OODT Filemgr User Guide

- The File Manager
  - An Overview of What is Installed
  - Configuring and Running the File Manager
    - Whats going to happen?
    - Now for some configuration
    - What have we configured?
    - How metadata is collected?
    - A brief overview of filemgr-client and query-tool
    - Command: filemgr-client
    - Command: query-tool
    - A Typical User Scenario
- A few more tools
- Tips and Tricks for FileManager

The File Manager

This self guided tutorial is intended for first time users.

The fact that you've found this page, I assume that you are seriously thinking of using the OODT File Manager but are eager to get something up and running. It hopefully also means that you've checked out the code and built a cas-filemgr install target (e.g. a `cas-filemgr-${version}-dist.tar.gz` file).

This tutorial is by no means a complete overview of all the File Managers functionality. However, it's an attempt to get you started using the basic tools. Like learning to drive a car, the most difficult part is getting it started and on the road!

The following topics are covered on this page:

- An Overview of What is Installed
- Configuring and Running the File Manager
- A Typical User Scenario - ingesting and querying

An Overview of What is Installed

Assumption - you have built or have access to a cas-filemgr install target. This also means that you've correctly configured maven and java for your system.

Here are the commands to install the cas-filemgr target from a tarfile. You will need to fit in the "..." with the appropriate content.

```
$ mkdir -p /usr/local/oodt/
$ tar xzvf .../filemgr/target/cas-filemgr-${version}-dist.tar.gz -C /usr/local/oodt/
$ cd /usr/local/oodt/
$ ln -s cas-filemgr-${version}/ cas-filemgr
```

The decompressed tar file creates a directory structure that looks as follows:

```
.
 bin
    filemgr
    filemgr-client
    query-tool
 etc
    filemgr.properties
    mime-types.xml
 lib
    *.jar
 logs
 policy
|    cmd-line-actions.xml
|    cmd-line-options.xml
|    core
|       elements.xml
|       product-type-element-map.xml
|       product-types.xml
|   |
|    trace
|   |    elements.xml
|   |    product-type-element-map.xml
|   |    product-types.xml
|   |
|    geo
|   |    elements.xml
|   |    product-type-element-map.xml
|   |    product-types.xml
|   |
|    (additional policy sub directories)
 run
```

Please note, if you are using version 0.3 of OODT or earlier, the policy directory will look like this (with no sub directories):

```
policy
    elements.xml
    product-type-element-map.xml
    product-types.xml
```

Here is a brief description of each directory that you see listed:

- `bin` : contains shell convenience scripts for launching java classes
- `etc` : contains configuration files, i.e. *.property and *.xml files
- `lib` : contains java resources, i.e *.jar files
- `logs` : contains file manager log files.
- `policy` : contains product specifications, i.e *.xml specification files

The `bin` directory contains a number of executables:

- `filemgr` : file manager (startup/shutdown) script
- `filemgr-client` : file manager client interface script
- `query-tool` : catalog query tool

Configuring and Running the File Manager

You're now ready to run the file manager!

```
$ cd /usr/local/oodt/cas-filemgr/bin
$ ./filemgr --help
Usage: ./filemgr {start|stop|status}
$ ./filemgr start
```

Whats going to happen?

The filemgr should be up and running, however, some WARNING messages *may* appear, complaining about configuration.

If you get a java.net.BindException exception, make sure that no other service is running on port 9000. This is the port for an RPC interface that will be used for transferring data files into a repository.

There's also a new file in the /usr/local/oodt/run directory. The file contains the filemgr process id. This is typical for *nix service house keeping. It is done to try and avoid running multiple filemgr services.

There's also a new log file /usr/local/oodt/cas-filemgr/logs/cas_filemgr0.log. Tailing this file can often alert to you problems.

```
$ tail -f /usr/local/oodt/cas-filemgr/logs/cas_filemgr0.log
```

Now for some configuration

To do anything useful with your filemgr, you will need to specify some configurations in the /usr/local/oodt/cas-filemgr/etc/filemgr.properties file.

Here is a basic modification to the filemgr.properties file:

filemgr.properties

```
org.apache.oodt.cas.filemgr.catalog.lucene.idxPath=/usr/local/oodt/cas-filemgr/catalog
org.apache.oodt.cas.filemgr.repositorymgr.dirs=file:///usr/local/oodt/cas-filemgr/policy/core
org.apache.oodt.cas.filemgr.validation.dirs=file:///usr/local/oodt/cas-filemgr/policy/core
org.apache.oodt.cas.filemgr.mime.type.repository=/usr/local/oodt/cas-filemgr/etc/mime-types.xml
```

You will also need to specify a repository path in the product-types.xml file. Make sure that this path exists before you change the repository path xml element.

product-types.xml

```
<repository path="file:///var/archive/data"/>
```

Restart your filemgr so that it re-reads the filemgr.properties and product-types.xml:

```
$ cd /usr/local/oodt/cas-filemgr/bin
$ ./filemgr restart
```

What have we configured?

