for the documents returned by your transform. Use the following property names and data formats for specifying various categories of metadata:

Context Property	Expected Value Format
`collections`	An array of strings, each repesenting a collection URIs.
`permissions`	An array of permission objects, each containing a `capability` and a `roleId` property. For details, see `xdmp:permission`.
`quality`	An integer value (or a string that can be converted to an integer).
`metadata`	An object where each property represents a key-value metadata item.
`temporalCollection`	A string containing a temporal document collection URI.

For a description of the meaning of the keys, see "Input Parameters" on page 71.

If your function returns multiple documents, they will all share the metadata settings from the `context` parameter.

4.15.2.4 Example Implementation

The following example adds a property named "NEWPROP" to incoming JSON documents and leaves non-JSON documents unmodified. The property value is specified on the mlcp command line, using the `-transform_param` option.

```
// Add a property named "NEWPROP" to any JSON input document.
// Otherwise, input passes through unchanged.
```

```
function addProp(content, context)
{
  const propVal = (context.transform_param == undefined)
                 ? "UNDEFINED" : context.transform_param;

  if (xdmp.nodeKind(content.value) == 'document' &&
      content.value.documentFormat == 'JSON') {
    // Convert input to mutable object and add new property
    const newDoc = content.value.toObject();
    newDoc.NEWPROP = propVal;

    // Convert result back into a document
    content.value = xdmp.unquote(xdmp.quote(newDoc));
  }
  return content;
};

exports.addProp = addProp;
```

4.15.3 Implementation Guidelines

You should be aware of the following guidelines and limitations when implementing your transformation function:

- If you use a server-side transform with `-fastload` (or `-output_directory`, which enables `-fastload`), your transformation function only has access to database content in the same forest as the input document. If your transformation function needs general access to the database, do not use `-fastload` or `-output_directory`.

4.15.4 Installing a Custom Transformation

Install the XQuery library module containing your function into the modules database or modules root directory of the XDBC App Server associated with the destination database. For `import` operations, this is the App Server identified by `-host` and `-port` mlcp command line options. For `copy` operations, this is the App Server identified by `-output_host` and `-output_port` mlcp command line options.

Best practice is to install your libraries into the modules database of your XDBC App Server. If you install your module into the modules database, MarkLogic Server automatically makes the implementation available throughout your MarkLogic Server cluster. If you choose to install dependent libraries into the Modules directory of your MarkLogic Server installation, you must manually do so on each node in your cluster.

MarkLogic Server supports several methods for loading modules into the modules database:

- Run an XQuery or JavaScript query in Query Console. For example, you can run a query similar to the following to install a module using Query Console. Note: First select your modules database in the Query Console Content Source dropdown.

```
xquery version "1.0-ml";
xdmp:document-load("/space/mlcp/transform.xqy",
    <options xmlns="xdmp:document-load">
      <uri>/example/mlcp-transform.xqy</uri>
      <repair>none</repair>
      <permissions>{xdmp:default-permissions()}</permissions>
    </options>)
```

- If you use the App Server on port 8000 or have a REST API instance, you can use any of the following Client APIs:
  - Java: `ResourceExtensionsManager.write`. For details, see Managing Dependent Libraries and Other Assets in the *Java Application Developer's Guide*.
  - Node.js: `DatabaseClient.config.extlibs`. For details, see Managing Assets in the Modules Database in the *Node.js Application Developer's Guide*.
  - REST: `PUT /v1/ext/{directories}/{asset}`. For details, see Managing Dependent Libraries and Other Assets in the *REST Application Developer's Guide*.

If you use the filesystem instead of a modules database, you can manually install your module into the Modules directory. Copy the module into *`MARKLOGIC_INSTALL_DIR`*`/Modules` or into a subdirectory of this directory. The default location of this directory is:

- Unix: `/opt/MarkLogic/Modules`
- Windows: `C:\Program Files\MarkLogic\Modules`

If your transformation function requires other modules, you should also install the dependent libraries in the modules database or the modules directory.

For a complete example, see "Example: Server-Side Content Transformation" on page 76.

4.15.5 Using a Custom Transformation

Once you install a custom transformation function on MarkLogic Server, you can apply it to your mlcp `import` or `copy` job using the following options:

- `-transform_module` - The path to the module containing your transformation. Required.
- `-transform_namespace` - The namespace of your transformation function. If omitted, no namespace is assumed.
- `-transform_function` - The localname of your transformation function. If omitted, the name `transform` is assumed.
- `-transform_param` - Optional additional string data to be passed through to your transformation function.

Take note of the following limitations:

- When `-fastload` is in effect, your transform function runs in the scope of a single forest (the forest mlcp determines is the appropriate destination for the file being inserted). This means if you change the document URI as part of your transform, you can end up creating documents with duplicate URIs.

- When you use a transform function, all the documents in each batch are transformed and inserted into the database as a single statement. This means, for example, that if the (transformed) batch contain more than one document with the same URI, you will get an XDMP-CONFLICTINGUPDATES error.

The following example command assumes you previously installed a transform module with path /example/mlcp-transform.xqy, and that the function implements a transform function (the default function) in the namespace http://marklogic.com/example. The function expects a user-defined parameter value, supplied using the -transform_param option.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -mode local -host mlhost -port 8000 \
    -username user -password password \
    -input_file_path /space/mlcp-test/data \
    -transform_module /example/mlcp-transform.xqy \
    -transform_namespace "http://marklogic.com/example" \
    -transform_param "my-value"
```

For a complete example, see “Example: Server-Side Content Transformation” on page 76.

4.15.6 Example: Server-Side Content Transformation

This example walks you through installing and using an XQuery or Server-Side JavaScript transform function to modify content ingested with mlcp. The example XQuery transform function modifies XML documents by adding an attribute named NEWATTR, with an attribute value specified on the mlcp command line. The example JavaScript transform function modifies JSON documents by adding a new property named NEWPROP, with a value specified on the mlcp command line.

This example assumes you have already created an XDBC App Server, configured to use "/" as the root and a modules database of Modules.

1. Create the sample input files

2. Create the XQuery transform module

3. Create the JavaScript transform module

4. Install the transformation module

5. Apply the transformation

4.15.6.1 Create the sample input files

This section walks you through creating sample input data to be ingested by mlcp. You can use other data.

1. Create a directory to hold the sample input data. For example:

   ```
   $ mkdir /space/mlcp/txform/data
   ```

2. Create a file named `txform.xml` in the sample data directory with the following contents:

   ```
   <parent><child/></parent>
   ```

3. Create a file named `txform.json` in the sample data directory with the following contents:

   ```
   { "key": "value" }
   ```

4.15.6.2 Create the XQuery transform module

If you prefer to work with a Server-Side JavaScript transform function, skip this section and go to “Create the JavaScript transform module” on page 78.

This example module modifies XML input documents by adding an attribute named `NEWATTR`. Other input document types pass through the transform unmodified.

In a location other than the sample input data directory, create a file named `transform.xqy` with the following contents. For example, copy the following into `/space/mlcp/txform/transform.xqy`.

```
xquery version "1.0-ml";
module namespace example = "http://marklogic.com/example";

(: If the input document is XML, insert @NEWATTR, with the value
 : specified in the input parameter. If the input document is not
 : XML, leave it as-is.
 :)
declare function example:transform(
  $content as map:map,
  $context as map:map
) as map:map*
{
  let $attr-value :=
    (map:get($context, "transform_param"), "UNDEFINED")[1]
  let $the-doc := map:get($content, "value")
  return
    if (fn:empty($the-doc/element()))
    then $content
    else
      let $root := $the-doc/*
      return (
        map:put($content, "value",
          document {
            $root/preceding-sibling::node(),
            element {fn:name($root)} {
              attribute { fn:QName("", "NEWATTR") } {$attr-value},
              $root/@*,
              $root/node()
```

```
                },
                $root/following-sibling::node()
              }
          ), $content
        )
    };
```

4.15.6.3 Create the JavaScript transform module

If you prefer to work with an XQuery transform function, skip this section and go to “Create the XQuery transform module” on page 77.

This example module modifies JSON input documents by adding an attribute named `NEWPROP`. Other input document types pass through the transform unmodified.

In a location other than the sample input data directory, create a file named `transform.xqy` with the following contents. For example, copy the following into `/space/mlcp/txform/transform.sjs`.

```
// Add a property named "NEWPROP" to any JSON input document.
// Otherwise, input passes through unchanged.

function addProp(content, context)
{
  var propVal = (context.transform_param == undefined)
                ? "UNDEFINED" : context.transform_param;

  var docType = xdmp.nodeKind(content.value);
  if (xdmp.nodeKind(content.value) == 'document' &&
      content.value.documentFormat == 'JSON') {
    // Convert input to mutable object and add new property
    var newDoc = content.value.toObject();
    newDoc.NEWPROP = propVal;

    // Convert result back into a document
    content.value = xdmp.unquote(xdmp.quote(newDoc));
  }
  return content;
};

exports.transform = addProp;
```

4.15.6.4 Install the transformation module

This section walks you through installing the transform module(s) created in “Create the XQuery transform module” on page 77 or “Create the JavaScript transform module” on page 78.

These instructions assume you use the XDBC App Server and Documents database pre-configured on port 8000. This procedure installs the module using Query Console. You can use another method.

For more detailed instructions on using Query Console, see *Query Console User Guide*.

1. Navigate to Query Console in your browser:

   ```
   http://yourhost:8000/qconsole/
   ```

2. Create a new query by clicking the "+" at the top of the query editor.

3. Select XQuery in the Query Type dropdown.

4. Install the XQuery and/or JavaScript module by copying one of the following scripts into the new query. Modify the first parameter of `xdmp:document-load` to match the path to the transform module you previously created.

   a. To install the XQuery module, use the following script:

   ```
   xquery version "1.0-ml";
   xdmp:document-load("/space/mlcp/txform/transform.xqy",
       <options xmlns="xdmp:document-load">
         <uri>/example/mlcp-transform.xqy</uri>
         <repair>none</repair>
         <permissions>{xdmp:default-permissions()}</permissions>
       </options>)
   ```

   b. To install the JavaScript module, use the following script:.

   ```
   xquery version "1.0-ml";
   xdmp:document-load("/space/mlcp/txform/transform.sjs",
       <options xmlns="xdmp:document-load">
         <uri>/example/mlcp-transform.sjs</uri>
         <repair>none</repair>
         <permissions>{xdmp:default-permissions()}</permissions>
       </options>)
   ```

5. Select the modules database of your XDBC App Server in the Content Source dropdown at the top of the query editor. If you use the XDBC App Server on port 8000, this is the database named Modules.

6. Click the Run button. Your module is installed in the modules database.

7. To confirm installation of your module, click the Explore button at the top of the query editor and note your module installed with URI `/example/mlcp-transform.xqy` or `/example/mlcp-transform.sjs`.

4.15.6.5 Apply the transformation

To ingest the sample documents and apply the previously installed transformation, use a command similar to the following. Change the `username`, `password`, `host`, `port`, and `input_file_path` options to match your environment.

Use a command similar to the following if you installed the XQuery transform module:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -mode local -host mlhost -port 8000 \
    -username user -password password \
    -input_file_path /space/mlcp/txform/data \
    -transform_module /example/mlcp-transform.xqy \
    -transform_namespace "http://marklogic.com/example" \
    -transform_param "my-value"
```

Use a command similar to the following if you installed the JavaScript transform module:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -mode local -host mlhost -port 8000 \
    -username user -password password \
    -input_file_path /space/mlcp/txform/data \
    -transform_module /example/mlcp-transform.sjs \
    -transform_function transform \
    -transform_param "my-value"
```

Mlcp should report creating two documents. Near the end of the mlcp output, you should see lines similar to the following:

```
... INFO contentpump.LocalJobRunner: OUTPUT_RECORDS: 2
... INFO contentpump.LocalJobRunner: Total execution time: 1 sec
```

Use Query Console to explore the content database associated with your XDBC App Server. Confirm that mlcp created 2 documents. If your input was in the directory `/space/mlcp/txform/data`, then the document URIs will be:

- `/space/mlcp/txform/data/txform.xml`
- `/space/mlcp/txform/data/txform.json`

If you use the XQuery transform, then exploring the contents of `txform.xml` in the database should show a NEWATTR attribute was inserted by the transform, with the value from `-transform_param`. The document contents should be as follows:

```
<parent NEWATTR="my-value">
  <child/>
</parent>
```

If you use the JavaScript transform, then exploring the contents of txform.json in the database should show a NEWPROP property was inserted by the transform, with the value from `-transform_param`. The document contents should be as follows:

```
{ "key": "value", "NEWPROP": "my-value"}
```

4.15.7 Example: Changing the URI and Document Type

This example demonstrates changing the type of a document from binary to XML and changing the document URI to match.

> **Note:** Transforms that change the document URI should not be combined with the `-fastload` or `-output_directory` options as they can cause duplicate document URIs. For details, see "Time vs. Correctness: Understanding -fastload Tradeoffs" on page 59.

As described in "How mlcp Determines Document Type" on page 36, the URI extension and MIME type mapping are used to determine document type when you use `-document_type mixed`. However, transform functions do not run until after document type selection is completed. Therefore, if you want to affect document type in a transform, you must convert the document node, as well as optionally changing the output URI.

Suppose your input document set generates an output document URI with the unmapped extension ".1", such as `/path/doc.1`. Since "1" is not a recognized URI extension, mlcp creates a binary document node from this input file by default. The example transform function in this section intercepts such a document and transforms it into an XML document.

- XQuery Implementation
- JavaScript Implementation

Note that if you define a MIME type mapping that maps the extension "1" to XML (or JSON) in your MarkLogic Server configuration, then mlcp creates a document of the appropriate type to begin with, and this conversion becomes unnecessary.

4.15.7.1 XQuery Implementation

This module detects input documents with URI suffixes of the form ".1" and converts them into XML documents with a ".xml" URI extension. Note that the transform does not snoop the content to ensure it is actually XML.

