MarkLogic®

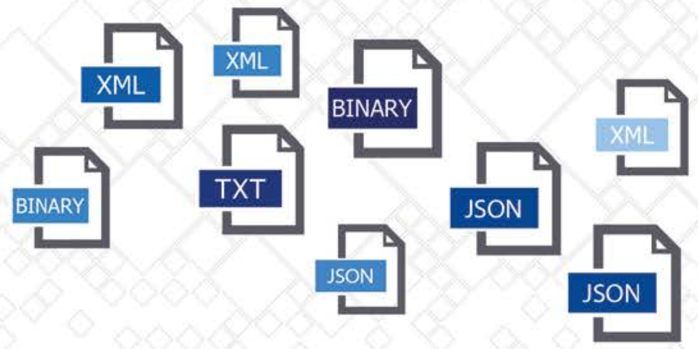
Unit 6: Loading and Managing Data

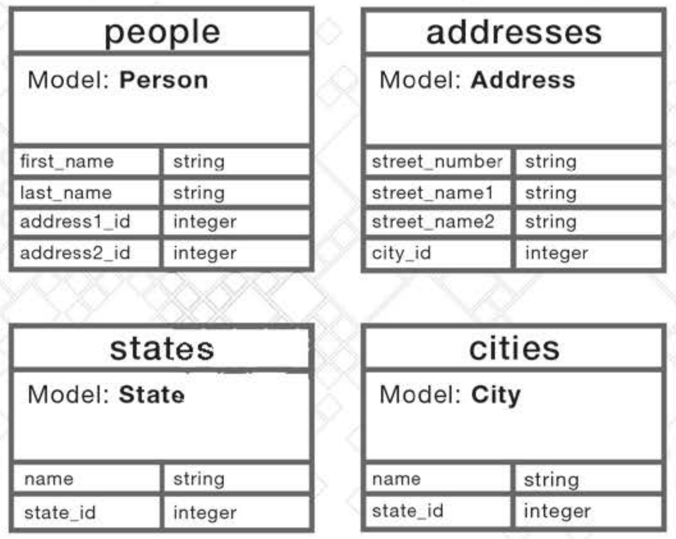
MarkLogic

Learning Objectives

- Describe data modeling concepts of documents, URIs, collections, and directories.
- Load JSON, XML, binary and full text data using the MarkLogic Node.js client API.
- Manage document permissions, collections and quality.
- Ingest data with MarkLogic Content Pump (mlcp).

Types of Data in MarkLogic





people

Model: **Person**

first_name	string
last_name	string
address1_id	integer
address2_id	integer

addresses

Model: **Address**

street_number	string
street_name1	string
street_name2	string
city_id	integer

states

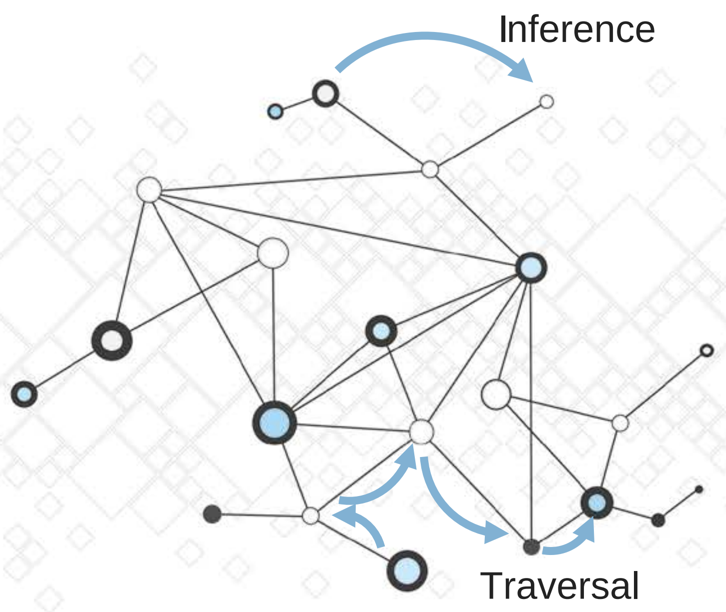
Model: **State**

name	string
state_id	integer

cities

Model: **City**

name	string
state_id	integer



Document Store + Data Store + Triple Store

URIs

- URI = Uniform Resource Identifier
  - Uniquely identifies a document inside of MarkLogic
  - Specified during ingestion
  - Used in CRUD operations

/song/Beatles/Yesterday.json

```
{
"song":
  {
    "artist":  "The Beatles",
    "title":  "Yesterday"
}
```

/book/Melville/MobyDick.xml

```
<book>
  <author>
    H. Melville
  </author>
  <title>
    Moby Dick
  </title>
  <genre>
    Classics
  </genre>
</book>
```

/movie/Spielberg/ET.xml

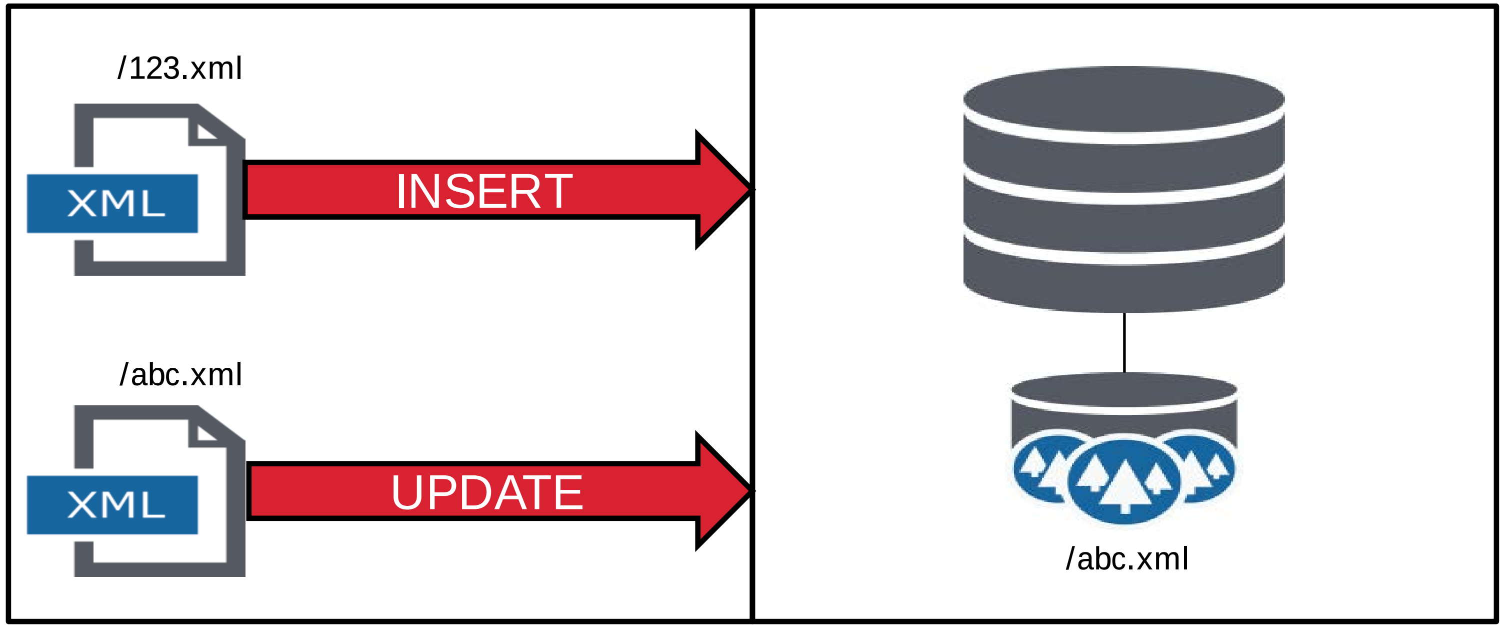
```
<movie>
  <director>
    Steven Spielberg
  </director>
  <title>
    ET
  </title>
  <link>
    /movie/Spielberg/ET.mov
  </link>
</movie>
```