- A place to store your catalog, i.e. the database of metadata.
- A place to store your ingested files, i.e. the repository.

- The location of your policy directory for product specifications.
- Your mime-types configuration file for file recognition.

How metadata is collected?

Now for some brief notes about how metadata is collected. The filemgr captures metadata in two different ways - from client side metadata extraction and server side metadata extraction.

Client side metadata is passed to the filemgr via an xml formatted metadata file. E.g. a file called `blah.txt` can have a metadata file called `blah.txt.met`. This met file can be created in many ways, even by hand! And thats exactly what we're going to do.

Server side metadata is generated by using java classes and the extractors that will be used are configured in the product-types.xml file in the chosen policy directory. For this example configuration, you should have `/usr/local/oodt/cas-filemgr/policy/oodt` as the policy directory, unless you're running version 0.3 or earlier of OODT, in which case you should have `/usr/local/oodt/cas-filemgr/policy` as the policy directory.

Now would be a good time to have a quick look at the `product-types.xml` file. It contains some critical information about what is going to happen when we ingest our first file into the repository.

Specified in the product-types.xml file, there is a default product type called GenericFile. This is the product type that we are going to use for the first file for ingestion.

For the GenericFile type find the `<metadata>` key. It's specifying some metadata. We're defining the product type!

For the GenericFile type find the `<metExtractors>` key. It's specifying some extractors to use for server side metadata extraction, namely: CoreMetExtractor, MimeTypeExtractor, FinalFileLocationExtractor. For more details about metadata and extractors see Metadata Extractors.

If you're feeling curious, check out the other xml files in the `/usr/local/oodt/cas-filemgr/policy` subdirectories to get a better feel for how we define product types and elements. For a discussion of best practices w.r.t File Manager Policy, the reader is referred to Everything you want to know about File Manager Policy

A brief overview of filemgr-client and query-tool

These commands are found in `/usr/local/oodt/cas-filemgr/bin`.

Command: filemgr-client

In order to trigger a file ingestion we're going to use the `filemgr-client`. This is by no means the most automated way to ingest data into an repository, however it's a really easy and intuitive way to trigger a file ingestion. The `filemgr-client` is a wrapper script, making it easier to invoke a java executable from the command line.

```
$ cd /usr/local/oodt/cas-filemgr/bin
$ ./filemgr-client --help
filemgr-client --url <url to xml rpc service> --operation [<operation> [params]]
operations:
--addProductType --typeName <name> --typeDesc <description>
    --repository <path> --versionClass <classname of versioning impl>
--ingestProduct --productName <name> --productStructure <Hierarchical|Flat>
    --productTypeName <name of product type> --metadataFile <file>
    [--clientTransfer --dataTransfer <java class name of data transfer factory>]
    --refs <ref1>...<refn>
--hasProduct --productName <name>
--getProductTypeByName --productTypeName <name>
--getNumProducts --productTypeName <name>
--getFirstPage --productTypeName <name>
--getNextPage --productTypeName <name> --currentPageNum <number>
--getPrevPage --productTypeName <name> --currentPageNum <number>
--getLastPage --productTypeName <name>
--getCurrentTransfer
--getCurrentTransfers
--getProductPctTransferred --productId <id> --productTypeName <name>
--getFilePctTransferred --origRef <uri>
```

As you can see there's a number of different ways this command can be executed.

The first command line argument is `--url`. This is the location of the filemgr xml-rpc data transfer interface. Looking at the filemgr logs (specifically cas_filemgr0.log), we see an INFO statement telling us that local data transfer is enable on http://localhost:9000. This is the url that we need to specify.

The second command line argument is `--operation` and there are 13 different types of operations that are possible! For now we are going to use the `--ingestProduct` operation. From the help command you can see that the `--ingestProduct` operation requires some further command line arguments to be specified.

However, before we take a look at the `--operation --ingestProduct`, I would first like to shed a bit more light on the `query-tool` command.

Command: query-tool

This is a very useful wrapper script to query the content of your repository.

```
$ cd /usr/local/oodt/cas-filemgr/bin
$ ./query-tool
Must specify a query and filemgr url!
Usage: QueryTool [options]
options:
--url <fm url>
  Lucene like query options:
    --lucene
         -query <query>
  SQL like query options:
    --sql
         -query <query>
         -sortBy <metadata-key>
         -outputFormat <output-format-string>
```

We see that we need to set some command line arguments to get anything useful out of the query tool. Try the next command:

```
$ ./query-tool --url http://localhost:9000 --sql -query 'SELECT * FROM GenericFile'
```

This should throw an exception, telling us it failed to perform a query. This is because there is no catalog yet (and therefore the GenericFile information does not exist). In fact if you have a look there is no catalog directory:

```
$ ls /usr/local/oodt/cas-filemgr/catalog
ls: /usr/local/oodt/cas-filemgr/catalog: No such file or directory
```

A Typical User Scenario

Time to ingest a very, very simple file. If you have not already, restart your filemgr so that it re-reads the filemgr.properties:

```
$ cd /usr/local/oodt/cas-filemgr/bin
$ ./filemgr restart
```

For this simple ingestion we are not going to include any client side metadata, all the metadata collection will happen on the server side using the specified *Extractor extractors in the `product-types.xml` file.