```
xquery version "1.0-ml";
module namespace example = "http://marklogic.com/example";

declare function example:mod_doc_type(
  $content as map:map,
  $context as map:map
) as map:map*
{
  let $orig-uri := map:get($content, "uri")
  return
  if (fn:substring-after($orig-uri, ".") = "1") then
    let $doc-type := xdmp:node-kind(map:get($content, "value"))
    return (
      (: change the URI to an xml suffix :)
      map:put($content, "uri",
        fn:concat(fn:substring-before($orig-uri, "."), ".xml")
      ),
      (: convert the input from binary node to xml document node :)
      if ($doc-type = "binary") then
        map:put(
          $content, "value",
          xdmp:unquote(xdmp:quote(map:get($content, "value")))
        )
      else (),
      $content
    )
  else $content
};
```

4.15.7.2 JavaScript Implementation

This module detects input documents with URI suffixes of the form “.1” and converts them into JSON documents with a “.json” URI extension. Note that the transform does not snoop the content to ensure it is actually JSON.

```
function modDocType(content, context)
{
  var uri = String(content.uri);
  var dot = uri.lastIndexOf('.');
  if (dot > 0) {
    var suffix = uri.slice(dot);
    if (suffix == '.1') {
      content.uri = uri.substring(0,dot+1) + 'json';
      if (xdmp.nodeKind(content.value) == 'binary') {
        // convert the content to a JSON document
        content.value = xdmp.unquote(xdmp.quote(content.value));
      }
    }
  }
  return content;
};

exports.transform = modDocType;
```

4.16 Controlling How mlcp Connects to MarkLogic

This section describes how mlcp connects to MarkLogic by default, and options you can use to modify this behavior. For example, you can force mlcp to only connect to MarkLogic through a load balancer host.

See the following topics for more details:

- How mlcp Uses the Host List
- Restricting the Hosts mlcp Uses to Connect to MarkLogic

- How -restrict_hosts Affects -fastload

4.16.1 How mlcp Uses the Host List

You must specify at least one host with `-host` command line option. You can specify multiple hosts.

If any hostname listed in the value of the `-host` option is not resolvable by mlcp at the beginning of a job, then mlcp will abort the job with an `IllegalArgumentException`.

Assuming all hostnames are resolvable, mlcp uses the first of these hosts to gather information about the target database. If mlcp is unable to connect to the first host in the `-host` list, then mlcp will move on to the next host in the list. If mlcp cannot connect to any of the listed hosts, then the job will fail with an `IOException`.

If mlcp successfully retrieves a list of forest hosts, then mlcp subsequently connects directly to these hosts when distributing work across the cluster, whether or not these hosts are specified in the `-host` option. In this way, your job does not need to be aware cluster topology.

This behavior applies to the `import`, `export`, and `copy` commands. (For a copy job, you specify hosts through `-input_host` and `-output_host`, rather than `-host`.)

You can also restrict mlcp to just the hosts listed by the `-host` option. For details, see "Restricting the Hosts mlcp Uses to Connect to MarkLogic" on page 83.

4.16.2 Restricting the Hosts mlcp Uses to Connect to MarkLogic

You can restrict the hosts to which mlcp distributes work using the `-restrict_hosts` and `-host` command line options. You might find this option combination useful in situations such as the following:

- Limit the host working set to just the e-nodes in your cluster.
- Your cluster sits behind a load balancer and mlcp should always connect to the load balancer hosts(s) rather than making a direct connection to hosts in the cluster.
- The public and private DNS names of a host differ, such as can occur for an AWS instance.

If you set `-restrict_hosts` to true, then mlcp will only connect to the hosts listed in the `-host` option, rather than using the approach described in "How mlcp Uses the Host List" on page 83.

> **Note:** Using `-restrict_hosts` will usually degrade the performance of an mlcp job because mlcp cannot distribute work as efficiently.

For example, if you're using mlcp with a load balancer between your client and your MarkLogic cluster, you can specify the load balancer with `-host` and set `-restrict_hosts` to true to prevent mlcp from attempting to bypass the load balancer and connect directly to the forest hosts.

You can restrict mlcp's host list when using the `import`, `export`, and `copy` commands. For import and export, use the `-host` and `-restrict_hosts options`. For copy, use `-input_host` and `-restrict_input_hosts` and/or `-output_host` and `-restrict_output_hosts`.

4.16.3 How -restrict_hosts Affects -fastload

You can use `-fastload` with `-restrict_hosts`. The performance improvement from `-fastload` will be less than if you did not use `-restrict_hosts`, but better than if you do not use `-fastload`. The usual cautions about `-fastload` apply; see "Time vs. Correctness: Understanding -fastload Tradeoffs" on page 59.

The `-fastload` and `-restrict_hosts` options interact as follows:

Without `-restrict_hosts`, mlcp figures out which hosts contains the destination forest for a document, and then connects directly to that host. When `-restrict_hosts` is true, a connection to the forest host might not possible. In this case, mlcp connects to an allowed e-node, and includes the detailed destination information along with the document. The destination details makes an insertion faster than it would otherwise be.

4.17 Import Command Line Options

This section summarizes the command line options available with the mlcp `import` command. The following command line options define your connection to MarkLogic:

Option	Description
`-host` *`comma-list`*	Required. A comma separated list of hosts through which mlcp can connect to the destination MarkLogic Server. You must specify at least one host. For more details, see "How mlcp Uses the Host List" on page 83.
`-port` *`number`*	Port number of the destination MarkLogic Server. There should be an XDBC App Server on this port. Default: 8000.
`-username` *`string`*	MarkLogic Server user with which to import documents. Required, unless using Kerberos authentication.
`-password` *`string`*	Password for the MarkLogic Server user specified with `-username`. Required, unless using Kerberos authentication.

The following table lists command line options that define the characteristics of the import operation:

Option	Description
`-aggregate_record_element` *`string`*	When splitting an aggregate input file into multiple documents, the name of the element to use as the output document root. Default: The first child element under the root element.
`-aggregate_record_namespace` *`string`*	The namespace of the element specificed by `-aggregate_record_element_name`. Default: No namespace.
`-aggregate_uri_id` *`string`*	Deprecated. Use `-uri_id` instead. When splitting an aggregate input file into multiple documents, the element or attribute name within the document root to use as the document URI. Default: In local mode, *`hashcode-seqnum`*, where the hashcode is derived from the split number; in distribute mode, *`taskid-seqnum`*.
`-batch_size` *`number`*	The number of documents to process in a single request to MarkLogic Server. Default: 100. Maximum: 200.
`-collection_filter` *`comma-list`*	A comma-separated list of collection URIs. Only usable with `-input_file_type forest`. mlcp extracts only documents in these collections. This option can be combined with other filter options. Default: Import all documents.
`-conf` *`filename`*	Pass extra setting to Hadoop when using distributed mode. For details, see “Setting Custom Hadoop Options and Properties” on page 16. This option must appear before mlcp-specific options.

Option	Description
`-content_encoding` *`string`*	The character encoding of input documents when `-input_file_type` is `documents`, `aggregates`, `delimited_text`, or `rdf`. The option value must be a character set name accepted by your JVM; see `java.nio.charset.Charset`. Default: `UTF-8`. Set to `system` to use the platform default encoding for the host on which mlcp runs.
`-copy_collections` *`boolean`*	When importing documents from an archive, whether to copy document collections from the source archive to the destination. Only applies when `-input_file_type` is `archive` or `forest`. Default: true.
`-copy_metadata` *`boolean`*	When importing documents from an archive, whether to copy document key-value metadata from the source archive to the destination. Only applies when `-input_file_type` is `archive` or `forest`. Default: true.
`-copy_permissions` *`boolean`*	When importing documents from an archive, whether to copy document permissions from the source archive to the destination. Only applies with `-input_file_type archive`. Default: true.
`-copy_properties` *`boolean`*	When importing documents from an archive, whether to copy document properties from the source archive to the destination. Only applies with `-input_file_type archive`. Default: true.
`-copy_quality` *`boolean`*	When importing documents from an archive, whether to copy document quality from the source archive to the destination. Only applies when `-input_file_type` is `archive` or `forest`. Default: true.
`-D` *`property=value`*	Pass a configuration property setting to Hadoop when using distributed mode. For details, see “Setting Custom Hadoop Options and Properties” on page 16. This option must appear before mlcp-specific options.

Option	Description
-data_type *comma-list*	When importing content with -input_file_type delimited_text and -document_type json, use this option to specify the data type (string, number, or boolean) to give to specific fields. The option value must be a comma separated list of *name,datatype* pairs, such as "a,number,b,boolean". Default: All fields have string type. For details, see "Controlling Data Type in JSON Output" on page 46.
-database *string*	The name of the destination database. Default: The database associated with the destination App Server identified by -host and -port.
-delimiter *character*	When importing content with -input_file_type delimited_text, the delimiting character. Default: comma (,).
-delimited_root_name *string*	When importing content with -input_file_type delimited_text, the localname of the document root element. Default: root.
-delimited_uri_id *string*	Deprecated. use -uri_id instead. When importing content -input_file_type delimited_text, the column name that contributes to the id portion of the URI for inserted documents. Default: The first column.
-directory_filter *comma-list*	A comma-separated list of database directory names. Only usable with -input_file_type forest. mlcp extracts only documents from these directories, plus related metadata. Directory names should usually end with "/". This option can be combined with other filter options. Default: Import all documents.
-document_type *string*	The type of document to create when -input_file_type is documents, sequencefile or delimited_text. Accepted values: mixed (documents only), xml, json, text, binary. Default: mixed for documents, xml for sequencefile, and xml for delimited_text.

Option	Description
`-fastload` *boolean*	Whether or not to force optimal performance, even at the risk of creating duplicate document URIs. See “Time vs. Correctness: Understanding -fastload Tradeoffs” on page 59. Default: `false`.
`-filename_as_collection` *boolean*	Add each loaded document to a collection corresponding to the name of the input file. You cannot use this option when `-input_file_type` is `rdf` or `forest`. Useful when splitting an input file into multiple documents. If the filename contains characters not permitted in a URI, those characters are URI encoded. Default: `false`.
`-generate_uri` *boolean*	When importing content with -`input_file_type delimited_text`, or `-input_file_type delimited_json`, whether or not MarkLogic Server should automatically generate document URIs. Default: `false` for `delimited_text`, `true` for `delimited_json`. For details, see “Default Document URI Construction” on page 33.
`-hadoop_conf_dir` *string*	When using distributed mode, the Hadoop config directory. For details, see “Configuring Distributed Mode” on page 15.
`-archive_metadata_optional` *boolean*	When importing documents from a database archive, whether or not to ignore missing metadata files. If this is `false` and the archive contains no metadata, an error occurs. Default: `false`.
`-input_compressed` *boolean*	Whether or not the source data is compressed. Default: false.
`-input_compression_codec` *string*	When `-input_compressed` is true, the code used for compression. Accepted values: `zip`, `gzip`.
`-input_file_path` *string*	A regular expression describing the filesystem location(s) to use for input. For details, see “Regular Expression Syntax” on page 9.

Option	Description
`-input_file_pattern` *string*	Load only input files that match this regular expression from the path(s) matched by `-input_file_path`. For details, see "Regular Expression Syntax" on page 9. Default: Load all files. This option is ignored when `-input_file_type` is `forest`.
`-input_file_type` *type*	The input file type. Accepted value: `aggregates`, `archive`, `delimited_text`, `delimited_json`, `documents`, `forest`, `rdf`, `sequencefile`. Default: `documents`.
`-sequencefile_key_class` *string*	When importing Hadoop sequence files, the name of the Java class to use as the input key. Required when using `-input_file_type sequencefile`.
`-sequencefile_value_class` *string*	When importing Hadoop sequence files, the name of the Java class to use as the input value. Required when using `-input_file_type sequencefile`.
`-sequencefile_value_type` *string*	When importing Hadoop sequence files, the type of the value data returned by the class named by `-sequencefile_value_class`. Accepted values: `Text`, `BytesWritable`. (Values are case-insensitive). Default: `Text`.
`-max_split_size` *number*	When importing from files, the maximum number of bytes in one input split. Default: The maximum Long value (`Long.MAX_VALUE`).
`-min_split_size` *number*	When importing from files, the minimum number of bytes in one input split. Default: 0.
`-mode` *string*	Ingestion mode. Accepted values: `distributed`, `local`. Distributed mode requires Hadoop. Default: `local`, unless you set the `HADOOP_CONF_DIR` variable; for details, see "Configuring Distributed Mode" on page 15.
`-namespace` *string*	The default namespace for all XML documents created during loading.

Option	Description
`-options_file` *`string`*	Specify an options file pathname from which to read additional command line options. If you use an options file, this option must appear first. For details, see "Options File Syntax" on page 9.
`-output_cleandir` *`boolean`*	Whether or not to delete all content in the output database directory prior to loading. Default: `false`.
`-output_collections` *`comma-list`*	A comma separated list of collection URIs. Loaded documents are added to these collections.
`-output_directory` *`string`*	The destination database directory in which to create the loaded documents. If the directory exists, its contents are removed prior to ingesting new documents. Using this option enables `-fastload` by default, which can cause duplicate URIs to be created. See "Time vs. Correctness: Understanding -fastload Tradeoffs" on page 59.
`-output_graph` *`string`*	Only usable with `-input_file_type rdf`. For quad data, specifies the default graph for quads that do not include an explicit graph label. For other triple formats, specifies the graph into which to load all triples. For details, see "Loading Triples" on page 54.
`-output_language` *`string`*	The `xml:lang` to associate with loaded documents.
`-output_partition` *`string`*	The name of the database partition in which to create documents. For details, see "How Assignment Policy Affects Optimization" on page 61, and Range Partitions or Query Partitions in the *Administrator's Guide*.
`-output_override_graph` *`string`*	Only usable with `-input_file_type rdf`. The graph into which to load all triples. For quads, overrides any graph label in the quads. For details, see "Loading Triples" on page 54.

Option	Description
`-output_permissions` *`comma-list`*	A comma separated list of (`role,capability)` pairs to apply to loaded documents. Default: The default permissions associated with the user inserting the document. Example: `-output_permissions role1,read,role2,update`
`-output_quality` *`string`*	The quality of loaded documents. Default: 0.
`-output_uri_prefix` *`string`*	Specify a prefix to prepend to the default URI. Used to construct output document URIs. For details, see “Controlling Database URIs During Ingestion” on page 33.
`-output_uri_replace` *`comma-list`*	A comma separated list of (`regex,string`) pairs that define string replacements to apply to the URIs of documents added to the database. The replacement strings must be enclosed in single quotes. For example, `-output_uri_replace "regex1,'string1',regext2,'string2'"`
`-output_uri_suffix` *`string`*	Specify a suffix to append to the default URI Used to construct output document URIs. For details, see “Controlling Database URIs During Ingestion” on page 33.
`-restrict_hosts` *`boolean`*	Restrict mlcp to connect to MarkLogic only through the hosts listed in the `-host` option. For more details, see “Restricting the Hosts mlcp Uses to Connect to MarkLogic” on page 83.
`-split_input` *`boolean`*	Whether or not to divide input data into logical chunks to support more concurrency. Only supported when `-input_file_type` is one of the following: `delimited_text`. Default: `false` for local mode, `true` for distributed mode. Do not use this option with multi-byte character data. For details, see “Improving Throughput with -split_input” on page 65.
`-ssl` *`boolean`*	Enable/disable SSL secured communication with MarkLogic. Default: false. If you set this option to true, your App Server must be SSL enabled. For details, see “Connecting to MarkLogic Using SSL” on page 17.

Option	Description
`-streaming` *`boolean`*	Whether or not to stream documents to MarkLogic Server. Applies only when `-input_file_type` is `documents`.
`-temporal_collection` *`string`*	The temporal collection into which the temporal documents are to be loaded. For details on loading temporal documents into MarkLogic, see Using MarkLogic Content Pump (MLCP) to Load Temporal Documents in the *Temporal Developer's Guide*.
`-thread_count` *`number`*	The number of threads to spawn for concurrent loading. The total number of threads spawned by the process can be larger than this number, but this option caps the number of concurrent sessions with MarkLogic Server. Only available in `local` mode. Default: 4.
`-thread_count_per_split` *`number`*	The maximum number of threads that can be assigned to each split. In local mode, the default behavior is a round-robin allocation of threads to each split, which results in one thread per split if there are more splits than threads. The total number of available threads is controlled by `-thread_count`. For details, see "Tuning Split Size and Thread Count for Local Mode" on page 63. In distributed mode, each split is assigned the number of threads specified by this option. For details, see "Tuning Split Size for Distributed Mode" on page 64.

Option	Description
`-tolerate_errors` *`boolean`*	NOTE: This option is deprecated, ignored, and will be removed in a future release. Mlcp always behaves as if `-tolerate_errors` is true. Applicable only when `-batch_size` is greater than 1. When this option is true and batch size is greater than 1, if an error occurs for one or more documents during loading, only the erroneous documents are skipped; all other documents are inserted into the database. When this option is false or batch size is 1, errors during insertion can cause all the inserts in the current batch to be rolled back. Default: false.
`-transform_function` *`string`*	The localname of a custom content transformation function installed on MarkLogic Server. Ignored if `-transform_module` is not specified. Default: `transform`. For details, see “Transforming Content During Ingestion” on page 66.
`-transform_module` *`string`*	The path in the modules database or modules directory of a custom content transformation function installed on MarkLogic Server. This option is required to enable a custom transformation. For details, see “Transforming Content During Ingestion” on page 66.
`-transform_namespace` *`string`*	The namespace URI of the custom content transformation function named by `-transform_function`. Ignored if `-transform_module` is not specified. Default: no namespace. For details, see “Transforming Content During Ingestion” on page 66.
`-transform_param` *`string`*	Optional extra data to pass through to a custom transformation function. Ignored if `-transform_module` is not specified. Default: no namespace. For details, see “Transforming Content During Ingestion” on page 66.
`-transaction_size` *`number`*	The number of requests to MarkLogic Server per transaction. Default: 10. Maximum: 4000/*actualBatchSize*.

Option	Description
`-type_filter` *`comma-list`*	A comma-separated list of document types. Only usable with `-input_file_type forest`. mlcp imports only documents with these types. This option can be combined with other filter options. Default: Import all documents.
`-uri_id` *`string`*	Specify a field, XML element name, or JSON property name to use as the basis of the output document URIs when importing delimited text, aggregate XML, or line-delimited JSON data. With `-input_file_type aggregates` or `-input_file_type delimited_json`, the element, attribute, or property name within the document to use as the document URI. Default: None; the URI is based on the file name, as described in "Default Document URI Construction" on page 33. With -`input_file_type delimited_text`, the column name that contributes to the id portion of the URI for inserted documents. Default: The first column.
`-xml_repair_level` *`string`*	The degree of repair to attempt on XML documents in order to create well-formed XML. Accepted values: `default`, `full`, `none`. Default: `default`, which depends on the configured MarkLogic Server default XQuery version: In XQuery 1.0 and 1.0-ml the default is none. In XQuery 0.9-ml the default is full.

5.0 Exporting Content from MarkLogic Server

You can export content in a MarkLogic Server database to files or an archive. Use archives to copy content from one MarkLogic Server database to another. Output can be written to the native filesystem or to HDFS.

For a list of export related command line options, see “Export Command Line Options” on page 117.

You can also use mlcp to extract documents directly from offline forests. For details, see “Using Direct Access to Extract or Copy Documents” on page 135.

This section covers the following topics:

5.1 Exporting Documents as Files

Use the mlcp `export` command to export documents in their original format as files on the native filesystem or HDFS. For example, you can export an XML document as a text file containing XML, or a binary document as a JPG image.

To export documents from a database as files:

1. Select the files to export. For details, see “Filtering Document Exports” on page 99.

   - To select documents in one or more collections, set `-collection_filter` to a comma separated list of collection URIs.
   - To select documents in one or more database directories, set `-directory_filter` to a comma separated list of directory URIs.
   - To select documents matching an XPath expression, use `-document_selector`. To use namespace prefixes in the XPath expression, define the prefix binding using `-path_namespace`.

- To select documents matching a query, use `-query_filter`, alone or in combination with one of the other filter options. False postives are possible; for details, see "Understanding When Filters Are Accurate" on page 101.
- To select all documents in the database, leave `-collection_filter`, `-directory_filter`, `-document_selector`, and `-query_filter` unset.

2. Set `-output_file_path` to the destination file or directory on the native filesystem or HDFS.

3. To prettyprint exported XML when using local mode, set `-indented` to true.

Directory names specified with `-directory_filter` should end with "/".

When using `-document_selector` to filter by XPath expression, you can define namespace prefixes using the `-path_namespace` option. For example:

```
-path_namespace 'ex1,http://marklogic.com/example,ex2,http://my/ex2'
-document_selector '/ex1:elem[ex2:attr > 10]'
```

Note: Document URIs are URI-decoded before filesystem directories or filenames are constructed for them. For details, see "How URI Decoding Affects Output File Names" on page 99.

For a full list of export options, see "Export Command Line Options" on page 117.

The following example exports selected documents in the database to the native filesystem directory `/space/mlcp/export/files`. The directory filter selects only the documents in `/plays`.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8000 -username user \
    -password password -mode local -output_file_path \
    /space/mlcp/export/files -directory_filter /plays/
```

5.2 Exporting Documents to a Compressed File

Use the mlcp `export` command to export documents in their original format as files in a compressed ZIP file on the native filesystem or HDFS.

To export documents from a database as files:

1. Select the files to export. For details, see "Filtering Document Exports" on page 99.

- To select documents in one or more collections, set `-collection_filter` to a comma separated list of collection URIs.
- To select documents in one or more database directories, set `-directory_filter` to a comma separated list of directory URIs.

- To select documents matching an XPath expression, use `-document_selector`. To use namespace prefixes in the XPath expression, define the prefix binding using `-path_namespace`.
- To select documents matching a query, use `-query_filter`, alone or in combination with one of the other filter options. False postives are possible; for details, see “Understanding When Filters Are Accurate” on page 101.
- To select all documents in the database, leave `-collection_filter`, `-directory_filter`,`-document_selector`, and `-query_filter` unset.

2. Set `-output_file_path` to the destination directory on the native filesystem or HDFS. This directory must not already exist.

3. Set `-compress` to `true`.

4. To prettyprint exported XML when using local mode, set `-indented` to true.

For a full list of export options, see “Export Command Line Options” on page 117.

The zip files created by export have filenames of the form *`timestamp-seqnum`*`.zip`.

The following example exports all the documents in the database to the directory `/space/examples/export` on the native filesystem.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8000 -username user \
    -password password -mode local \
    -output_file_path /space/examples/export -compress true

$ ls /space/examples/export
20120823135307-0700-000000-XML.zip
```

5.3 Exporting to an Archive

Use the mlcp `export` command with an output type of `archive` to create a database archive that includes content and metadata. You can use the mlcp `import` command to copy the archive to another database or restore database contents.

To export database content to an archive file with mlcp:

1. Select the documents to export. For details, see “Filtering Archive and Copy Contents” on page 100.

   - To select documents in one or more collections, set `-collection_filter` to a comma separated list of collection URIs.
   - To select documents in one or more database directories, set `-directory_filter` to a comma separated list of directory URIs.

- To select documents matching an XPath expression, use `-document_selector`. To use namespace prefixes in the XPath expression, define the prefix binding using `-path_namespace`.
- To select documents matching a query, use `-query_filter`, alone or in combination with one of the other filter options. False postives are possible; for details, see “Understanding When Filters Are Accurate” on page 101.
- To select all documents in the database, leave `-collection_filter`, `-directory_filter`, `-document_selector`, and `-query_filter` unset.

2. Set `-output_file_path` to the destination directory on the native filesystem or HDFS. This directory must not already exist.

3. Set `-output_type` to `archive`.

4. If you want to exclude some or all document metadata from the archive:

    - Set `-copy_collections` to `false` to exclude document collections metadata.
    - Set `-copy_permissions` to `false` to exclude document permissions metadata.
    - Set `-copy_properties` to `false` to exclude document properties.
    - Set `-copy_quality` to `false` to exclude document quality metadata.
    - Set `-copy_metadata` to `false` to exclude document key-value metadata.

For a full list of export options, see “Export Command Line Options” on page 117.

The following example exports all documents and metadata to the directory `/space/examples/exported`. After export, the directory contains one or more compressed archive files.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8000 -username user \
    -password password -mode local \
    -output_file_path /space/examples/exported -output_type archive
```

The following example exports only documents in the database directory `/plays/`, including their collections, properties, and quality, but excluding permissions:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8000 -username user \
    -password password -mode local \
    -output_file_path /space/examples/exported -output_type archive \
    -copy_permissions false -directory_filter /plays/
```

You can use the mlcp `import` command to import an archive into a database. For details, see “Loading Content and Metadata From an Archive” on page 41.

5.4 How URI Decoding Affects Output File Names

This discussion only applies when `-output_type` is `document`.

When you export a document to a file (or to a file in a compressed file), the output file name is based on the document URI. The document URI is decoded to form the file name. For example, if the document URI is “foo%20bar.xml”, then the output file name is “foo bar.xml”.

If the document URI does not conform to the standard URI syntax of RFC 3986, decoding may fail, resulting in unexpected file names. For example, if the document URI contains unescaped special characters then the raw URI may be used.

If the document URI contains a scheme, the scheme is removed. If the URI contains both a scheme and an authority, both are removed. For example, if the document URI is “file:foo/bar.xml”, then the output file path is *output_file_path*`/foo/bar.xml`. If the document URI is “http://marklogic.com/examples/bar.xml” (contains a scheme and an authority), then the output file path is *output_file_path*`/examples/bar.xml`.

If the document URI includes directory steps, then corresponding output subdirectories are created. For example, if the document URI is “/foo/bar.xml”, then the output file path is *output_file_path*`/foo/bar.xml`.

5.5 Controlling What is Exported, Copied, or Extracted

By default, mlcp exports all documents or all documents and metadata in the database, depending on whether you are exporting in document or archive format or copying the database. Several command line options are available to enable customization. This section covers the following topics:

- Filtering Document Exports
- Filtering Archive and Copy Contents
- Understanding When Filters Are Accurate
- Example: Exporting Documents Matching a Query
- Filtering Forest Contents
- Extracting a Consistent Database Snapshot

5.5.1 Filtering Document Exports

This section covers options available for filtering what is exported by the mlcp `export` command when `-output_type` is `document`.

By default, mlcp exports all documents in the database. That is, mlcp exports the equivalent of `fn:collection()`. The following options allow you to filter what is exported. These options are mutually exclusive.

- `-directory_filter` - export only the documents in the listed database directories. You cannot use this option with `-collection_filter` or `-document-selector`.
- `-collection_filter` - export only the documents in the listed collections. You cannot use this option with `-directory_filter` or `-document_selector`.
- `-document_selector` - export only documents selected by the specified XPath expression. You cannot use this option with `-directory_filter` or `-collection_filter`. Use `-path_namespace` to define namespace prefixes.
- `-query_filter` - export only documents matched by the specified cts query. You can use this option alone or in combination with a directory, collection or document selector filter. You can only use this filter with the `export` and `copy` commands. Results may not be accurate; for details, see "Understanding When Filters Are Accurate" on page 101.

  Note: When filtering with a document selector, the XPath filtering expression should select fragment roots only. An XPath expression that selects nodes below the root is very inefficient.

When using -document_selector to filter by XPath expression, you can define namespace prefixes using the `-path_namespace` option. For example:

```
-path_namespace 'ex1,http://marklogic.com/example,ex2,http://my/ex2'
-document_selector '/ex1:elem[ex2:attr > 10]'
```

5.5.2 Filtering Archive and Copy Contents

This section covers options available for controlling what is exported by mlcp export when `-output_type` is `archive`, or what is copied by the mlcp `copy` command.

By default, all documents and metadata are exported/copied. The following options allow you to modify this behavior:

- `-directory_filter` - export/copy only the documents in the listed database directories, including related metadata. You cannot use this option with `-collection_filter` or `-document_selector`.
- `-collection_filter` - export/copy only the documents in the listed collections, including related metadata. You cannot use this options with `-directory_filter` or `-document_selector`.
- `-document_selector` - export/copy only documents selected by the specified XPath expression.You cannot use this option with `-directory_filter` or `-collection_filter`. Use `-path_namespace` to define namespace prefixes.
- `-query_filter` - export/copy only documents matched by the specified cts query. You can use this option alone or in combination with a directory, collection or document selector filter. Results may not be accurate; for details, see "Understanding When Filters Are Accurate" on page 101.

- -copy_collections - whether to include collection metadata
- -copy_permissions - whether to include permissions metadata
- -copy_properties - whether to include naked and document properties
- -copy_quality - whether to include document quality metadata
- -copy_metadata - whether to include document key-value metadata

If you set all the -copy_* options to false when exporting to an archive, the archive contains no metadata. When you import an archive with no metadata, you must set -archive_metadata_optional to true.

> **Note:** When filtering with a document selector, the XPath filtering expression should select fragment roots only. An XPath expression that selects nodes below the root is very inefficient.

When using -document_selector to filter by XPath expression, you can define namespace prefixes using the -path_namespace option. For example:

```
-path_namespace 'ex1,http://marklogic.com/example,ex2,http://my/ex2'
-document_selector '/ex1:elem[ex2:attr > 10]'
```

5.5.3 Understanding When Filters Are Accurate

When you use -directory_filter, -collection_filter, or -document_selector without -query_filter, the set of documents selected by mlcp exactly matches your filtering criteria.

The query you supply with -query_filter is used in an unfiltered search, which means there can be false positives among the selected documents. When you combine -query_filter with -directory_filter, -collection_filter, or -document_selector, mlcp might select documents that do not meet your directory, collection, or path filter criteria.

The interaction between -query_filter and the other filtering options is similar to the following. In this example, the search can match documents that are not in the “parts” collection.

```
-collection_filter parts
-query_filter yourSerializedQuery

==> selects the documents to export similar to the following:

cts:search(
  fn:collection("parts"),
  yourQuery,
  ("unfiltered"))
```

For a complete example using -query_filter, see “Example: Exporting Documents Matching a Query” on page 102.

To learn more about the implications of unfiltered searches, see Fast Pagination and Unfiltered Searches in the *Query Performance and Tuning Guide*.

5.5.4 Example: Exporting Documents Matching a Query

This example demonstrates how to use `-query_filter` to select documents for export. You can apply the same technique to filtering the source documents when copying documents from one database to another.

The `-query_filter` option accepts a serialized XML cts:query or JSON cts.query as its value. For example, the following table shows the serialization of a cts word query, prettyprinted for readability:

Format	Example

XML	`<cts:word-query xmlns:cts="http://marklogic.com/cts">`
	`  <cts:text xml:lang="en">mark</cts:text>`
	`</cts:word-query>`

JSON	`{"wordQuery":{`
	`  "text":["huck"],`
	`  "options":["lang=en"]`
	`}}`

For details on how to obtain the serialized representation of a cts query, see Serializations of cts:query Constructors in the *Search Developer's Guide*.

Using an options file is recommended when using `-query_filter` because both XML and JSON serialized queries contain quotes and other characters that have special meaning to the Unix and Windows command shells, making it challenging to properly escape the query.

For example, you can create an options file similar to the following. It should contain at least 2 lines: One for the option name and one for the serialized query. You can include other options in the file. For details, see "Options File Syntax" on page 9.

Format	Options File Contents

XML	`-query_filter`
	`<cts:word-query xmlns:cts="http://marklogic.com/cts"><cts:text`
	`xml:lang="en">mark</cts:text></cts:word-query>`

JSON	`-query_filter`
	`{"wordQuery":{"text":["huck"], "options":["lang=en"]}}`

If you save the above option in a file named “query_filter.txt”, then the following mlcp command exports files from the database that contain the word “huck”:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8000 -username user \
    -password password -mode local -output_file_path \
    /space/mlcp/export/files -options_file query_filter.txt
```

You can combine `-query_filter` with another filtering option. For example, the following command combines the query with a collection filter. The command exports only documents containing the word “huck” in the collection named “classics”:

```
$ mlcp.sh export -host localhost -port 8000 -username user \
    -password password -mode local -output_file_path \
    /space/mlcp/export/files -options_file query_filter.txt
    -collection_filter classics
```

Note: The documents selected by `-query_filter` can include false positives, including documents that do not match other filter criteria. For details, see “Understanding When Filters Are Accurate” on page 101.

The following example demonstrates generating a serialized XML `cts:and-query` or JSON `cts.andQuery` using the wrapper technique. Copy either example into Query Console, select the appropriate query type, and run it to see the output.

<table>
<tr><th>Language</th><th>Example</th></tr>
<tr><td>XQuery</td><td><pre>
xquery version "1.0-ml";
let $query := cts:and-query((
  cts:word-query("mark"),
  cts:word-query("twain")
))
let $q := xdmp:quote(
  <query>{$query}</query>/*,
  <options xmlns="xdmp:quote"><indent>no</indent></options>
)
return $q

(: Output: (whitespace added for readability)
<cts:and-query xmlns:cts="http://marklogic.com/cts">
  <cts:word-query>
    <cts:text xml:lang="en">mark</cts:text>
  </cts:word-query>
  <cts:word-query>
    <cts:text xml:lang="en">twain</cts:text>
  </cts:word-query>
</cts:and-query>
:)
</pre></td></tr>
<tr><td>Server-Side JavasScript</td><td><pre>
var wrapper =
  { query:
      cts.andQuery([
        cts.wordQuery("huck"),
        cts.wordQuery("tom")])
  };
xdmp.quote(wrapper.query.toObject())

/* Output: (whitespace added for readability)
{"andQuery":{
  "queries":[
    {"wordQuery":{"text":["huck"], "options":["lang=en"]}},
    {"wordQuery":{"text":["tom"], "options":["lang=en"]}}
  ]
}}
*/
</pre></td></tr>
</table>

Notice that in the XML example, the `xdmp:quote` "indent" option is used to disable XML prettyprinting, making the output better suited for inclusion on the mlcp command line:

```
xdmp:quote(
  <query>{$query}</query>/*,
  <options xmlns="xdmp:quote"><indent>no</indent></options>
)
```

Notice that in the JavaScript example, it is necessary to call `toObject` on the wrapped query to get the proper JSON serialization. Using `toObject` converts the value to a JavaScript object which xdmp.quote will serialize as JSON.

```
xdmp.quote(wrapper.query.toObject())
```

If you want to test your serialized query before using it with mlcp, you can round-trip your XML query with `cts:search` in XQuery or your JSON query with `cts.search` or the JSearch API in Server-Side JavaScript, as shown in the following examples.

Language	Example

XQuery	`xquery version "1.0-ml";`
	`let $wrapper :=`
	`  <query>{`
	`    cts:and-query((`
	`      cts:word-query("tom"),`
	`      cts:word-query("huck")))`
	`  }</query>`
	`let $q := xdmp:quote(`
	`  $wrapper/*,`
	`  <options xmlns="xdmp:quote"><indent>no</indent></options>)`
	`return cts:search(`
	`  fn:doc(),`
	`  cts:query(xdmp:unquote($q)/*[1])`
	`)`

Server-Side JavasScript	`var wrapper =`
	`  { query:`
	`      cts.andQuery([`
	`        cts.wordQuery("huck"),`
	`        cts.wordQuery("tom")])`
	`  };`
	`var serializedQ = xdmp.quote(wrapper.query.toObject())`
	`cts.search(`
	`  cts.query(fn.head(xdmp.unquote(serializedQ)).root))`

Note that `xdmp:unquote` returns a document node in XQuery, so you need to use XPath to address the underlying query element root node when reconstructing the query:

```
        cts:query(xdmp:unquote($q)/*[1])
```

Similarly, `xdmp.unquote` in JavaScript returns a `Sequence` on document nodes, so you must “dereference” both the iterator and the document node when reconstructing the query:

```
        cts.query(fn.head(xdmp.unquote(serializedQ)).root)
```

5.5.5 Filtering Forest Contents

This section covers options available for filtering what is extracted from from a forest when you use Direct Access. That is, when you use the mlcp `import` command with `-input_file_type forest` or the mlcp `extract` command.

By default, mlcp extracts all documents in the input forests. That is, mlcp extracts the equivalent of `fn:collection()`. The following options allow you to filter what is extracted from a forest with Direct Access. These options can be combined.

- `-type_filter`: Extract only documents with the listed content type (text, XML, or binary).
- `-directory_filter`: Extract only the documents in the listed database directories.
- `-collection_filter`: Extract only the documents in the listed collections.

For example, following combination of options extracts only XML documents in the collections named “2004” or “2005”.

```
        mlcp.sh extract -type_filter xml -collection_filter "2004,2005" ...
```

Similarly, the following options import only binary documents in the source database directory `/images/`:

```
        mlcp.sh import -input_file_type forest \
            -type_filter binary -directory_filter /images/
```

When you use Direct Access, filtering is performed in the process that reads the forest files rather than being performed by MarkLogic Server. For example, in local mode, filters are applied by mlcp on the host where you run it; in distributed mode, filters are applied by each Hadoop task that reads in forest data.

In addition, filtering cannot be applied until after a document is read from the forest. When you import or extract files from a forest file, mlcp must “touch” every document in the forest.

For details, see “Using Direct Access to Extract or Copy Documents” on page 135.

5.5.6 Extracting a Consistent Database Snapshot

By default, when you export or copy database contents, content is extracted from the source database at multiple points in time. You get whatever is in the database when mlcp accesses a given document. If the database contents are changing while the job runs, the results are not deterministic relative to the starting time of the job. For example, if a new document is inserted into the database while an export job is running, it might or might not be included in the export.

If you require a consistent snapshot of the database contents during an export or copy, use the `-snapshot` option to force all documents to be read from the database at a consistent point in time. The submission time of the job is used as the timestamp. Any changes to the database occurring after this time are not reflected in the output.

If a merge occurs while exporting or copying a consistent snapshot, and the merge eliminates a fragment that is subsequently accessed by the mlcp job, you may get an `XDMP-OLDSTAMP` error. If this occurs, the documents included in the same batch or task may not be included in the export/copy result. If the source database is on MarkLogic Server 7 or later, you may be able to work around this problem by setting the merge timestamp to retain fragments for a time period longer than the expected running time of the job; for details, see Understanding and Controlling Database Merges in the *Administrator's Guide*.

5.6 Redacting Content During Export or Copy Operations

Redaction is the process of eliminating or obscuring portions of a document when retrieving the document from MarkLogic. For example, you can eliminate or mask sensitive personal information such as credit card numbers, phone numbers, or email addresses from documents. You can only redact document content, not document properties.

> **Note:** Advanced Security License option is required when using redaction.

Redaction support in MarkLogic is covered in detail in Redacting Document Content in the *Application Developer's Guide*. This section describes how to use mlcp as the redaction driver. This section includes the following topics:

- Basic Steps for Redacting Documents
- Example: Using mlcp for Redaction

5.6.1 Basic Steps for Redacting Documents

Use the `-redaction` option of mlcp to apply redaction rules to an export or copy operation. This option accepts a comma-separated list of redaction rule collection URIs. For example:

```
-redaction "pii-rules,sec-rules"
```

Before you can use redaction, you must install one or more redaction rule sets in the Schemas database. For details on defining and installing redaction rules, see Redacting Document Content in the *Application Developer's Guide*.

Preparing to redact documents with mlcp requires the following steps. For a complete example, see “Example: Using mlcp for Redaction” on page 108.

1. Install one or more redaction rules in the Schemas database. Each rule must be part of at least one collection. For details, see Defining Redaction Rules and Installing Redaction Rules in the *Application Developer's Guide*.

2. If you create a rule that uses a user-defined redaction function, install the implementation of your redaction function in the modules database associated with the App Server you will connect to using mlcp. For details, see User-Defined Redaction Functions in the *Application Developer's Guide*.