/movie/Spielberg/ET.mov

MarkLogic

URIs Continued

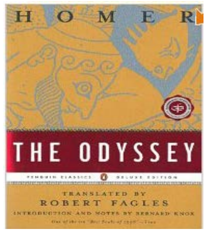
- Insert: Loading a document at a URI that **does not exist** in the database
- Update: Loading a document at a URI that **already exists** in the database



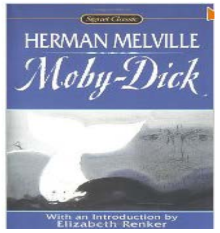
Document Metadata

Collections

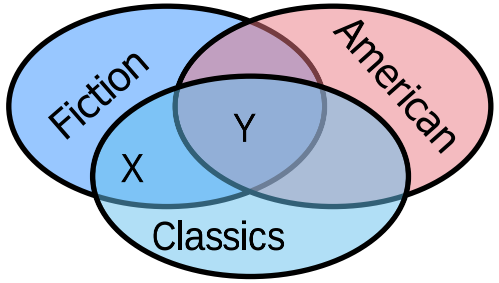
- Way of organizing docs in a database
- Non-hierarchical
- A doc can belong to zero, one or many



URI=X



URI=Y



Permissions

- Security is based on assignment of roles
- Permissions define what can be done with a document

- Execute
- Insert
- Read
- Update
  - To delete a document the update permission is required.

Properties

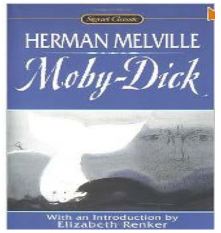
- Extracted metadata from binary docs
- Additional metadata you wish to include
- Stored as XML

Metadata for a JPG may include:

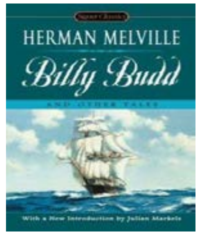
- Caption
- Subjects
- Photographer
- Geospatial Location
- Dimensions
- Camera Settings

Quality

- Affect relevance ranking of documents in search results
- Default quality = 0



Q = 1



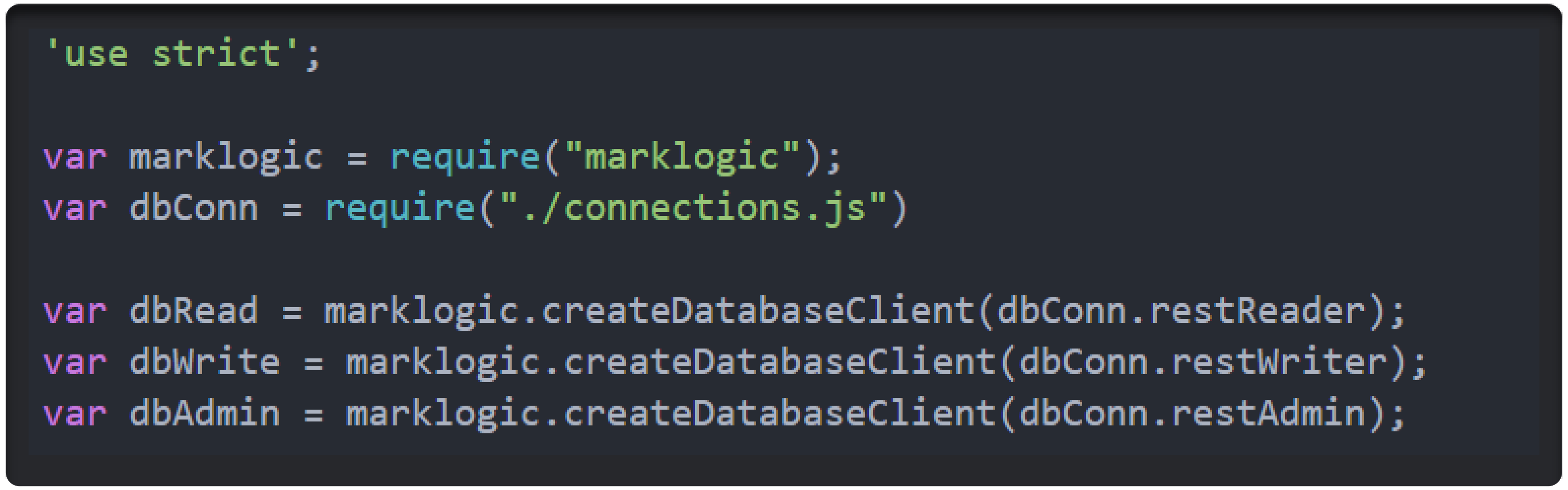
Q = 0

Find fiction books written by the author Herman Melville.

1. Moby Dick
2. Billy Budd

Creating Database Clients

- To save valuable screen space we'll omit this code in the next few examples.
- But remember, we do need this in our project:



```
'use strict';

var marklogic = require("marklogic");
var dbConn = require("./connections.js")

var dbRead = marklogic.createDatabaseClient(dbConn.restReader);
var dbWrite = marklogic.createDatabaseClient(dbConn.restWriter);
var dbAdmin = marklogic.createDatabaseClient(dbConn.restAdmin);
```

MarkLogic

Writing a Document

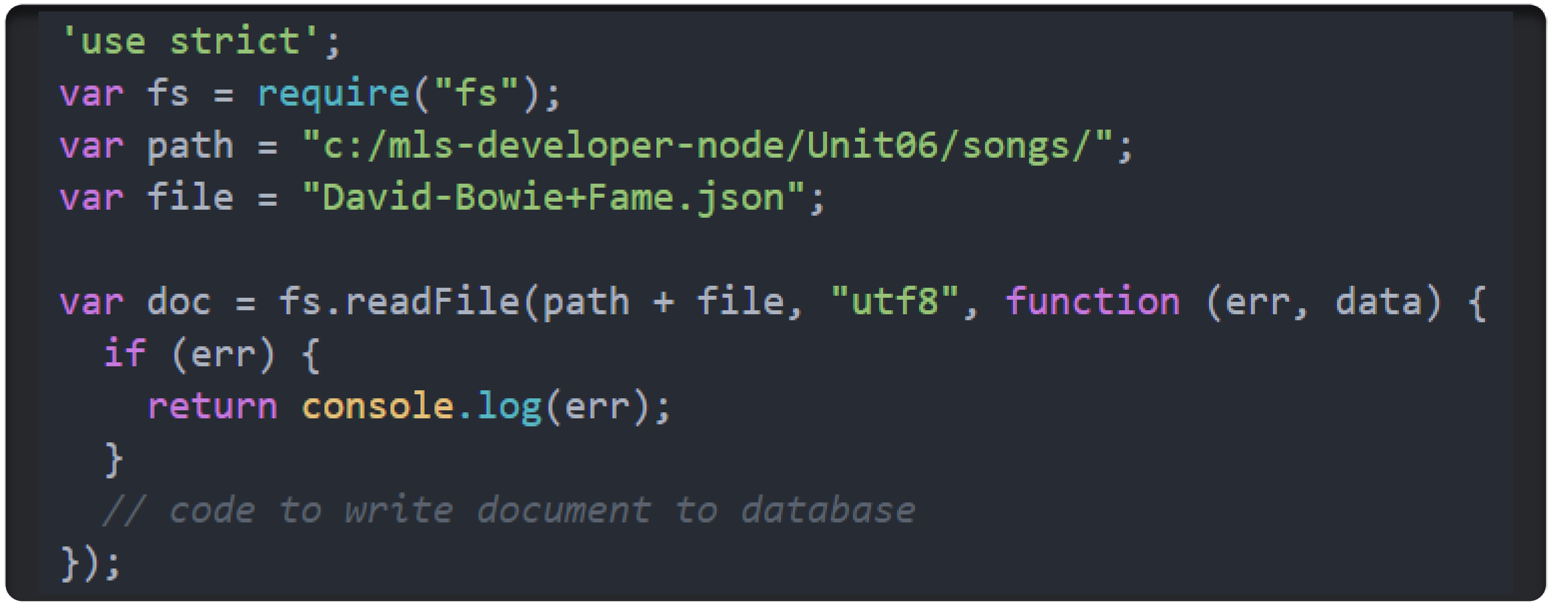
- This example:
  - Builds a doc in memory
    - Document descriptor
  - Writes the document
    - Notice the use of dbWrite
  - Reads the document
  - Outputs some information

