# ModeShape Artifacts Layout

**Walkthrough**

The following steps simulate a typical user session. An end result (i.e a layout of file and directories) is then shown.

The user creates a node ('greetings_en') through the UI and uploads content – in this example it's a simple text file (fcrepo4_greetings.txt) with a string ("hello, world!"):

### Start Fedora

The modeshape configuration specifies using a file-based backend store.

```
cd fcrepo4/fcrepo-webapp
mvn jetty:run -Dfcrepo.modeshape.configuration=classpath:config/single-file/repository.json
```

### Add Content

```
curl -X PUT --data-binary "@fcrepo4_greetings.txt" "http://localhost:8080/rest/greetings_en/ds0/fcr:content"
```

Fedora will create a directory "fcrepo4-data" in the current working directory. The default directories found in "fcrepo4-data" will be the following:

```
> ls fcrepo4-data
com.arjuna.ats.arjuna.common.ObjectStoreEnvironmentBean.default.objectStoreDir
com.arjuna.ats.arjuna.objectstore.objectStoreDir
fcrepo.activemq.dir
fcrepo.ispn.repo.CacheDirPath
fcrepo.modeshape.index.location
```

"fcrepo.ispn.repo.CacheDirPath" contains the generated data files. "fcrepo.modeshape.index.location" contains the Lucene index.

### Inspecting Generated Data Files

The serialized Fedora nodes can be found in the "fcrepo.ispn.repo.CacheDirPath/FedoraRepository" directory. The files in that directory would look something like this:

```
> ls fcrepo4-data/fcrepo.ispn.repo.CacheDirPath/FedoraRepository/
-891938816
1008466944
1016939520
102151168
1036310528
...
```

The generated files contain serialized data about each of the JCR/Fedora nodes. Running the cURL command above, for example, creates hundreds of *bin ary* files.

Some key things to note about the files and their contents:

- The file names in the case of FileCacheStore are just signed 32-bit integers, so the usual and maximum length of file names is 10 or 11 characters. The file names (i.e. integers) are generated by calculating the Java hashCode of the node's UUID string (prefixed with root node UUID and masking it by a 22 fixed bit mask). For example, the following Java snippet code calculates the file name for root node (UUID "87a0a8c75085d64/"):

  ```
  fileName = "87a0a8c7505d64/".hashCode() & 0xfffffc00; //equals -891938816
  ```

  Therefore, file **-891938816** would contain data about the root node.

- Each of the files contain serialized ModeShape nodes. Although the generated files are binary, using a tool (a simple utility is under development) can help in reading the contents of the file (It might not be easy to read these files using mongoDB bsondump alone). For example, for our root node file (-891938816) , a tool might display the root node data in JSON as following:

```
{ "properties" :
  { "http://www.jcp.org/jcr/1.0" :
    { "primaryType" :
      { "$name" : "mode:root" } , "uuid" : "87a0a8c7505d64/"
    }
  } ,
 "children" :
    [ { "key" : "87a0a8c317f1e7jcr:system" , "name" : "jcr:system" } ,
      { "key" : "7c366d9-25bc-4e2d-9e53-4428fe7b8152" , "name" : "greetings_en" } ] ,
    "childrenInfo" : { "count" : 2 }
}
```

As an aside, a tool could be written to show the full file-to-content mapping; the utility under development aims to display this information like the following snippet:

```
-891938816  UUID    87a0a8c7505d64 mode:root
-1857312768 UUID    7c366d9-25bc-4e2d-9e53-4428fe7b8152 greetings_en
328781824   UUID    7e84184d-3a8e-4f57-8e49-a550f1c19b3a greetings_en/ds0
39043072    UUID    fc859bab-6f7b-46de-9b4c-f40dfe28643c hello, world!
```

- The serialization is done by the JBoss serialization library, not JDK's native object serialization machinery. For this reason the generated serialized files look different from an ordinary JDK serialized file. JBoss Marshalling can be configured to use custom serialization classes that read and write content in the format of the repository's choosing.
- The data is encoded in Binary JSON (BSON). If the file containing the root node (referencing the node 'greetings_en') is opened up in a hex editor, you would see /u0002 preceding strings (such as "name","key"); /u0004 preceding an array (representing sub-nodes); /u0003 representing the UUID -- in accordance with the BSON spec.

```
\u0000\u0000\u0000\u0004children\u0000ü\u0000\u0000\u0000\u00030\u0000<\u0000\u0000\u0000
\u0002key\u0000\u0019\u0000\u0000\u000087a0a8c317f1e7jcr:system\u0000\u0002name\
u0000\u000B\u0000\u0000\u0000jcr:system\u0000\u0000\u00031\u0000X
\u0000\u0000\u0000\u0002key\u00003\u0000\u0000\u000087a0a8c7505d6417c366d9-25bc-4e2d-9e53-
4428fe7b8152\u0000\u0002name\u0000\u0000\u0000\u0000greetings_en\u0000\u0000\u0000\u0003childrenI
```