3. Add the `-redaction` option to your mlcp command line. For example, the following command applies the rules in the collections “pii-rules” and “sec-rules” to all exported documents.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8000 -username user \
    -password password -mode local -output_file_path \
    /space/mlcp/export/files -directory_filter /people/ \
    -redaction "pii-rules,sec-rules"
```

The `-redaction` option works similarly for copy operations. For details, see “Redacting Content During a Copy” on page 124.

The user who extracts redacted documents must have read permissions on the source documents and the rules, but need not be able to modify the rule collection or rule definitions. For details, see Security Considerations in *Application Developer's Guide*.

The following behaviors apply when exceptional conditions occur. You should be aware of these behaviors so you understand when content might not be redacted as expected:

- If a rule collection is empty, mlcp issues a warning and continues with the job.
- If any of the rules contain errors, an error is reported and mlcp aborts the export or copy operation.
- If a rule is valid, but an error occurs when applying the rule, the rule is skipped for the current document and a warning is logged. The job continues.

5.6.2 Example: Using mlcp for Redaction

This example walks you through using mlcp to install and apply redaction rules based on the built-in redaction functions. For a similar example using XQuery and Query console, see Example: Getting Started With Redaction in the *Application Developer's Guide*.

The example has the following parts:

- Creating a Work Area
- Installing the Source Documents
- Installing the Redaction Rules
- Understanding the Example Rules
- Applying the Redaction Rules

This example uses rules based on built-in redaction functions. For an example of using user-defined redaction functions, see User-Defined Redaction Functions in the *Application Developer's Guide*.

5.6.2.1 Creating a Work Area

This example assumes the following directory hierarchy:

```
redact-gs/
  data/
  rules/
```

The `data/` directory will hold the source documents. The `rules/` directory will hold redaction rules. The example walks you through populating these directories and uploading the contents to MarkLogic using mlcp in preparation for exporting a set of redacted documents with mlcp.

Create the required directories on Linux by running the following command in a location of your choosing:

```
$ mkdir -p redact-gs/data redact-gs/rules
```

Create the required directories on Windows by running the following command in a location of your choice:

```
>mkdir redact-gs\data redact-gs\rules
```

5.6.2.2 Installing the Source Documents

When you complete this exercise, the Documents database should contain the following documents. The documents are inserted into a collection named "gs-samples" for easy reference.

- `/redact-gs/sample1.xml`
- `/redact-gs/sample2.json`

Follow the steps in this procedure to install two sample documents in the Documents database.

1. Change directory to the data directory you created in "Creating a Work Area" on page 109. You should be in your `redact-gs/data` directory.

2. Copy the following text into a file named “sample1.xml”:

```
<personal>
  <name>Little Bopeep</name>
  <summary>Seeking lost sheep. Please call 123-456-7890.</summary>
  <id>12-3456789</id>
</personal>
```

3. Copy the following text into a file name “sample2.json”:

```
{"personal": {
  "name": "Jack Sprat",
  "summary": "Free nutrition advice! Call (234)567-8901 now!",
  "id": "45-6789123"
}}
```

4. Run the following mlcp command to insert the sample documents into the Documents database. Modify the connection details as needed to match your environment.

```
$ mlcp.sh import -host localhost -port 8000 \
    -username user -password password -mode local \
    -input_file_path . \
    -output_uri_replace ".*/redact-gs/data/,'/redact-gs/'" \
    -output_collections "gs-samples"
```

You can use Query Console to explore the Documents database and confirm the upload.

The use of `-output_uri_replace` on the import command line replaces the portion of the default URI that is based on the filesystem location with the fixed directory prefix “/rules/gs”. For more details, see “Controlling Database URIs During Ingestion” on page 33.

5.6.2.3 Installing the Redaction Rules

Rules must be installed in the schemas database associated with your content database. Rules must also be part of a collection before you can use them. This section installs rules in the Schemas database, which is the default schemas database associated with the Documents database.

When you complete this exercise, the Schemas database should contain the following documents. The documents are inserted into a rule collection named “gs-rules”. Rules must be in a rule collection before you can apply them.

- `/rules/gs/redact-phone.xml`
- `/rules/gs/conceal-id.json`

The rules installed in this step use the redact-us-phone and conceal built-in redaction functions. For details on these and other built-in redaction functions, see Built-in Redaction Function Reference in the *Application Developer’s Guide*.

Follow the steps in this procedure to install two sample rules in the Schemas database. For an explanation of what the rules do, see “Understanding the Example Rules” on page 111.

1. Change directory to the `rules` directory you created in “Creating a Work Area” on page 109. You should be in your `redact-gs/rules` directory.

2. Copy the following text into a file named “redact-phone.xml”.

```
<rule xml:lang="zxx" xmlns="http://marklogic.com/xdmp/redaction">
  <description>Obscure phone numbers.</description>
  <path>//summary</path>
  <method>
    <function>redact-us-phone</function>
  </method>
  <options>
    <level>partial</level>
  </options>
</rule>
```

3. Copy the following text into a file name “conceal-id.json”:

```
{ "rule": {
    "description": "Remove customer ids.",
    "path": "//id",
    "method": { "function": "conceal" }
}}
```

4. Run the following mlcp command to insert the rules into the Schemas database. Modify the connection details as needed to match your environment.

```
$ mlcp.sh import -host localhost -port 8000 \
    -username user -password password -mode local \
    -database Schemas -input_file_path . \
    -output_uri_replace ".*/redact-gs/rules/,'/rules/gs/'" \
    -output_collections "gs-rules"
```

You can use Query Console to explore the Schemas database and confirm the upload.

The use of `-output_uri_replace` on the import command line replaces the portion of the default URI that is based on the filesystem location with the fixed directory prefix “/rules/gs”. For more details, see “Controlling Database URIs During Ingestion” on page 33.

5.6.2.4 Understanding the Example Rules

The XML rule installed in “Installing the Redaction Rules” on page 110 has the following form:

```
<rule xml:lang="zxx" xmlns="http://marklogic.com/xdmp/redaction">
  <description>Obscure phone numbers.</description>
  <path>//summary</path>
  <method>
```

```
      <function>redact-us-phone</function>
    </method>
    <options>
      <level>partial</level>
    </options>
  </rule>
```

The rule elements have the following effect:

- `description` - Optional metadata for informational purposes.
- `path` - Apply the redaction function specified by the rule to nodes selected by the path expression "//summary".
- `method` - Use the built-in redaction function `redact-us-phone` to redact the value in a `summary` XML element or JSON property. By default, this function replaces all digits in a phone number by the character "#". You can tell this is a built-in function because `method` has no `module` child.
- `options` - Pass a `level` parameter value of "partial" to redact-us-phone, causing the function to leave the last 4 digits of the value unchanged.

The expected result of applying this rule is that any text in the value of a node named "summary" that matches the pattern of a US phone number will be replaced. The replacement value uses the "#" number to replace all but the last 4 digits. For example, a value such as 123-456-7890 is redacted to ###-###-7890. For more details, see redact-us-phone in the *Application Developer's Guide*.

The JSON rule installed in "Installing the Redaction Rules" on page 110 has the following form:

```
{ "rule": {
    "description": "Remove customer ids.",
    "path": "//id",
    "method": { "function": "conceal" }
}}
```

The rule properties have the following effect:

- `description` - Optional metadata for informational purposes.
- `path` - Apply the redaction function specified by the rule to nodes selected by the path expression `//id`.
- `method` - Use the built-in redaction function `conceal` to redact the `id` XML element or JSON property. This function will hide the nodes selected by `path`. You can tell this is a built-in function because `method` has no `module` child.

The expected result of applying this rule is to remove nodes named `id`. For example, if `//id` selects and XML element or JSON property, the element or property does not appear in the redacted output. Note that, if `//id` selects array items in JSON, the items are eliminated, but the `id` property might remain, depending on the structure of the document. For more details, see conceal in the *Application Developer's Guide*.

5.6.2.5 Applying the Redaction Rules

Run the following command from your `redact-gs/` directory to export redacted versions of the sample documents. Modify the connection details as needed to match your environment. A collection filter (`-collection_filter "gs-samples"`) is used to select the documents for redaction and export.

```
$ mlcp.sh export -host localhost -port 8000 \
    -username user -password password -mode local \
    -collection_filter "gs-samples" \
    -output_file_path ./output/ \
    -redaction "gs-rules"
```

Running the export command saves the redacted documents to an `output/` sub-directory. You should have the following filesystem hierarch. The "extra" redact-gs sub-directory is created by mlcp because the document URIs are of the form `/redact-s/`*`filename`*.

```
redact-gs/
  output/
    redact-gs/
      sample1.xml
      sample2.json
```

The following table shows the result of redacting the XML sample document. Notice that the telephone number in the `summary` element has been partially redacted by the `redact-us-phone` function. Also, the `id` element has been completely hidden by the `conceal` function. The affected parts of the content are highlighted in the table.

Stage	XML Content

Original Document	<personal>
	  <name>Little Bopeep</name>
	  <summary>Seeking lost sheep. Please call **123-456-7890**.</summary>
	  <id>12-3456789</id>
	</personal>

Redacted Result	<personal>
	  <name>Little Bopeep</name>
	  <summary>Seeking lost sheep. Please call **###-###-7890**.</summary>
	</personal>

The following table shows the result of redacting the JSON sample document. Notice that the telephone number in the `summary` property has been partially redacted by the `redact-us-phone` function. Also, the `id` property has been completely hidden by the `conceal` function. The affected parts of the content are highlighted in the table.

Stage	JSON Content
Original Document	{"personal": { "name": "Jack Sprat", "summary": "Free nutrition advice! Call **(234)567-8901** now!", **"id":** "45-6789123" }}

Redacted Result	{"personal": {
	"name": "Jack Sprat",
	"summary": "Free nutrition advice! Call **(###)###-8901** now!"
	}}

To redact documents when copying them between databases rather than exporting them, add the `-redaction` option to the mlcp copy command line.

5.7 Advanced Document Selection and Transformation

The mlcp tool uses the MarkLogic Connector for Hadoop to distribute work across your MarkLogic cluster, even when run in local mode. When you use the mlcp export command, MarkLogic Server acts as an input source for a Hadoop MapReduce job. The exported documents are the output of the job. You can take low level control of the job by setting connector and Hadoop configuration properties.

> **Note:** Setting low level configuration properties is an advanced technique. You should understand how to use the MarkLogic Connector for Hadoop before attempting this. For details, see Advanced Input Mode in the *MarkLogic Connector for Hadoop Developer's Guide*.

The following list describes some use cases in which you might choose to set low level configuration properties:

- The options discussed in "Controlling What is Exported, Copied, or Extracted" on page 99 do not enable you to select the desired set of documents to export.
- You want to apply a server-side transformationt to each document as it is read from the database.

Similar use cases and techniques apply to copy operations. For details, see "Advanced Document Selection for Copy" on page 124.

The following table lists some connector and Hadoop configuration properties relevant to advanced configuration for export.

Configuration Property	Description
`mapreduce.marklogic.input.mode`	Controls whether the connectors runs in basic or advanced mode. Set to "advanced".
`mapreduce.marklogic.input.splitquery`	A query that generates input splits. This distributes work across export tasks. The query can be either XQuery or Server-Side JavaScript. For details, see Creating Input Splits in the *MarkLogic Connector for Hadoop Developer's Guide*.
`mapreduce.marklogic.input.query`	A query that selects the input fragments to export. You can use the input query to apply server-side transformations to each output item. The query can be either XQuery or Server-Side JavaScript. For details, see Creating Input Key-Value Pairs in the *MarkLogic Connector for Hadoop Developer's Guide*.
`mapreduce.job.inputformat.class`	Optional. You do not need to set this property unless your input query produces something other than documents. This property identifies a subclass of the connector `InputFormat` class, describing the "type" of the values produced by the input query. You can create your own `InputFormat` subclass, but most applications will use one of the classes defined by the connector, such as `DocumentInputFormat`, which is the default used by mlcp. For details, see InputFormat Subclasses in the *MarkLogic Connector for Hadoop Developer's Guide*.

You can pass a connector configuration file through mlcp with the `-conf` option. The `-conf` option must appear after `-options_file` (if present) and before any other mlcp options. The following example command demonstrates the `-conf` option.

```
$ mlcp.sh export -conf conf.xml -host localhost -port 8000 \
    -username user -password password -mode local \
    -output_file_path /space/examples/exported \
    -directory_filter /binaies/
```

The following example connector configuration file uses an XQuery split query (`mapreduce.marklogic.input.splitquery`) to distribute the documents across export tasks, and an XQuery transformation query (`mapreduce.marklogic.input.query`) that returns just the first 1000 bytes of each selected binary document.