- What type of result handler?
  - Why does this matter here?

```
var uri = "/songs/song3.json";
var doc = [
  { "uri": uri,
    "contentType": "application/json",
    "content": { "top-song": { "title": "My New Song", "artist": "My Name" } }
  }
];

dbWrite.documents.write(doc).result().then(
  function(response){
    console.log("Finished with write");

    dbRead.documents.read(uri).result(
      function(documents){
        documents.forEach(function(document){
          console.log("URI=" + document.uri);
          console.log("DOCUMENT=" + JSON.stringify(document.content));
        });
      },
      function(error){
        console.log(JSON.stringify(error, null, 2));
      }
    );
  },
  function(error) {
      console.log(JSON.stringify(error));
    }
);
```

Writing a Document from the File System

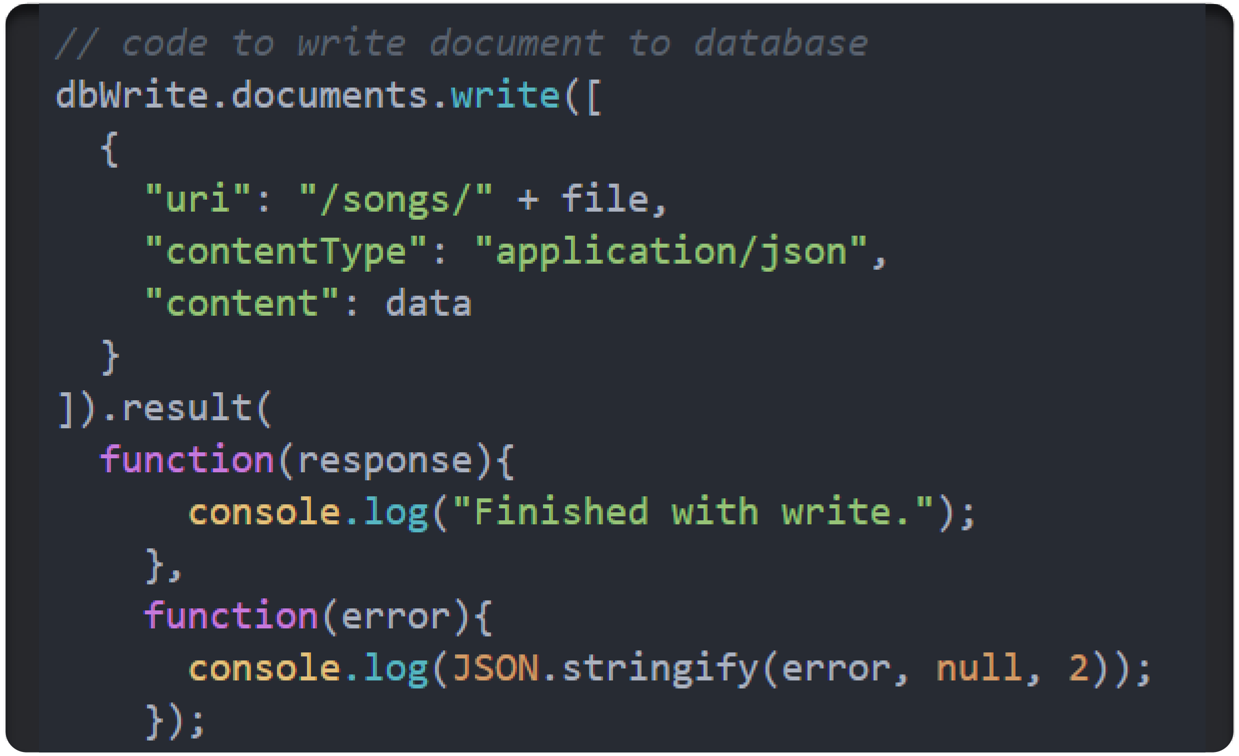
- Step 1:
  - Use fs, the standard Node.js API for file I/O (https://nodejs.org/api/fs.html).
  - Read the desired document from the file system.
    - Document from file system is now stored in **data** variable.



```javascript
'use strict';
var fs = require("fs");
var path = "c:/mls-developer-node/Unit06/songs/";
var file = "David-Bowie+Fame.json";

var doc = fs.readFile(path + file, "utf8", function (err, data) {
  if (err) {
    return console.log(err);
  }
  // code to write document to database
});
```

Writing a Document from the File System

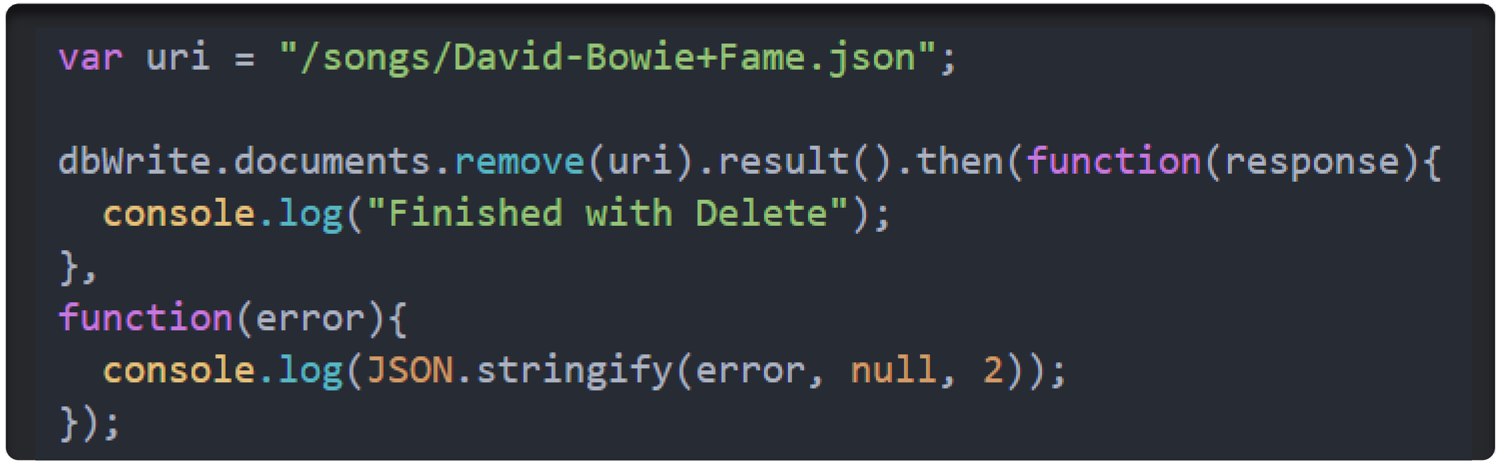
- Step 2:
  - Build a document descriptor, setting content equal to doc read from file system.



```
// code to write document to database
dbWrite.documents.write([
  {
    "uri": "/songs/" + file,
    "contentType": "application/json",
    "content": data
  }
]).result(
  function(response){
      console.log("Finished with write.");
    },
    function(error){
      console.log(JSON.stringify(error, null, 2));
    });
```