Create a text file and its metadata file for ingestion:

```
$ echo 'hello' > /tmp/blah.txt
$ touch /tmp/blah.txt.met
```

Add the following xml to the /tmp/blah.txt.met file:

blah.txt.met

```
<cas:metadata xmlns:cas="http://oodt.jpl.nasa.gov/1.0/cas">
</cas:metadata>
```

Lets ingest the file! For `--operation --ingestProduct` we need to specify the following arguments:

- `--productName` : The name you want for your ingested product
- `--productStructure` : Flat file or directory (i.e. hierarchical). Yes... we can ingest whole directories as one product
- `--productTypeName` : A product type (as per product-types.xml)
- `--metadataFile` : The client side metadata file
- `--refs` : The product location

There's also an optional argument `--clientTransfer`, however, we're going to leave this and use the default local transfer.

```
[--clientTransfer --dataTransfer <java class name of data transfer factory>]
```

Here is the complete command:

```
$ ./filemgr-client --url http://localhost:9000 --operation --ingestProduct --productName blah.txt --productStructure Flat --productTypeName GenericFile --metadataFile file:///tmp/blah.txt.met --refs file:///tmp/blah.txt
```

The output should look like:

```
Sep 16, 2011 2:09:42 PM org.apache.oodt.cas.filemgr.system.XmlRpcFileManagerClient <init>
...
...
ingestProduct: Result: c2fbf4b9-e05c-11e0-9022-77a707615e7f
```

You've just archived your first file 😉.

To complete the process, lets see if we can retrieve the metadata. Run the query command again:

```
$ cd /usr/local/oodt/cas-filemgr/bin
$ ./query-tool --url http://localhost:9000 --sql -query 'SELECT * FROM GenericFile'
```

The output should look like:
```
Sep 16, 2011 2:21:54 PM org.apache.oodt.cas.filemgr.system.XmlRpcFileManager complexQuery
INFO: Query returned 1 results
/var/archive/data/blah.txt,GenericFile,blah.txt,blah.txt,2011-09-16T14:09:43.405+02:00,c2fbf4b9-e05c-11e0-9022-
77a707615e7f,Flat,text/plain,text,plain
```

Check to see if the file has appeared in the archive:
```
$ ls /var/archive/data/blah.txt/
blah.txt
```

Query commands do not depend on the underlying catalog implementation. The `--sql` and `--lucene` instead describe the filemgr query syntax.

At the time of writing this tutorial, composing queries using query-tool is not entirely straight forward, but entirely usable. Formatting of these queries is critical, small deviations from the syntax can result in the query return an unexpected value or throwing an exception.

Some things to note about SQL queries:

1. Use double quotes ("") for when specifying the SQL syntax. The single quote (") is used for string values in a WHERE clause, e.g WHERE Filename='blah.txt'
2. Count the number of -- before each command line option. Some are -- and others are -.
3. The order of the return values for a search is not guaranteed unless you specify the \outputFormat option.

Here is a somewhat verbose example that uses all the SQL-like syntax that I am currently aware of (apologies for all the line breaks).

```
$ cd /usr/local/oodt/cas-filemgr/bin
$ ./query-tool --url http://localhost:9000 --sql \
-query "SELECT CAS.ProductReceivedTime,CAS.ProductName,CAS.ProductId,ProductType,\
ProductStructure,Filename,FileLocation,MimeType \
FROM GenericFile WHERE Filename='blah.txt'" -sortBy 'CAS.ProductReceivedTime' \
-outputFormat '$CAS.ProductReceivedTime,$CAS.ProductName,$CAS.ProductId,$ProductType,\
$ProductStructure,$Filename,$FileLocation,$MimeType'
```

The output should look like:
```
2011-10-07T10:59:12.031+02:00,blah.txt,a00616c6-f0c2-11e0-baf4-65c684787732,
GenericFile,Flat,blah.txt,/var/kat/archive/data/blah.txt,text/plain
```

Now you can also check out some of the other 12 `--operation` possibilities for filemgr-client. For instance:

```
$ ./filemgr-client --url http://localhost:9000 --operation --hasProduct --productName blah.txt
```

Or:

```
$ ./filemgr-client --url http://localhost:9000 --operation --getFirstPage --productTypeName GenericFile
```

A few more tools

Cameron Goodale has written some useful command line tools aliases that are worth mentioning before we continue. See the following two web pages: https://issues.apache.org/jira/browse/OODT-306
BASH and TCSH shell tools for File Manager

Tips and Tricks for FileManager

Q: My Lucene Index Catalog is running slow now that I have over 100,000 products cataloged. How can I get the speed back?

A: Run this command:
```
java -Djava.endorsed.dirs=<path/to/lib/dir> org.apache.oodt.cas.filemgr.tools.OptimizeLuceneCatalog --catalogPath
<path/to/catalog>
```