```
<property>
  <name>mapreduce.marklogic.input.query</name>
  <value><![CDATA[
    xquery version "1.0-ml";
    declare namespace mlmr="http://marklogic.com/hadoop";
    declare variable $mlmr:splitstart as xs:integer external;
    declare variable $mlmr:splitend as xs:integer external;
    for $doc in fn:doc()[$mlmr:splitstart to $mlmr:splitend]
    return xdmp:subbinary($doc/binary(), 1, 1000)
  ]]></value>
</property>
<property>
  <name>mapreduce.marklogic.input.splitquery</name>
  <value><![CDATA[
    xquery version "1.0-ml";
    import module namespace hadoop = "http://marklogic.com/xdmp/hadoop"
      at "/MarkLogic/hadoop.xqy";
    hadoop:get-splits('', 'fn:doc()', '()')
  ]]></value>
</property>
<property>
  <name>mapreduce.marklogic.input.mode</name>
  <value>advanced</value>
</property>
```

For more details and examples, see the *MarkLogic Connector for Hadoop Developer's Guide*.

5.8 Export Command Line Options

This section summarizes the command line options available with the mlcp `export` command. The following command line options define your connection to MarkLogic:

Option	Description
`-host` *`comma-list`*	REquired. A comma separated list of hosts through which mlcp can connect to the destination MarkLogic Server. You must specify at least one host. For more details, see "How mlcp Uses the Host List" on page 83.
`-port` *`number`*	Port number of the source MarkLogic Server. There should be an XDBC App Server on this port. Default: 8000.
`-username` *`string`*	MarkLogic Server user from which to export documents. Required, unless using Kerberos authentication.
`-password` *`string`*	Password for the MarkLogic Server user specified with `-username`. Required, unless using Kerberos authentication.

The following table lists command line options that define the characteristics of the `export` operation:

Option	Description
`-collection_filter` *`comma-list`*	A comma-separated list of collection URIs. mlcp exports only documents in these collections, plus related metadata. This option may not be combined with `-directory_filter` or `-document_selector`. Default: All documents and related metadata.
`-compress` *`boolean`*	Whether or not to compress the output document. Only applicable when `-output_type` is `document`. Default: `false`.

Option	Description
`-conf filename`	Pass extra setting to Hadoop when using distributed mode. For details, see "Setting Custom Hadoop Options and Properties" on page 16. This option must appear before mlcp-specific options.
`-content_encoding string`	The character encoding of output documents when `-input_file_type` is `documents`. The option value must be a character set name accepted by your JVM; see `java.nio.charset.Charset`. Default: `UTF-8`. Set to `system` to use the platform default encoding for the host on which mlcp runs.
`-copy_collections boolean`	When exporting documents to an archive, whether or not to copy collections to the destination. Default: `true`.
`-copy_metadata boolean`	When exporting documents to an archive, whether or not to copy key-value metadata to the destination. Default: `true`.
`-copy_permissions boolean`	When exporting documents to an archive, whether or not to copy document permissions to the destination. Default: `true`.
`-copy_properties boolean`	When exporting documents to an archive, whether or not to copy properties to the destination. Default: `true`.
`-copy_quality boolean`	When exporting documents to an archive, whether or not to copy document quality to the destination. Default: `true`.
`-D property=value`	Pass a configuration property setting to Hadoop when using distributed mode. For details, see "Setting Custom Hadoop Options and Properties" on page 16. This option must appear before mlcp-specific options.
`-database string`	The name of the source database. Default: The database associated with the source App Server identified by `-host` and `-port`.

Option	Description
`-directory_filter` *`comma-list`*	A comma-separated list of database directory names. mlcp exports only documents from these directories, plus related metadata. Directory names should usually end with "/". This option may not be combined with `-collection_filter` or `-document_selector`. Default: All documents and related metadata.
`-document_selector` *`string`*	Specifies an XPath expression used to select which documents are exported from the database. The XPath expression should select fragment roots. This option may not be combined with `-directory_filter` or `-collection_filter`. Default: All documents and related metadata.
`-hadoop_conf_dir` *`string`*	When using distributed mode, the Hadoop config directory. For details, see "Configuring Distributed Mode" on page 15.
`-indented` *`boolean`*	Whether to pretty-print XML output. Default: `false`.
`-max_split_size` *`number`*	The maximum number of document fragments processed per split. Default: 20000 in local mode, 50000 in distributed mode.
`-mode` *`string`*	Export mode. Accepted values: `distributed`, `local`. Distributed mode requires Hadoop. Default: `local`, unless you set the `HADOOP_CONF_DIR` variable; for details, see "Configuring Distributed Mode" on page 15.
`-options_file` *`string`*	Specify an options file pathname from which to read additional command line options. If you use an options file, this option must appear first. For details, see "Options File Syntax" on page 9.
`-output_file_path` *`string`*	Destination directory where the archive or documents are saved. The directory must not already exist.
`-output_type` *`string`*	The type of output to produce. Accepted values: `document`, `archive`. Default: `document`.

Option	Description
`-path_namespace` *`comma-list`*	Specifies one or more namespace prefix bindings for namespace prefixes usable in path expressions passed to `-document_selector`. The list items should be alternating pairs of prefix names and namespace URIs, such as `'pfx1,http://my/ns1,pfx2,http://my/ns2'`.
`-query_filter` *`string`*	Specifies a query to apply when selecting documents for export. The argument must be the XML serialization of a cts:query or JSON serialization of a cts.query. Only documents matching the query are considered for export; false positives are possible. For details, see “Controlling What is Exported, Copied, or Extracted” on page 99.
`-redaction` *`comma-list`*	Apply one or more redaction rule collections. The argument must be a comma-separated list of rule collection URIs. The rule collections must be installed in the schemas database. For details and example, see “Redacting Content During Export or Copy Operations” on page 107 and Redacting Document Content in the *Application Developer’s Guide*.
`-restrict_hosts` *`boolean`*	Restrict mlcp to connect to MarkLogic only through the hosts listed in the `-host` option. Default: false (no restriction). For more details, see “Restricting the Hosts mlcp Uses to Connect to MarkLogic” on page 83.
`-snapshot` *`boolean`*	Whether or not to export a consistent point-in-time snapshot of the database contents. Default: `false`. When `true`, the job submission time is used as the database read timestamp for selecting documents to export. For details, see “Extracting a Consistent Database Snapshot” on page 107.

Option	Description
`-ssl` *`boolean`*	Enable/disable SSL secured communication with MarkLogic. Default: false. If you set this option to true, your App Server must be SSL enabled. For details, see "Connecting to MarkLogic Using SSL" on page 17.
`-thread_count` *`number`*	The number of threads to spawn for concurrent exporting. The total number of threads spawned by the process can be larger than this number, but this option caps the number of concurrent sessions with MarkLogic Server. Only available in `local` mode. Default: 4.

6.0 Copying Content Between Databases

Use the mlcp `copy` command to copy content and associated metadata from one MarkLogic Server database to another when both are reachable on the network. You can also copy data from offline forests to a MarkLogic Server database; for details, see “Using Direct Access to Extract or Copy Documents” on page 135.

This chapter includes the following topics:

6.1 Basic Steps

To copy one database to another with mclp:

1. Set `-input_host`, `-input_port`, `-input_username`, and `-input_password` to identify the source MarkLogic Server instance and user.

2. Set `-output_host`, `-output_port`, `-output_username`, and `-output_password` to identify the destination MarkLogic Server instance and user.

3. Select what documents to copy. For details, see “Filtering Archive and Copy Contents” on page 100.

   - To select documents in one or more collections, set `-collection_filter` to a comma separated list of collection URIs.
   - To select documents in one or more database directories, set `-directory_filter` to a comma separated list of directory URIs.
   - To select documents matching an XPath expression, use `-document_selector`. To use namespace prefixes in the XPath expression, define the prefix binding using `-path_namespace`.
   - To select document matching a query, use `-query_filter`. You can use this option alone or in combination with a directory, collection or document selector filter. False positives are possible; for details, see “Understanding When Filters Are Accurate” on page 101.
   - To select all documents in the database, leave `-collection_filter`, `-directory_filter`, `-document_selector`, and `-query_filter` unset.

4. If you want to exclude some or all source document metadata:

- Set -copy_collections to false to exclude document collections metadata.
- Set -copy_permissions to false to exclude document permissions metadata.
- Set -copy_properties to false to exclude document properties.
- Set -copy_quality to false to exclude document quality metadata.
- Set -copy_metadata to false to exclude document key-value metadata.

5. If you want to add or override document metadata in the destination database:

   - Set -output_collections to add destination documents to a collection.
   - Set -output_permissions to add permissions to destination documents.
   - Set -output_quality to set the quality of destination documents.

6. If you want the destination documents to have database URIs different from the source URIs, set -output_uri_replace, -output_uri_prefix, and/or -output_uri_suffix. For details, see "Controlling Database URIs During Ingestion" on page 33.

For a complete list of mlcp copy command options, see "Copy Command Line Options" on page 128.

6.2 Examples

The following example copies all documents and their metadata from the source database to the destination database:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh copy -mode local -input_host srchost -input_port 8000 \
    -input_username user1 -input_password password1 \
    -output_host desthost -output_port 8010 -output_username user2 \
    -output_password password2
```

The following example copies selected documents, excluding the source permissions and adding the documents to 2 new collections in the destination database:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh copy -mode local -input_host srchost -input_port 8000 \
    -input_username user1 -input_password password1 \
    -output_host desthost -output_port 8000 -output_username user2 \
    -output_password password2 -copy_permissions false \
    -output_collections shakespeare,plays
```

For an example of using -query_filter, see "Example: Exporting Documents Matching a Query" on page 102.

6.3 Redacting Content During a Copy

Redaction is the process of eliminating or obscuring portions of a document when retrieving the document from MarkLogic. For example, you can eliminate or mask sensitive personal information such as credit card numbers, phone numbers, or email addresses from documents. You can only redact document content, not document properties.

Redaction is performed as documents are read from the source database. For example, if you copy documents between databases in two different MarkLogic installations, the unredacted content never leaves the source installation.

Redaction support in MarkLogic is covered in detail in “Redacting Content During Export or Copy Operations” on page 107 and Redacting Document Content in the *Application Developer’s Guide*.

Use the `-redaction` option to apply redaction rules during a copy. For example, the following command copies documents in the “my_docs” collection from one database to another, and applies the redaction rules in the rule collections “hipaa-rules and “biz-rules” to the source documents before copying them to the destination database.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh copy -mode local -input_host srchost -input_port 8000 \
    -input_username user1 -input_password password1 \
    -output_host desthost -output_port 8000 -output_username user2 \
    -output_password password2 -collection_filter my_docs \
    -redaction "hipaa-rules,biz-rules"
```

For more details, see “Redacting Content During Export or Copy Operations” on page 107.

6.4 Advanced Document Selection for Copy

The mlcp tool uses the MarkLogic Connector for Hadoop to distribute work across your MarkLogic cluster, even when run in local mode. When you use the mlcp copy command, the source MarkLogic Server instance acts as an input source for a Hadoop MapReduce job. Similarly, the destination MarkLogic Server instance acts as the output sink for the job. You can take low level control of the job by setting connector and Hadoop configuration properties.

> **Note:** This is an advanced technique. You should understand how to use the MarkLogic Connector for Hadoop before attempting this. For details, see Advanced Input Mode in the *MarkLogic Connector for Hadoop Developer’s Guide*.

The following list describes some use cases in which you might choose to set low level configuration properties:

- The options discussed in “Controlling What is Exported, Copied, or Extracted” on page 99 do not enable you to select the desired set of documents to copy.

Similar use cases and techniques apply to export operations. For details, see “Advanced Document Selection and Transformation” on page 114.

The following table lists some connector and Hadoop configuration properties relevant to advanced configuration for copy.

Configuration Property	Description
`mapreduce.marklogic.input.mode`	Controls whether the connectors runs in basic or advanced mode. Set to "advanced".
`mapreduce.marklogic.input.splitquery`	A query that generates input splits. This distributes the work required to extract documents from the source database. The query can be either XQuery or Server-Side JavaScript. For details, see Creating Input Splits in the *MarkLogic Connector for Hadoop Developer's Guide*.
`mapreduce.marklogic.input.query`	A query that selects the input fragments to extract from the source database. You can use the input query to apply server-side transformations to each output item. The query can be either XQuery or Server-Side JavaScript. For details, see Creating Input Key-Value Pairs in the *MarkLogic Connector for Hadoop Developer's Guide*.
`mapreduce.inputformat.class`	This property identifies a subclass of the connector `InputFormat` class, describing the "type" of the values produced by your input query. You can create your own `InputFormat` subclass, but most applications will use one of the classes defined by the connector, such as `DocumentInputFormat`, which is the default used by mlcp. For details, see InputFormat Subclasses in the *MarkLogic Connector for Hadoop Developer's Guide*.

Configuration Property	Description
`mapreduce.outputformat.class`	This property identifies a subclass of the connector `OutputFormat` class, describing the "type" of input for the destination database. In most cases, you should use `ContentOutputFormat`. For details, see OutputFormat Subclasses in the *MarkLogic Connector for Hadoop Developer's Guide*.
`mapreduce.map.class`	Optional. This property identifies a subclass of `org.apache.hadoop.mapreduce.Mapper`. Defaults to `com.marklogic.contentpump.DocumentMapper`, but you override if for more advanced use cases. For details, see Defining the Map Function in the *MarkLogic Connector for Hadoop Developer's Guide*.

When you take low-level control of a copy operation, you can no longer use options such as `-copy_collections`, `-copy_permissions`, and `-copy_properties` to copy the various categories of metadata from the source database to the destination database. If you include the `-copy_*` options on the mlcp command line, they will be ignored.

You can pass a connector configuration file through mlcp with the `-conf` option. The `-conf` option must appear after `-options_file` (if present) and before any other mlcp options. The following example command demonstrates using the `-conf` option in a copy operation.

```
$ mlcp.sh copy -conf conf.xml -input_host srchost -input_port 8000 \
    -input_username user -input_password password \
    -output_host desthost -output_port 8000 \
    -output_username user -output_password password \
    -mode local
```