Deleting a Document

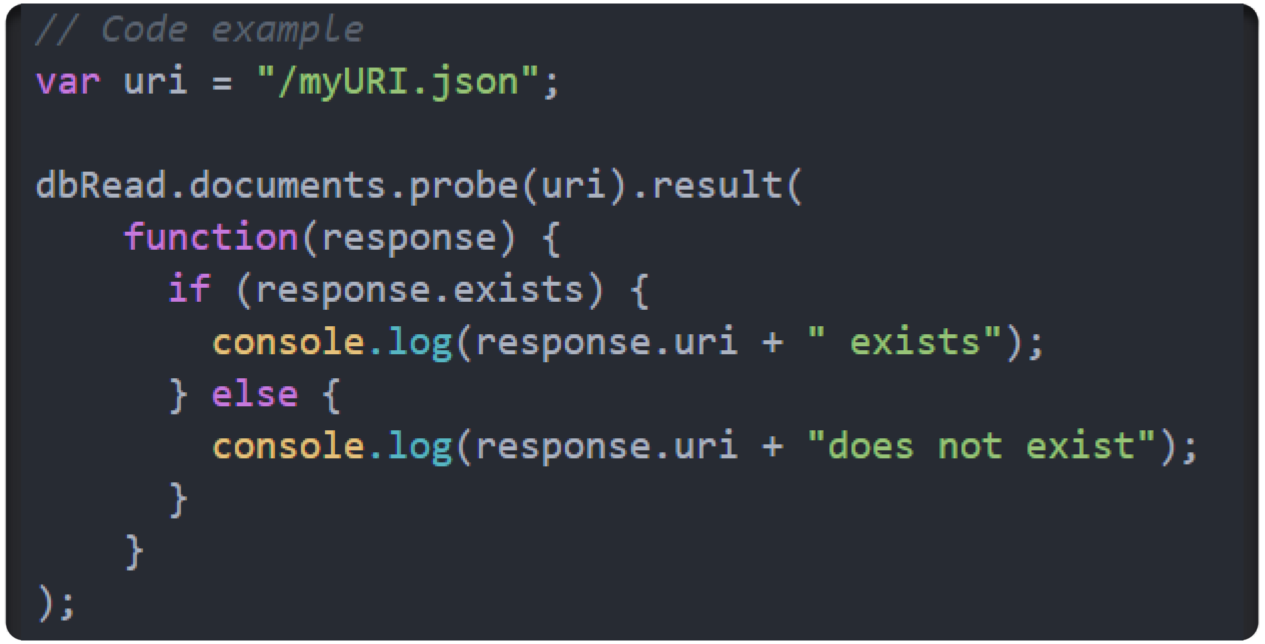
- This example removes (deletes) a specific document URI.
  - Note:
    - Remove is used in the MarkLogic API syntax so as not to conflict / confuse people with the standard JavaScript delete operator, which is used to remove a property from an object.



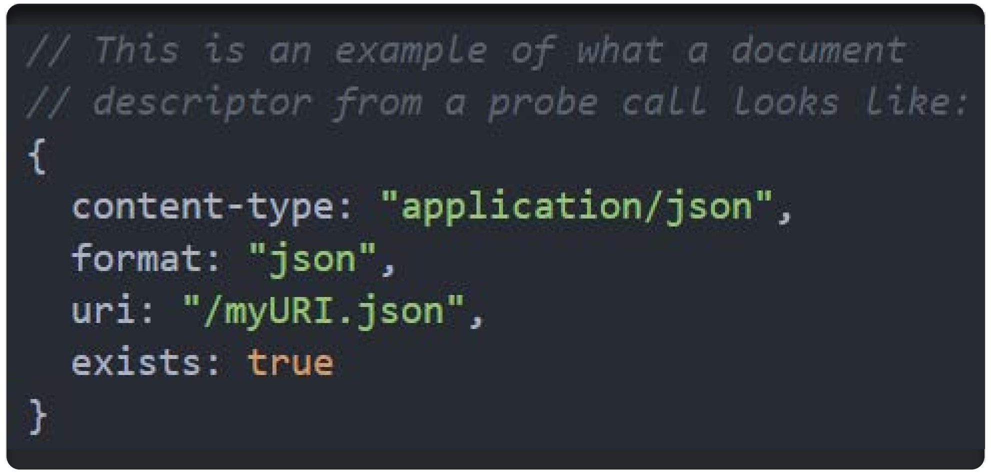
```
var uri = "/songs/David-Bowie+Fame.json";

dbWrite.documents.remove(uri).result().then(function(response){
  console.log("Finished with Delete");
},
function(error){
  console.log(JSON.stringify(error, null, 2));
});
```

Probe

- A lightweight check to see if a URI exists.
- Useful if your app needs to adapt based on existence.
- A probe call returns a document descriptor, which we then check for existence.



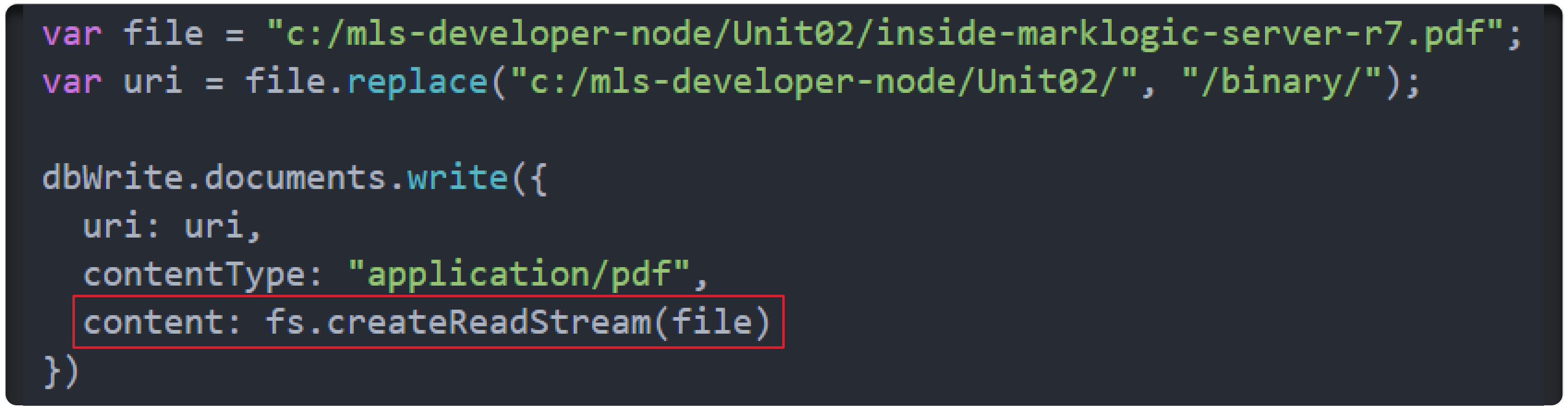
```
// Code example
var uri = "/myURI.json";

dbRead.documents.probe(uri).result(
    function(response) {
      if (response.exists) {
        console.log(response.uri + " exists");
      } else {
        console.log(response.uri + "does not exist");
      }
    }
);
```



```
// This is an example of what a document
// descriptor from a probe call looks like:
{
  content-type: "application/json",
  format: "json",
  uri: "/myURI.json",
  exists: true
}
```