### Inspecting Individual Binary Data Files

The *actual* binary file (e.g. -891938816) looks something like this when opened up in a text editor:

87a0a8c7505d64[1]/7org.infinispan.schematic.internal.SchematicEntryLiteral6org.infinispan.marshall.jboss.JBossExternalizerAdapterq externalizer$org.infinispan.marshall.ExternalizerDorg.infinispan.schematic.internal.SchematicEntryLiteral

$Externalizer7org.infinispan.schematic.internal.SchematicExternalizer8org.infinispan.schematic.internal.document.BasicDocument;?org.infinispan.schematic.internal.document.DocumentExternalizer[2];

23metadata?id87a0a8c7505d64/contentTypeapplication/jsoncontent>propertiesghttp://www.jcp.org/jcr/1.0FprimaryType$name**mode:root**[3]uuid

87a0a8c7505d64/children0[4]<key87a0a8c317f1e7jcr:systemnamejcr:system1key387a0a8c7505d6417c366d9-25bc-4e2d-9e53-4428fe7b8152name

greetings_en[5]childrenInfocount55[6]

Key elements are annotated:

1. 87a0a8c7505d64 refers to node UUID
2. The grayed out text refers to Modeshape classes responsible for data representation and serialization. The JBoss Marshalling serialization format contains the name of datastructures i.e. custom marshallers in org.infinispan.schematic.internal.*
3. Node name.
4. The file contains binary data for specifying different internal attributes, so when opened up in an ordinary text editor, the text editor might show some numbers as garbage characters. These numbers have meaning for node's attributes, though. For example, the highlighted entry #4 specifies an array of the children of a node. The index number of the child precedes its key and name. So, if object "greetings_en" gets another sibling "greetings_fr", the latter's entry would be preceded by 2 (its index). The full children node array would now appear something like:

   /children**0**<key87a0a8c317f1e7jcr:systemname**jcr:system**

   **1**Xkey387a0a8c7505d6417c366d9-25bc-4e2d-9e53-4428fe7b8152name**greetings_en**

   **2**Xkey387a0a8c7505d64aa21b2ee-d22a-4b0a-8b43-a8e8b58c5ec6name**greetings_fr**
5. Children count in binary.

Similarly, for the datastream content (omitting ModeShape API artifacts):

87a0a8c7505d64fc859bab-6f7b-46de-9b4c-f40dfe28643c[1] . . .[2]

metadatabid387a0a8c7505d6 4fc859bab-6f7b-46de-9b4c-f40dfe28643[3]ccontentTypeapplication/ jsoncontent[4]Ékey387a0a8c7505d64fc859bab-6f7b-46de-9b4c- f40dfe28643c[3]parent387a0a8c7505d647**e84184d-3a8e-4f57-8e49-a550f1c19b3a**[5]properties http:// www.jcp.org/jcr/1.0"primaryType$name