The following example connector configuration file includes an XQuery split query that selects documents from a specific collection (similar to what the -collection_filter option does), and an XQuery input query that selects specific elements of each document.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>mapreduce.job.inputformat.class</name>
    <value>com.marklogic.mapreduce.DocumentInputFormat</value>
  </property>
  <property>
    <name>mapreduce.job.outputformat.class</name>
    <value>com.marklogic.mapreduce.ContentOutputFormat</value>
  </property>
  <property>
    <name>mapreduce.marklogic.input.mode</name>
    <value>advanced</value>
  </property>
  <property>
    <name>mapreduce.marklogic.input.splitquery</name>
    <value><![CDATA[
xquery version "1.0-ml";
declare namespace wp="http://www.mediawiki.org/xml/export-0.4/";
import module namespace admin ="http://marklogic.com/xdmp/admin"
  at "/MarkLogic/admin.xqy";
let $conf := admin:get-configuration()
for $forest in xdmp:database-forests(xdmp:database())
let $host_id :=admin:forest-get-host($conf,$forest)
let $host_name := admin:host-get-name($conf,$host_id)
let $cnt := xdmp:estimate(
  cts:search(fn:collection("mycoll"),
             cts:and-query(()),(),0.0,$forest))
return
($forest,$cnt,$host_name)
    ]]></value>
  </property>
  <property>
    <name>mapreduce.marklogic.input.query</name>
    <value><![CDATA[
xquery version "1.0-ml";
declare default element namespace "http://HadoopTest";
fn:collection("mycoll")//*:bar/*:foo
    ]]></value>
  </property>
</configuration>
```

For more details and examples, see the *MarkLogic Connector for Hadoop Developer’s Guide*.

6.5 Copy Command Line Options

This section summarizes the command line options available with the mlcp `copy` command. The following command line options define your connection to MarkLogic:

Option	Description
`-input_host` *`string`*	Required. A comma separated list of hosts through which mlcp can connect to the source database. You must specify at least one host. For more details, see "How mlcp Uses the Host List" on page 83.
`-input_port` *`number`*	Port number of the source MarkLogic Server. There should be an XDBC App Server on this port. Default: 8000.
`-input_username` *`string`*	MarkLogic Server user with which to export documents. Required, unless using Kerberos authentication.
`-input_password` *`string`*	Password for the MarkLogic Server user specified with `-input_username`. Required, unless using Kerberos authentication.
`-output_host` *`string`*	Required. A comma separated list of hosts through which mlcp can connect to the destination database. You must specify at least one host. For more details, see "How mlcp Uses the Host List" on page 83.
`-output_port` *`number`*	Port number of the destination MarkLogic Server. There should be an XDBC App Server on this port. Default: 8000.
`-output_username` *`string`*	MarkLogic Server user with which to import documents to the destination. Required, unless using Kerberos authentication.
`-output_password` *`string`*	Password for the MarkLogic Server user specified with `-output_username`. Required, unless using Kerberos authentication.

The following table lists command line options that define the characteristics of the copy operation:

Option	Description
`-batch_size` *`number`*	The number of documents to load per request to MarkLogic Server. Default: 100. Maximum: 200.
`-collection_filter` *`comma-list`*	A comma-separated list of collection URIs. mlcp exports only documents in these collections, plus related metadata. This option may not be combined with `-directory_filter`. Default: All documents and related metadata.
`-conf` *`filename`*	Pass extra setting to Hadoop when using distributed mode. For details, see "Setting Custom Hadoop Options and Properties" on page 16. This option must appear before mlcp-specific options.
`-copy_collections` *`boolean`*	Whether to copy document collections from the source database to the destination database. Default: `true`.
`-copy_metadata` *`boolean`*	Whether to copy document key-value metadata from the source database to the destination database. Default: `true`.
`-copy_permissions` *`boolean`*	Whether to copy document permissions from the source database to the destination database. Default: `true`.
`-copy_properties` *`boolean`*	Whether to copy document properties from the source database to the destination database. Default: `true`.
`-copy_quality` *`boolean`*	Whether to copy document quality from the source database to the destination database. Default: `true`.
`-D` *`property=value`*	Pass a configuration property setting to Hadoop when using distributed mode. For details, see "Setting Custom Hadoop Options and Properties" on page 16. This option must appear before mlcp-specific options.

Option	Description
`-directory_filter` *`comma-list`*	A comma-separated list of database directories. mlcp exports only documents from these directories, plus related metadata. Directory names should usually end with “/”. This option may not be combined with `-collection_filter`. Default: All documents and related metadata.
`-document_selector` *`string`*	Specifies an XPath expression used to select which documents are extracted from the source database. The XPath expression should select fragment roots. This option may not be combined with `-directory_filter` or `-collection_filter`. Default: All documents and related metadata.
`-fastload` *`boolean`*	Whether or not to force optimal performance, even at the risk of creating duplicate document URIs. See “Time vs. Correctness: Understanding -fastload Tradeoffs” on page 59. Default: `false`.
`-hadoop_conf_dir` *`string`*	When using distributed mode, the Hadoop config directory. For details, see “Configuring Distributed Mode” on page 15.
`-input_database` *`string`*	The name of the source database. Default: The database associated with the source App Server identified by `-input_host` and `-input_port`.
`-input_ssl` *`boolean`*	Enable/disable SSL secured communication with the input App Server. Default: false. If you set this option to true, your App Server must be SSL enabled. For details, see “Connecting to MarkLogic Using SSL” on page 17.
`-max_split_size` *`number`*	The maximum number of document fragments processed per split. Default: 50000.
`-mode` *`string`*	Copy mode. Accepted values: `distributed`, `local`. Distributed mode requires Hadoop. Default: `local`, unless you set the `HADOOP_CONF_DIR` variable; for details, see “Configuring Distributed Mode” on page 15.

Option	Description
`-path_namespace` *`comma-list`*	Specifies one or more namespace prefix bindings for namespace prefixes usable in path expressions passed to `-document_selector`. The list items should be alternating pairs of prefix names and namespace URIs, such as `'pfx1,http://my/ns1,pfx2,http://my/ns2'`.
`-options_file` *`string`*	Specify an options file pathname from which to read additional command line options. If you use an options file, this option must appear first. For details, see "Options File Syntax" on page 9.
`-output_collections` *`comma-list`*	A comma separated list of collection URIs. Output documents are added to these collections.
`-output_database` *`string`*	The name of the destination database. Default: The database associated with the destination App Server identified by `-output_host` and `-output_port`.
`-output_permissions` *`comma-list`*	A comma separated list of (`role,capability)` pairs to apply to loaded documents. Default: The default permissions associated with the user inserting the document. Example: `-output_permissions role1,read,role2,update`
`-output_quality` *`string`*	The quality to assign to output documents.
`-output_partition` *`string`*	The name of the database partition in which to create documents. Required when using range assignment policy. For details, see "How Assignment Policy Affects Optimization" on page 61 and Range Partitions in the *Administrator's Guide*.
`-output_ssl` *`boolean`*	Enable/disable SSL secured communication with the output App Server. Default: false. If you set this option to true, your App Server must be SSL enabled. For details, see "Connecting to MarkLogic Using SSL" on page 17.
`-output_uri_prefix` *`string`*	Specify a prefix to prepend to the default URI. Used to construct output document URIs. For details, see "Controlling Database URIs During Ingestion" on page 33.

Option	Description
-output_uri_replace *comma-list*	A comma separated list of (regex,string) pairs that define string replacements to apply to the URIs of documents added to the database. The replacement strings must be enclosed in single quotes. For example, -output_uri_replace "regex1,'string1',regext2,'string2'"
-output_uri_suffix *string*	Specify a suffix to append to the default URI Used to construct output document URIs. For details, see "Controlling Database URIs During Ingestion" on page 33.
-query_filter *string*	Specifies a query to apply when selecting documents to be copied. The argument must be the XML serialization of a cts:query or JSON serialization of a cts.query. Only documents in the source database that match the query are considered for copying. For details, see "Controlling What is Exported, Copied, or Extracted" on page 99. False postives are possible; for details, see "Understanding When Filters Are Accurate" on page 101.
-redaction *comma-list*	Apply one or more redaction rule collections. The argument must be a comma-separated list of rule collection URIs. The rule collections must be installed in the schemas database on the source MarkLogic installation. For details and example, see "Redacting Content During Export or Copy Operations" on page 107 and Redacting Document Content in the *Application Developer's Guide*.
-restrict_input_hosts *boolean*	Restrict mlcp to connect to the source database only through the hosts listed in the -input_host option. Default: false (no restriction). For more details, see "Restricting the Hosts mlcp Uses to Connect to MarkLogic" on page 83.
-restrict_output_hosts *boolean*	Restrict mlcp to connect to the destination database only through the hosts listed in the -output_host option. Default: false (no restriction). For more details, see "Restricting the Hosts mlcp Uses to Connect to MarkLogic" on page 83.

Option	Description
`-snapshot` *`boolean`*	Whether or not to use a consistent point-in-time snapshot of the source database contents. Default: `false`. When `true`, the job submission time is used as the database read timestamp for selecting documents to export. For details, see "Extracting a Consistent Database Snapshot" on page 107.
`-temporal_collection` *`string`*	A temporal collection into which the documents are to be loaded in the destination database. For details on loading temporal documents into MarkLogic, see Using MarkLogic Content Pump (MLCP) to Load Temporal Documents in the *Temporal Developer's Guide*.
`-thread_count` *`number`*	The number of threads to spawn for concurrent copying. The total number of threads spawned by the process can be larger than this number, but this option caps the number of concurrent sessions with MarkLogic Server. Only available in `local` mode. Default: 4.
`-transform_function` *`string`*	The localname of a custom content transformation function installed on MarkLogic Server. Ignored if `-transform_module` is not specified. Default: `transform`. For details, see "Transforming Content During Ingestion" on page 66.

Option	Description
`-transform_module` *`string`*	The path in the modules database or modules directory of a custom content transformation function installed on MarkLogic Server. This option is required to enable a custom transformation. For details, see “Transforming Content During Ingestion” on page 66.
`-transform_namespace` *`string`*	The namespace URI of the custom content transformation function named by `-transform_function`. Ignored if `-transform_module` is not specified. Default: no namespace. For details, see “Transforming Content During Ingestion” on page 66.
`-transform_param` *`string`*	Optional extra data to pass through to a custom transformation function. Ignored if `-transform_module` is not specified. Default: no namespace. For details, see “Transforming Content During Ingestion” on page 66.
`-transaction_size` *`number`*	When loading documents into the destination database, the number of requests to MarkLogic Server in one transaction. Default: 10. Maximum: 4000/*actualBatchSize*.

7.0 Using Direct Access to Extract or Copy Documents

Direct Access enables you to bypass MarkLogic Server and extract documents from a database by reading them directly from the on-disk representation of a forest. This feature is best suited for accessing documents in archived, offline forests.

This section covers the following topics:

7.1 When to Consider Using Direct Access

Direct Access enables you to extract documents directly from an offline or read-only forest without going through MarkLogic Server. A forest is the internal representation of a collection of documents in a MarkLogic database; for details, see Understanding Forests in the *Administrator's Guide*. A database can span multiple forests on multiple hosts.

Direct Access is primarily intended for accessing archived data that is part of a tiered storage deployment; for details, see Tiered Storage in the *Administrator's Guide*. You should only use Direct Access on a forest that is offline or read-only; for details, see "Limitations of Direct Access" on page 136.

For example, if you have data that ages out over time such that you need to retain it, but you do not need to have it available for real time queries through MarkLogic Server, you can archive the data by taking the containing forests offline, but still access the contents using Direct Access.

Use Direct Access with mlcp to access documents in offline and read-only forests in the following ways:

- The mlcp `extract` command to extracts archived documents from a database as flat files. This operation is similar to exporting documents from a database to files, but does not require a source MarkLogic Server instance. For details, see "Choosing Between Export and Extract" on page 137.
- The mlcp `import` command with `-input_file_type forest` imports archived documents as to another database as live documents. A destination MarkLogic Server instance is required, but no source instance.

You will likely get the best performance out of these operations if you use mlcp in distributed mode and already use HDFS for forest storage. Otherwise, the client where you execute mlcp can become a bottleneck due to resource limitations.

Since Direct Access bypasses the active data management performed by MarkLogic Server, you should not use it on forests receiving document updates. Additional restrictions apply. For details, see “Limitations of Direct Access” on page 136.

7.2 Limitations of Direct Access

You should only use Direct Access on a forest that meets one of the following criteria:

- The forest is offline and not in an error state. A forest is offline if the availability is set to offline, or the forest or the database to which it is attached is disabled. For details, see Taking Forests and Partitions Online and Offline in the *Administrator’s Guide*.
- The forest is online, but the `updates-allowed` state of the forest is `read-only`. For details, see Setting the Updates-allowed State on Partitions in the *Administrator’s Guide*.

The following additional limitations apply to using Direct Access:

- Accessing documents with Direct Access bypasses security roles and privileges. The content is protected only by the filesystem permissions on the forest data.
- Direct Access cannot take advantage of indexing or caching when accessing documents. Every document in each participating forest is read, even when you use filtering criteria such as `-directory_filter` or `-type_filter`. Filtering can only be applied after reading a document off disk.
- Direct Access skips property fragments.
- Direct Access skips documents partitioned into multiple fragments. For details, see Fragments in the *Administrator’s Guide*.
- Older versions of mlcp might not be able to read forest data from MarkLogic 9 or later. For best results, use the version of mlcp that corresponds to your MarkLogic version.

When you use Direct Access, mlcp skips any forest (or a stand within a forest) that is receiving updates or that is in an error state. Processing continues even when some documents are skipped.

When you use mlcp with Direct Access, your forest data must be reachable from the host(s) processing the input. In distributed mode, the forests must be reachable from the nodes in your Hadoop cluster. In local mode, the forests must be reachable from the host on which you execute mlcp.

If mlcp accesses large or external binaries with Direct Access, then the reachability requirement also applies to the large data directory and any external binary directories. Furthermore, these directories must be reachable along the same path as when the forest was online.

For example, if a forest was configured to use `hdfs://my/large/data` as a large data directory when it was live and the forest contains a large binary document, then the path `hdfs://my/large/data` must be resolvable from your Hadoop cluster (distributed mode) or mlcp client host (local mode). Similarly, if a forest contains an external binary documented inserted into the database with `/my/external-images/huge.jpg`, then `/my/external-images/huge.jpg` must be reachable.

7.3 Choosing Between Export and Extract

You can use the `export` and `extract` commands to save content in a MarkLogic database to files on the native file system or HDFS. You should usually use `export` rather than `extract`. The `extract` command is best suited for archive data in offline or read-only forests. Otherwise, use the `export` command.

The `extract` command places no load on MarkLogic Server. The `export` command offloads most of the work to your MarkLogic cluster. Thus, `export` honors document permissions, takes advantage of database indexes, and can apply transformations and filtering at the server. By contrast, `extract` bypasses security (other than file permissions on the forest files), must access all document sequentially, and applies a limited set of filters on the client.

The `export` command offers a richer set of filtering options than `extract`. In addition, `export` only accesses the documents selected by your options, while `extract` must scan the entirety of each input forest, even when extracting selected documents.

For more information, see the following topics:

- "Exporting Documents as Files" on page 95
- "Extracting Documents as Files" on page 137

7.4 Extracting Documents as Files

Use the mlcp `extract` command to extract documents from archival forest files to files on the native filesystem or HDFS. For example, you can extract an XML document as a text file containing XML, or a binary document as a JPG image.

To extract documents from a forest as files:

1. Set `-input_file_path` to the path to the input forest directory(s). Specify multiple forests using a comma-separated list of paths.

2. Select the documents to extract. For details, see "Filtering Forest Contents" on page 106.

    - To select documents in one or more collections, set `-collection_filter` to a comma separated list of collection URIs.
    - To select documents in one or more database directories, set `-directory_filter` to a comma separated list of directory URIs.

- To select documents by document type, set `-type_filter` to a comma separated list of document types.
- To select all documents in the database, leave `-collection_filter`, `-directory_filter`, and `-type_filter` unset.

3. Set `-output_file_path` to the destination file or directory on the native filesystem or HDFS. This directory must not already exist.

4. Set `-mode` to `local` or `distributed`:

   - If Hadoop is available and you want to distribute the workload across a Hadoop cluster, set `-mode` to `distributed`. Your input forests must be reachable across your Hadoop cluster.
   - If Hadoop is not installed or you want mlcp to perform the work locally, set `-mode` to `local`. Your input forests must be reachable from the host where you execute mlcp.

5. If you want to extract the documents as files in compressed files, set `-compress` to `true`.

   Note: If you are loading from the native filesystem in distributed mode or from HDFS in local mode, you might need to qualify the input file path with a URI scheme of `file:` or `hdfs:`. See "Understanding Input File Path Resolution" on page 32.

Filtering options can be combined. Directory names specified with `-directory_filter` should end with "/". All filters are applied on the client (or Hadoop task nodes in distributed mode), so every document is accessed, even if it is filtered out of the output document set.

Note: Document URIs are URI-decoded before filesystem directories or filenames are constructed for them. For details, see "How URI Decoding Affects Output File Names" on page 99.

For a full list of `extract` options, see "Extract Command Line Options" on page 140.

The following example extracts selected documents from the forest files in `/var/opt/MarkLogic/Forests/example` to the native filesystem directory `/space/mlcp/extracted/files`. The directory filter selects only the input documents in the database directory `/plays`.

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh extract -mode local \
    -input_file_path /var/opt/MarkLogic/Forests/example \
    -output_file_path /space/mlcp/extracted/files \
    -directory_filter /plays/
```