Writing a Document: Streaming

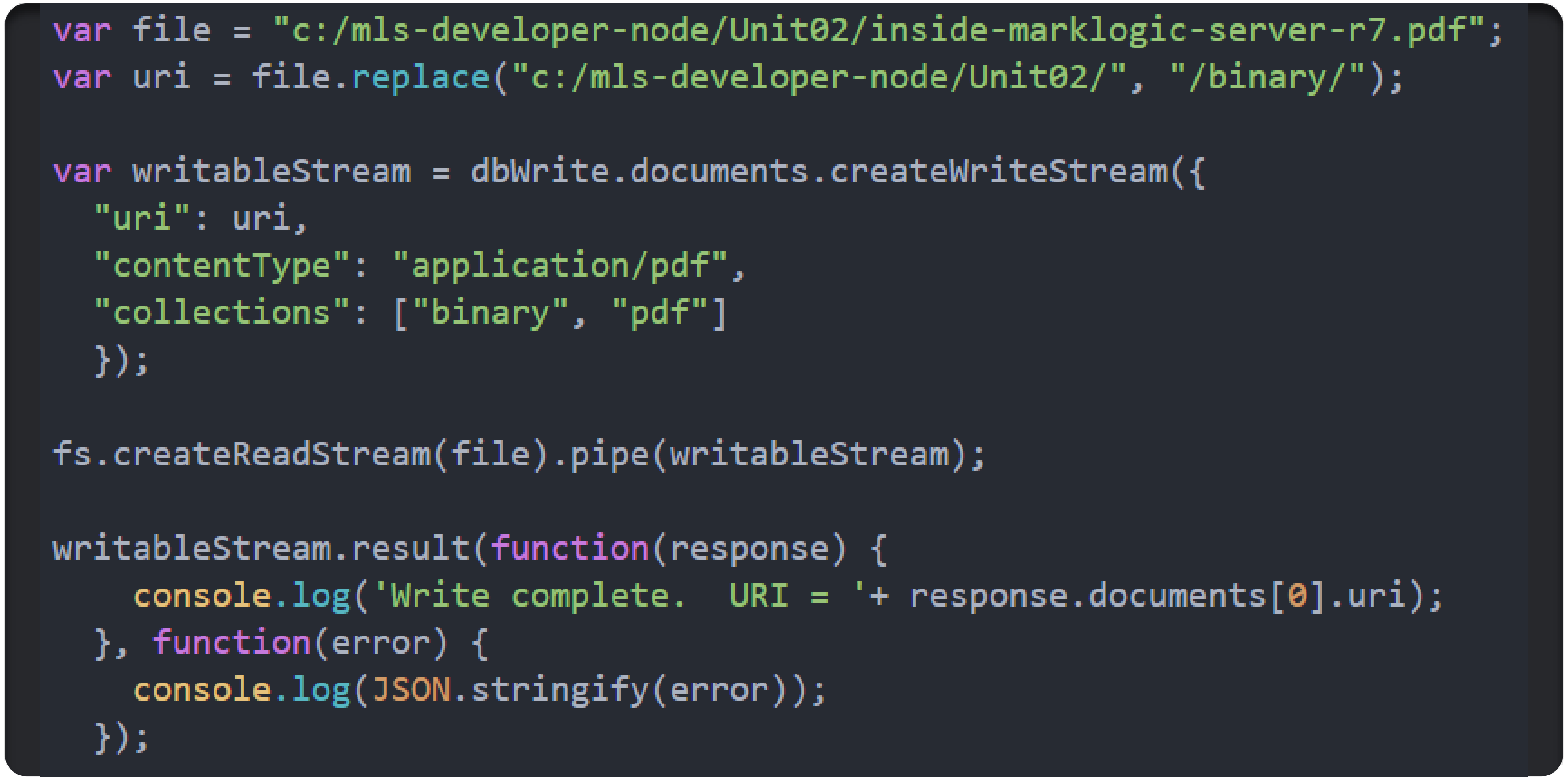
- For larger inputs such as binary documents.
- Pass in the data to the document descriptor as a **ReadableStream**.



```
var file = "c:/mls-developer-node/Unit02/inside-marklogic-server-r7.pdf";
var uri = file.replace("c:/mls-developer-node/Unit02/", "/binary/");

dbWrite.documents.write({
  uri: uri,
  contentType: "application/pdf",
  content: fs.createReadStream(file)
})
```

Writing a Document: Streaming

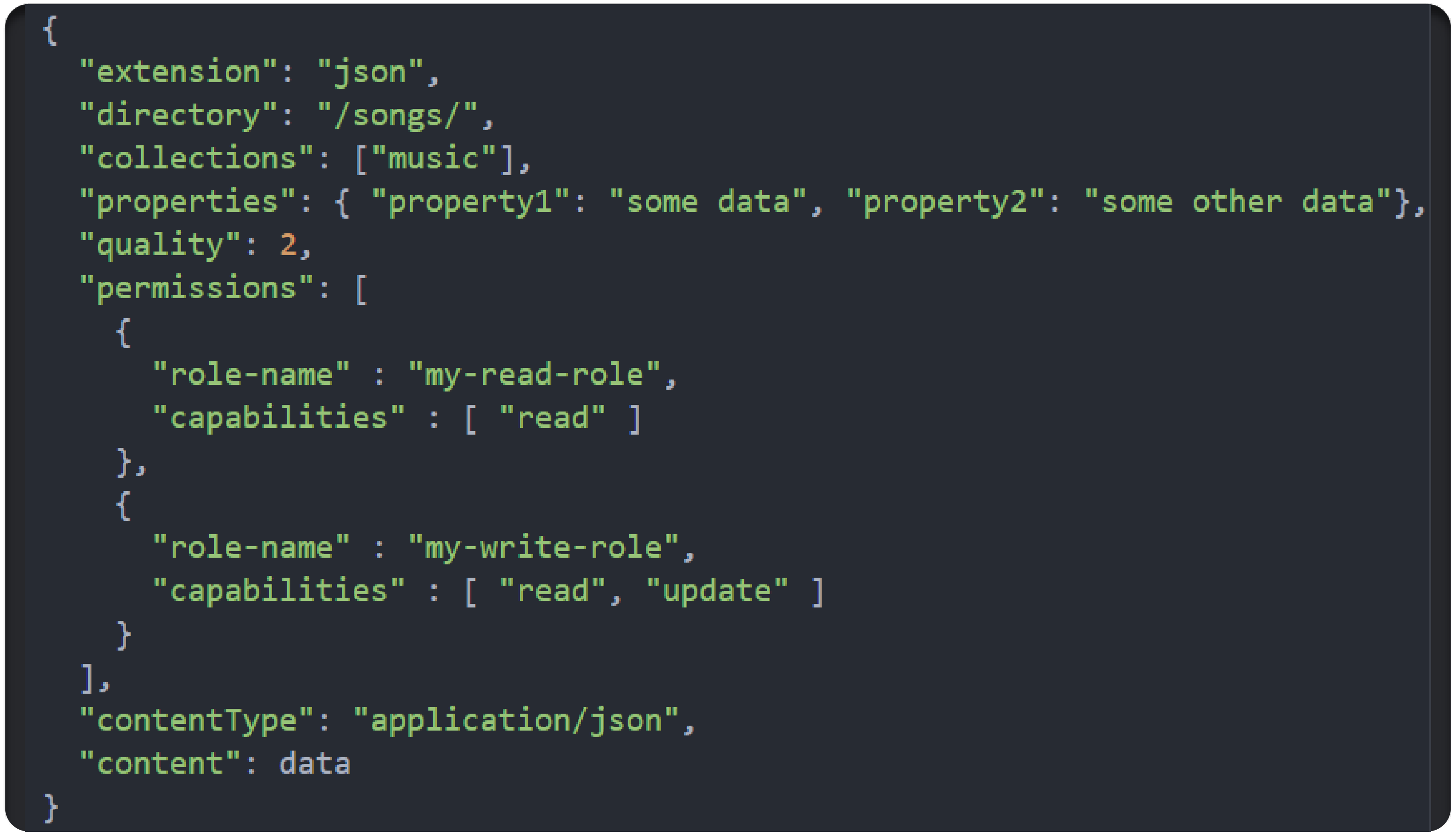
- Or use **.createWriteStream** if you need more control:



```
var file = "c:/mls-developer-node/Unit02/inside-marklogic-server-r7.pdf";
var uri = file.replace("c:/mls-developer-node/Unit02/", "/binary/");

var writableStream = dbWrite.documents.createWriteStream({
  "uri": uri,
  "contentType": "application/pdf",
  "collections": ["binary", "pdf"]
  });

fs.createReadStream(file).pipe(writableStream);

writableStream.result(function(response) {
    console.log('Write complete.  URI = '+ response.documents[0].uri);
  }, function(error) {
    console.log(JSON.stringify(error));
  });
```