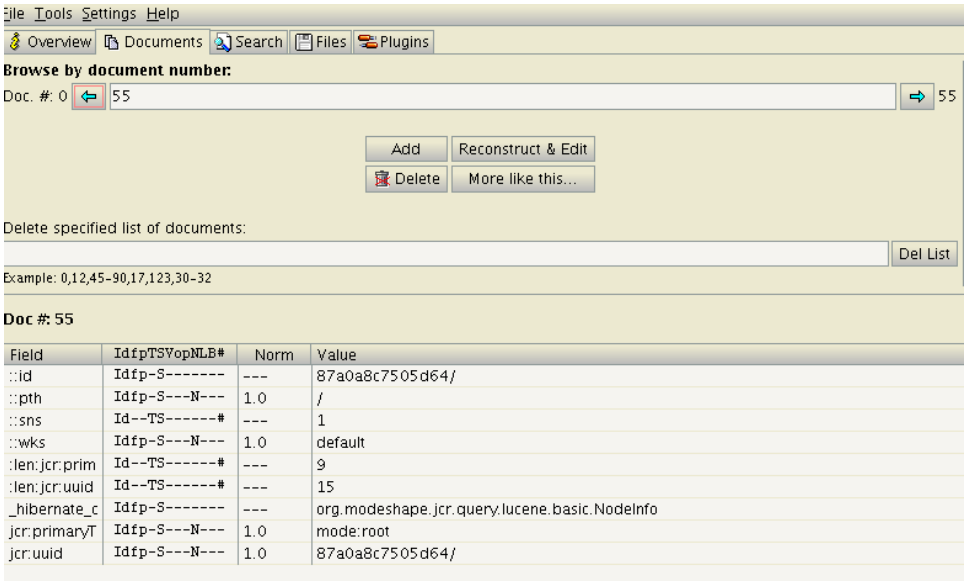
nt:resourcedata**hello, world!**[6] lastModified.$date2013-12-09T23:51:00.520-05:00[7]mixinTypesL0D$name4{http://fedora.info/ definitions/v4/rest-api#} binarylastModifiedBy bypassAdmin[8]mimeTypeapplication/octet-stream[9]http://fedora.info/definitions/v4/rest-api#NdigestA $uri2urn:sha1: e91ba0972b9055187fa2efa8b5c156f487a8293a[10]http://www.loc.gov/premis/rdf/ v1#hasSize55

Some of the elements of interests are:

1. Root UUID
2. Omitted Serialization artifacts (see note on parent node for details).
3. UUID
4. Content type of document (in this case it's the default content type for documents).
5. Parent UUID (datastream)
6. Actual content ("hello world" text in this case).
7. Last modified date.
8. Last modified admin.
9. Content type of datastream.
10. SHA-1 generated by Fedora.

## Inspecting Indexing Folder

ModeShape currently makes use of Hibernate Search to manage Lucene indexes. The indexes can be viewed by using Luke, e.g.
These files can be found in the "fcrepo.modeshape.index.location" directory.




## Inspecting ObjectStore Folders

Directories "com.arjuna.ats.arjuna.objectstore.objectStoreDir" and "com.arjuna.ats.arjuna.common.ObjectStoreEnvironmentBean.default.objectStoreDir" are JBoss JTA transaction engine artifacts. The default Fedora Infinispan configuration attempts to find a JBossJTA transaction manager implementation via "org.infinispan.transaction.lookup.GenericTransactionManagerLookup". This configuration uses Arjuna ShadowFileStore as a backend, resulting in several directories within fcrepo4-data such as "object-store" and "object-store-default":

```
|-object-store
|---ShadowNoFileLockStore
|-----defaultStore
|-------Recovery
|---------TransactionStatusManager
|-object-store-default
|---ShadowNoFileLockStore
|-----defaultStore
```

A detailed description of the artifacts maintained by the JBossJTA implementation is most likely beyond the scope of this document (at least for now).

# Infinispan Configuration Options

Depending on the configured Infinispan backend, the directory layout and contents of the binary files would be different. The following sections covers other cache store options.

## LevelDB Backend

Currently, the default configuration outputs Fedora data to LevelDB (a fast filesystem based key-value store). When Fedora 4 is started, ModeShape (actually Infinispan and LevelDB in the background) will create several directories on the filesystem. Currently, the directories created are:

1. fcrepo.ispn.binary.CacheDirPath (binary data)
2. fcrepo.ispn.CacheDirPath (metadata)
3. fcrepo.ispn.repo.CacheDirPath (repository)
4. fcrepo.modeshape.index.location

The layout of files in directories 1-3 is determined by LevelDB. Some of the important files are:

1. File .log holds entries for recent transactions. The relevant API for representing these entries is modeshape-schematics (see, e.g., org.infinispan.schematic.SchematicEntry)
2. File .sst stores these entries when the .log file reaches a size threshold. A new log file is generated.
3. File MANIFEST.x records info about .sst files (among other things).
4. File CURRENT specifies the current MANIFEST file.

### Inspecting Individual Binary Data Files

As is the case with the FileCacheStore, the .log files are binary. The default initial .log file is 000003.log, and it contains serialized entries for all the nodes from recent transactions. If the repository has only a few hundred added nodes, this file will contain all the nodes (i.e. the additions and default properties like mix:etag).