7.5 Importing Documents from a Forest into a Database

Use the following procedure to load all the files in a native or HDFS forest directory and its sub-directories. To load selected files, see “Filtering Documents Loaded From a Directory” on page 39. For more details on the command line options used in this procedure, see “Import Command Line Options” on page 84.

1. Set `-input_file_path` to the path to the input forest directory(s). Specify multiple forests using a comma-separated list of paths.

2. Set `-input_file_type` to `forest`.

3. Specify the connection information for the destination database using `-host`, `-port`, `-username`, and `-password`.

4. Select the files to extract from the input forest. For details, see “Filtering Forest Contents” on page 106. Filtering options can be used together.

   - To select documents in one or more collections, set `-collection_filter` to a comma separated list of collection URIs.
   - To select documents in one or more database directories, set `-directory_filter` to a comma separated list of directory URIs.
   - To select documents by document type, set `-type_filter` to a comma separated list of document types.
   - To select all documents in the database, leave `-collection_filter`, `-directory_filter`, and `-type_filter` unset.

5. If you want to exclude some or all of the document metadata in the forests:

   - Set `-copy_collections` to `false` to exclude document collections metadata.
   - Set `-copy_quality` to `false` to exclude document quality metadata.
   - Set `-copy_metadata` to `false` to exclude key-value metadata.

6. Set `-mode`:

   - If Hadoop is available and you want to distribute the workload across a Hadoop cluster, set `-mode` to `distributed`. Your input forests and the destination MarkLogic Server instance must be reachable across your Hadoop cluster.
   - If Hadoop is not installed or you want mlcp to perform the work locally, set `-mode` to `local`. (This is the default mode unless you set the `HADOOP_CONF_DIR` variable.) Your input forests and the destination MarkLogic Server instance must be reachable from the host where you run mlcp.

Note: If you are loading from the native filesystem in distributed mode or from HDFS in local mode, you might need to qualify the input file path with a URI scheme of `file:` or `hdfs:`. See "Understanding Input File Path Resolution" on page 32.

By default, an imported document has a database URI based on the input file path. You can customize the URI using options. For details, see "Controlling Database URIs During Ingestion" on page 33.

The following example command loads the documents in the forests in `/var/opt/MarkLogic/Forests/example`:

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh import -host localhost -port 8000 -username user \
    -password password -input_file_type forest \
    -input_file_path /var/opt/MarkLogic/Forests/example
```

7.6 Extract Command Line Options

This section summarizes the command line options available with the mlcp `extract` command. An extract command requires the `-input_file_path` and `-output_file_path` options. That is, an extract command has the following form:

```
mlcp.sh extract -input_file_path forest-path \
    -output_file_path dest-path ...
```

The following table lists command line options that define the characteristics of the extraction:

Option	Description
`-collection_filter` *`comma-list`*	A comma-separated list of collection URIs. mlcp extracts only documents in these collections. This option can be combined with other filter options. Default: All documents.
`-compress` *`boolean`*	Whether or not to compress the output. Mlcp might generate multiple compressed files. Default: `false`.
`-conf` *`filename`*	Pass extra setting to Hadoop when using distributed mode. For details, see "Setting Custom Hadoop Options and Properties" on page 16. This option must appear before mlcp-specific options.

Option	Description
`-D` *`property=value`*	Pass a configuration property setting to Hadoop when using distributed mode. For details, see "Setting Custom Hadoop Options and Properties" on page 16. This option must appear before mlcp-specific options.
`-directory_filter` *`comma-list`*	A comma-separated list of database directory names. mlcp extracts only documents from these directories, plus related metadata. Directory names should usually end with "/". This option can be combined with other filter options. Default: All documents and related metadata.
`-hadoop_conf_dir` *`string`*	When using distributed mode, the Hadoop config directory. For details, see "Configuring Distributed Mode" on page 15.
`-max_split_size` *`number`*	The maximum number of document fragments processed per split. Default: 50000.
`-mode` *`string`*	Export mode. Accepted values: `distributed`, `local`. Distributed mode requires Hadoop. Default: `local`, unless you set the `HADOOP_CONF_DIR` variable; for details, see "Configuring Distributed Mode" on page 15.
`-options_file` *`string`*	Specify an options file pathname from which to read additional command line options. If you use an options file, this option must appear first. For details, see "Options File Syntax" on page 9.
`-output_file_path` *`string`*	Destination directory where the documents are saved. The directory must not already exist.

Option	Description
`-thread_count` *`number`*	The number of threads to spawn for concurrent exporting. The total number of threads spawned by the process can be larger than this number, but this option caps the number of concurrent sessions with MarkLogic Server. Only available in `local` mode. Default: 4.
`-type_filter` *`comma-list`*	A comma-separated list of document types. mlcp extracts only documents with these types. This option can be combined with other filter options. Allowed documentypes: `xml`, `text`, `binary`. Default: All documents.

8.0 Troubleshooting

This chapter includes tips for debugging some common problems. The following topics are covered:

8.1 Checking Your Runtime Environment

You can use the mlcp `version` command to generate a report of key software versions mlcp detects in your runtime environment. This is useful for confirming your path and other environment settings create the environment you expect or mlcp requires.

For example, the command below reports the version of mlcp, the Java JRE, and Hadoop that mlcp will use at runtime, plus the versions of MarkLogic supported by this version of mlcp.

```
$ mlcp.sh version
ContentPump version: 8.0
Java version: 1.7.0_45
Hadoop version: 2.6.0
Supported MarkLogic versions: 6.0 - 8.0
```

Note that not all features of mlcp are supported by all versions of MarkLogic, even within the reported range of supported versions. For example, if MarkLogic version X introduces a new feature that is supported by mlcp, that doesn't mean you can use mlcp to work with the feature in MarkLogic version X-1.

8.2 Resolving Connection Issues

All mlcp command lines include host and port information for connecting to MarkLogic Server. This host must be reachable from the host where you run mlcp. In distributed mode, this host must also be reachable from all the nodes in your Hadoop cluster.

In addition, mlcp connects directly to hosts in your MarkLogic Server cluster that contain forests of the target database. Therefore, all the hosts that serve a target database must be reachable from the host where mlcp runs (local mode) or the nodes in your Hadoop cluster (distributed mode).

Mlcp gets the lists of participating hosts by querying your MarkLogic Server cluster configuration. If a hostname returned by this query is not resolvable, mlcp will not be able to connect, which can prevent document loading.

If you think you might have connection issues, enable debug level logging to see details on name resolution and connection failures. For details, see "Enabling Debug Level Messages" on page 144.

8.3 Enabling Debug Level Messages

You can enable debug level log messages to see detailed debugging information about what mlcp is doing. Debug logging generates many messages, so you should not enable it unless you need it to troubleshoot a problem.

To enable debug logging:

1. Edit the file `MLCP_INSTALL_DIR/conf/log4j.properties`. For example, if mlcp is installed in `/opt/mlcp`, edit `/opt/mlcp/conf/log4j.properties`.

2. In `log4j.properties`, set the properties `log4j.logger.com.marklogic.mapreduce` and `log4j.logger.com.marklogic.contentpump` to `DEBUG`. For example, include the following:

```
log4j.logger.com.marklogic.mapreduce=DEBUG
log4j.logger.com.marklogic.contentpump=DEBUG
```

You may find these property settings are already at the end of `log4j.properties`, but commented out. Remove the leading "#" to enable them.

8.4 Error loading class com.marklogic.contentpump.ContentPump

The cause of the following error is usually running `mlcp.sh` on Windows under Cygwin, which is not a supported configuration.

```
Error: Could not find or load main class com.marklogic.contentpump.ContentPump
```

You should always use `mlcp.bat` on Windows.

8.5 No or Too Few Files Loaded During Import

If `ATTEMPTED_INPUT_RECORD_COUNT` is non-zero and `SKIPPED_INPUT_RECORD_COUNT` is zero, then errors may have occurred on the server side or your combination of options may be inconsistent. For example:

- The input type is `documents`, and the document type is set to (or determined to be) XML, but the input file fails to parse properly as XML. Correct the error in the input data and try again.
- You set `-input_file_path` to a location containing compressed files, but you do not set `-input_compressed` and `-input_compression_codec`. In this case, mlcp will load the compressed files as binary documents, rather than creating documents from the contents of the compressed files.
- You set `-document_type` to a value inconsistent with the input data referenced by `-input_file_path`.

If `ATTEMPTED_INPUT_RECORD_COUNT` is non-zero and `SKIPPED_INPUT_RECORD_COUNT` is non-zero, then there are probably formatting errors in your input that mlcp detected on the client. Correct the input errors and try again. For example:

- A syntax error was encountered while splitting an aggregate XML file into multiple pieces of document content.
- A delimited text file contains records (lines) with an incorrect number of column values or with no value for the URI id column.

If mlcp reports an `ATTEMPTED_INPUT_RECORD_COUNT` of 0, then the tool found no input documents meeting your requirements. If there are errors or warnings, correct them and try again. If there are no errors, then the combination of options on your command line probably does not select any suitable documents. For example:

- You set `-input_compressed -input_compression_codec zip`, but `-input_file_path` references a location that contains no ZIP files.
- You set `-input_compressed` and set -input_file_path to a location containing compressed files, but failed to set `-input_compression_codec`.

8.6 Unable to load realm info from SCDynamicStore

Depending on your JVM version, you might see the message“Unable to load realm info from SCDynamicStore” when using mlcp if your system has Kerberos installed and `krb5.conf` doesn’t explictly list the realm information. You can safely ignore this message.

8.7 File Not Found in Distributed Mode

If you use mlcp in distributed mode and your input or output pathname pathname contains whitespace, you may get a FileNotFound or other error. Change your pathnames to eliminate the whitespace.

In distributed mode, mlcp uses Hadoop and HDFS to manage the distributed work. Some versions of Hadoop and HDFS cannot handle whitespace in pathnames.

8.8 XDMP_SPECIALPROP Error on Archive Import

An `XDMP_SPECIALPROP` when importing documents from an archive is caused by attempting to update the “last modified” document property that is maintained by MarkLogic on the destination database. To eliminate this error, choose one of the following solutions:

- Set `-copy_propertie`s to `false` on your import command line so that mlcp does attempt to import any document properties.
- Temporarily disable the “maintain last modified” setting on the destination database using the Admin Interface or the library function `admin:set-maintain-last-modified`.

8.9 JCE Warning When Using MapR

If you see the following warning when using mlcp with MapR, make sure you have installed the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files compatible with your JRE.

```
WARN util.KerberosUtil: JCE Unlimited Strength Jurisdiction Policy
Files are not installed. This could cause authentication failures.
```

8.10 Warning that a Job Remains Running

If you interrupt an mlcp job before it completes, such as by entering Ctrl-C, the job might continue running.

An mlcp job in distributed mode distributes its work across a Hadoop cluster. Interrupting mlcp locally does not stop the work already distributed to Hadoop. In local mode, an interrupted job will shutdown gracefully as long as it can finish withint 30 seconds.

If mlcp cannot gracefully shut down the job, you might see the following warning:

```
WARN contentpump.ContentPump: Job yourJobName status remains RUNNING
```

9.0 Technical Support

MarkLogic provides technical support according to the terms detailed in your Software License Agreement or End User License Agreement.

We invite you to visit our support website at http://help.marklogic.com to access information on known and fixed issues, knowledge base articles, and more. For licensed customers with an active maintenance contract, see the Support Handbook for instructions on registering support contacts and on working with the MarkLogic Technical Support team.

Complete product documentation, the latest product release downloads, and other useful information is available for all developers at http://developer.marklogic.com. For general questions, join the general discussion mailing list, open to all MarkLogic developers.

10.0 Copyright

MarkLogic Server 9.0 and supporting products.
Last updated: April 28, 2017

COPYRIGHT

For all copyright notices, including third-party copyright notices, see the Combined Product Notices.