Managing Document Metadata

- It's all in the document descriptor:

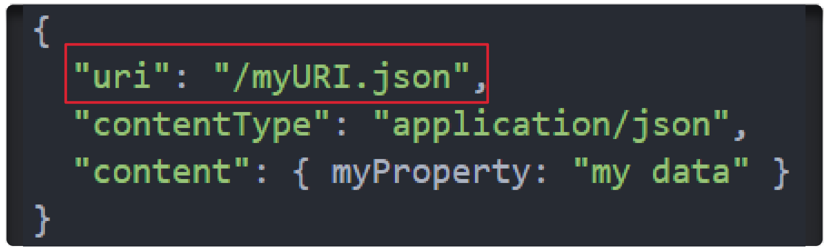


```
{
  "extension": "json",
  "directory": "/songs/",
  "collections": ["music"],
  "properties": { "property1": "some data", "property2": "some other data"},
  "quality": 2,
  "permissions": [
    {
      "role-name" : "my-read-role",
      "capabilities" : [ "read" ]
    },
    {
      "role-name" : "my-write-role",
      "capabilities" : [ "read", "update" ]
    }
  ],
  "contentType": "application/json",
  "content": data
}
```

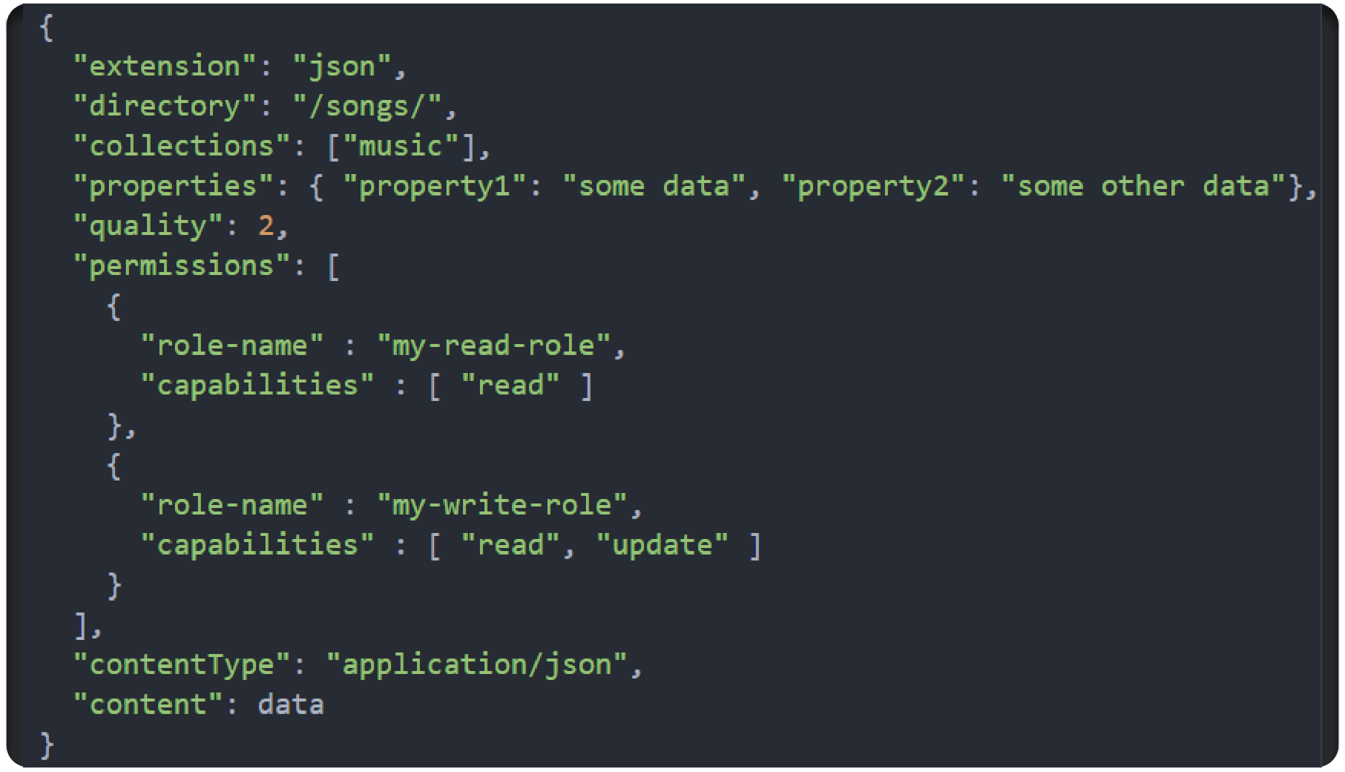
MarkLogic

Managing Document Metadata

- In prior examples, our document descriptors had a **uri** property.
- This example has no uri property.
- Question:
  - What will the URI be if we load the document descriptor in the example on the right, which does not cointain a uri property?



```
{
  "uri": "/myURI.json",
  "contentType": "application/json",
  "content": { myProperty: "my data" }
}
```

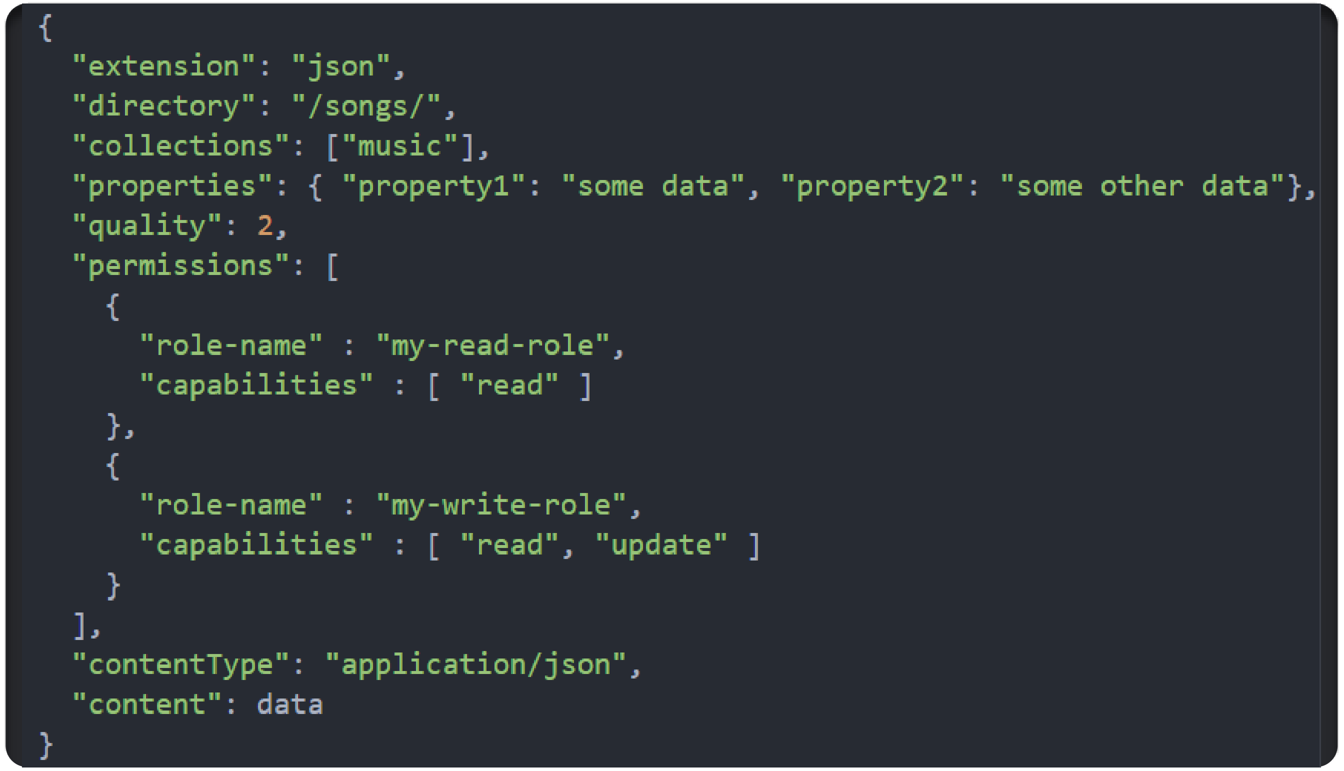


```
{
  "extension": "json",
  "directory": "/songs/",
  "collections": ["music"],
  "properties": { "property1": "some data", "property2": "some other data"},
  "quality": 2,
  "permissions": [
    {
      "role-name" : "my-read-role",
      "capabilities" : [ "read" ]
    },
    {
      "role-name" : "my-write-role",
      "capabilities" : [ "read", "update" ]
    }
  ],
  "contentType": "application/json",
  "content": data
}
```

MarkLogic

Managing Document Metadata

- Question:
  - What will the URI be if we load the document descriptor in the example on the right, which does not cointain a uri property?
- Answer:
  - The URI will be automatically generated, using the extension and directory properties as we have defined, and a 64 bit long integer random number.
  - /songs/################.json



```
{
  "extension": "json",
  "directory": "/songs/",
  "collections": ["music"],
  "properties": { "property1": "some data", "property2": "some other data"},
  "quality": 2,
  "permissions": [
    {
      "role-name" : "my-read-role",
      "capabilities" : [ "read" ]
    },
    {
      "role-name" : "my-write-role",
      "capabilities" : [ "read", "update" ]
    }
  ],
  "contentType": "application/json",
  "content": data
}
```

Labs: Unit 6

Exercise 1 - Exercise 9: Loading and Managing Data Using the Node.js API

Stop when finished with Exercise 9

MarkLogic Content Pump

MarkLogic Content Pump (mlcp) is a command line tool:

- Load content into a MarkLogic database
  - JSON, XML, binary, RDF and full text
  - compressed ZIP and GZIP files
  - mlcp database archives
  - Hadoop sequence files
- Export the contents of a MarkLogic database
  - native file format
  - compressed ZIP file
  - mlcp archive
- Copy documents and metadata between two databases

Benefits of using mlcp

- Improves performance and reliability of ingestion workflows
  - Bulk load billions of local files
  - Split and load large aggregate files or delimited text
- Better integrates with other tools and environments
  - Load documents from HDFS, including Hadoop Sequence Files
  - Archive and restore database contents across environments
  - Copy subsets of data between databases

mlcp Operational Modes

- Local
  - Local file system
  - MarkLogic database
  - Parallelizes I/O processing over multiple threads

- Disributed
  - HDFS
  - Parallelizes I/O across multiple hosts in Hadoop Cluster

MarkLogic

mlcp Command Line Syntax

Windows

```
mlcp.bat import ^
     -host localhost      -port 8012 ^
        -username admin   -password admin ^
        -input_file_path C:\mlcp-data\socialmedia\content ^
        -mode local ^
        -input_file_pattern "twitter.*\.xml " ^
        -output_uri_replace "C:/mlcp-data/socialmedia/content, 'socialmedia'"
```

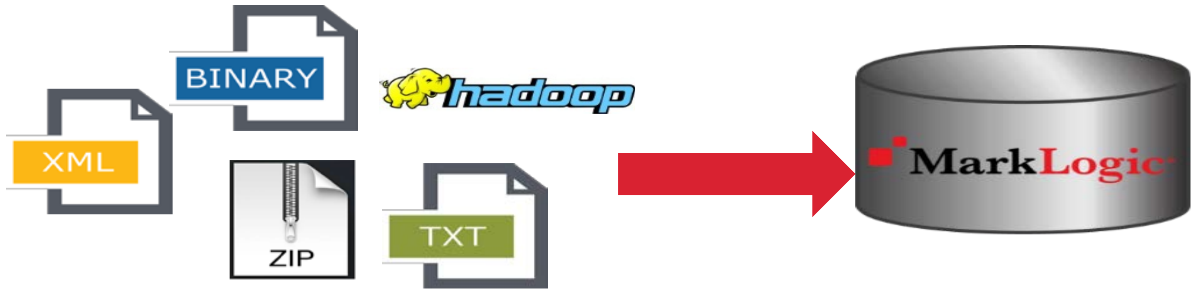
Linux, Solaris, and OS X

```
    mlcp.sh import \
      -host localhost     -port 8012 \
        -username admin   -password ****\
        -input_file_path C:/mlcp-data/socialmedia/content \
        -mode local \
        -input_file_pattern 'twitter.*\.xml' \
        -output_uri_replace "C:/mlcp-data/socialmedia/content, 'socialmedia'"
```

MarkLogic

mlcp Import

- JSON, XML, RDF, binary, text
- Aggregate XML (automated split capability)
- Compressed ZIP and GZIP files
- MarkLogic database archives
- Hadoop sequence files



MarkLogic

mlcp Import Example

```
mlcp.bat import ^
  -host localhost  -port 8012 ^
  -username admin  -password admin ^
  -input_file_path C:\mlcp-data\socialmedia\content ^
  -mode local ^
  -input_file_pattern "twitter.*\.xml" ^
  -output_uri_replace "C:/mlcp-data/socialmedia/content,'socialmedia‘" ^
  -output_directory  twitter
```

```
<MedlineCitationSet>
   <MedlineCitation Owner="NLM" Status="Completed">
   <MedlineID>21978177</MedlineID>
   <PMID>11981951</PMID>
   <DateCreated><Year>2002</Year><Month>04</Month><Day>30</Day></DateCreated>
   ...
   </MedlineCitation>
   <MedlineCitation Owner="HSR" Status="Completed">
   <MedlineID>21978178</MedlineID>
   <PMID>11982031</PMID>
   <DateCreated><Year>2002</Year><Month>04</Month><Day>30</Day></DateCreated>
   ...
   </MedlineCitation>
   <MedlineCitation Owner="NLM" Status="Completed">
   <MedlineID>21978179</MedlineID>
   <PMID>11981952</PMID>
   <DateCreated><Year>2002</Year><Month>04</Month><Day>30</Day></DateCreated>
   ...
   </MedlineCitation>
...
```

Aggregate record element

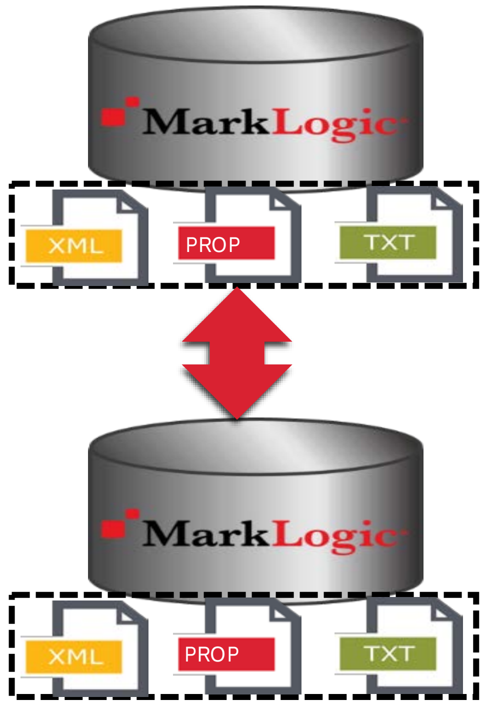
Aggregate URI ID

mlcp Splitting Aggregate Documents

```
mlcp.bat import ^
 -host localhost -port 8021 ^
 -username admin -password admin ^
 -mode local ^
 -input_file_path C:\medline\medline.xml ^
 -input_file_type aggregates
 -aggregate_record_element MedlineCitation ^
 -aggregate_uri_id MedlineID ^
 -output_uri_prefix /journal/MedlineID ^
 -output_uri_suffix .xml ^
 -output_collections published
```

mlcp Copy

- Across environments
- No intermediate copy required
- Subsets of data or all content
- Add or override document metadata in the destination database

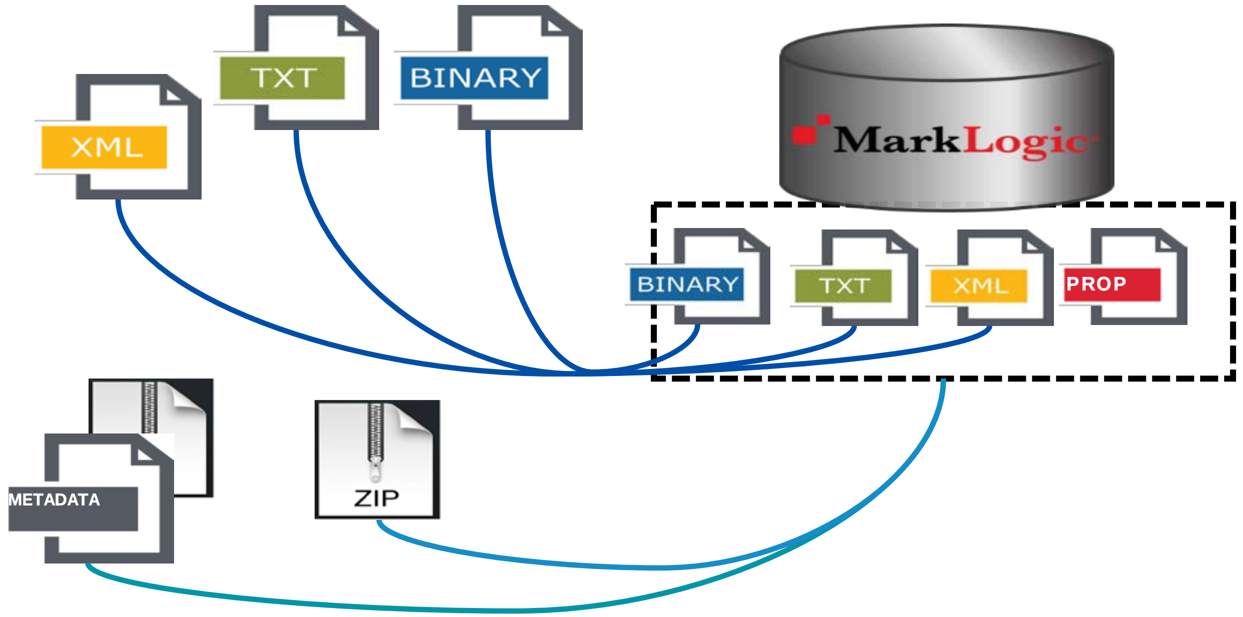


mlcp Copy Example

```
mlcp.bat copy ^
  -input_host source.example.com -input_port 5275 ^
  -input_username reader -input_password password ^
  -collection_filter medicine
  -output_host dest.example.com -output_port 9987 ^
  -output_username writer -output_password password ^
  -copy_permissions false ^
  -output_collections biomedicine,health
```

mlcp Export

- Export all database content or subset of data as:
  - Files in their original format
  - Compressed ZIP files
  - MarkLogic database archive



mlcp Export Example

```
mlcp.bat export ^
    -host localhost -port 8012 ^
    -username admin -password admin ^
    -mode local ^
    -output_file_path /Social_Media/Sentiment ^
    -output_type archive ^
    -copy_permissions false ^
    -directory_filter /twitter/
```

Resources

- Download mlcp
  - http://developer.marklogic.com/
  - Note:
    - It’s already on your training VM @ c:\mlcp

- mlcp Documentation:
  - http://docs.marklogic.com/

Labs: Unit 6

Exercise 10: Create an XDBC Application Server

Exercise 11: Load Data with MarkLogic Content Pump (mlcp)

DIY: Load the Star Wars Data

Unit Review Question 1:

A document descriptor is:

1. A JSON object
2. A way to define document data and metadata
3. Used when performing a read
4. Used when performing a write
5. All of the above

Unit Review Question 1:

A document descriptor is:

1. A JSON object
2. A way to define document data and metadata
3. Used when performing a read
4. Used when performing a write
5. **All of the above**

Unit Review Question 2:

To write a document to the database, the database client must authenticate as:

1. A user with a role with insert permissions on the database
2. A user with the rest-writer role
3. A user with the MarkLogic admin role
4. Both 1 and 2 are correct

Unit Review Question 2:

To write a document to the database, the database client must authenticate as:

1. A user with a role with insert permissions on the database
2. A user with the rest-writer role
3. A user with the MarkLogic admin role
4. **Both 1 and 2 are correct**

Unit Review Question 3:

Properties metadata about a document is stored in the database:

1. As a separate fragment (document) in the database
2. Embedded in the document as header information
3. Emdedded in the document in a <properties> element
4. None of the above

Unit Review Question 3:

Properties metadata about a document is stored in the database:

1. **As a separate fragment (document) in the database**
2. Embedded in the document as header information
3. Emdedded in the document in a <properties> element
4. None of the above

Unit Review Question 4:

MarkLogic Content Pump (mlcp) is designed to perform:

1. Load, copy, export
2. Load only…but fast
3. Load, export, entity enrichment
4. None of the above

Unit Review Question 4:

MarkLogic Content Pump (mlcp) is designed to perform:

1. **Load, copy, export**
2. Load only…but fast
3. Load, export, entity enrichment
4. None of the above