Unit 5

Using the Node.js Client API

Install the MarkLogic Node.js Client API

Define Project Database Connections

Read a JSON Document Using a Callback

Read an XML Document Using a Callback

Read Documents Using Promises

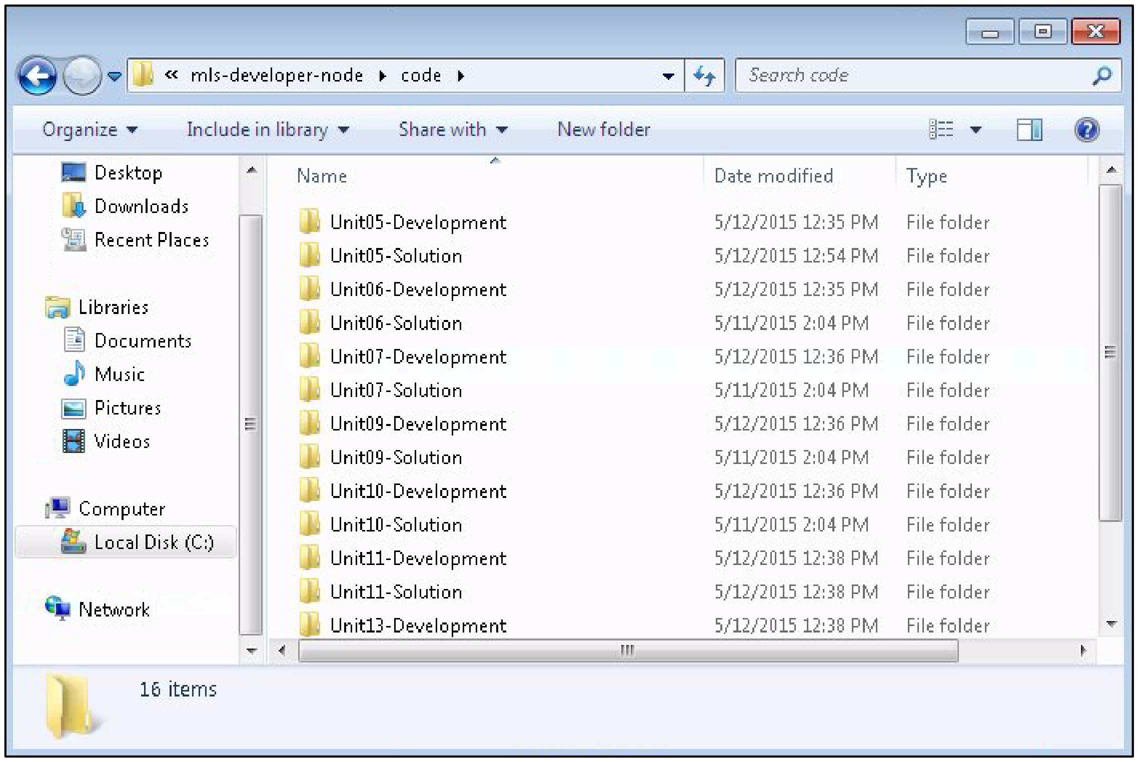
Read Documents Using Streams

DIY: Create a Database Client

Exercise 1: Install the MarkLogic Node.js Client API

In this exercise you will install the MarkLogic Node.js client API for a project using NPM.

1. In Windows Explorer, view the contents at **c:\mls-developer-node\code\**
2. Notice that for each Unit there is a **Development** project and a **Solution** project:



Name	Date modified	Type
Unit05-Development	5/12/2015 12:35 PM	File folder
Unit05-Solution	5/12/2015 12:54 PM	File folder
Unit06-Development	5/12/2015 12:35 PM	File folder
Unit06-Solution	5/11/2015 2:04 PM	File folder
Unit07-Development	5/12/2015 12:36 PM	File folder
Unit07-Solution	5/11/2015 2:04 PM	File folder
Unit09-Development	5/12/2015 12:36 PM	File folder
Unit09-Solution	5/11/2015 2:04 PM	File folder
Unit10-Development	5/12/2015 12:36 PM	File folder
Unit10-Solution	5/11/2015 2:04 PM	File folder
Unit11-Development	5/12/2015 12:38 PM	File folder
Unit11-Solution	5/12/2015 12:38 PM	File folder
Unit13-Development	5/12/2015 12:38 PM	File folder

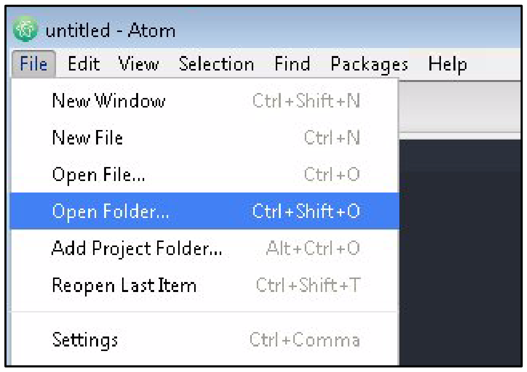
16 items

Note:
The solution project is there for you as a reference should you need to consult a working example.

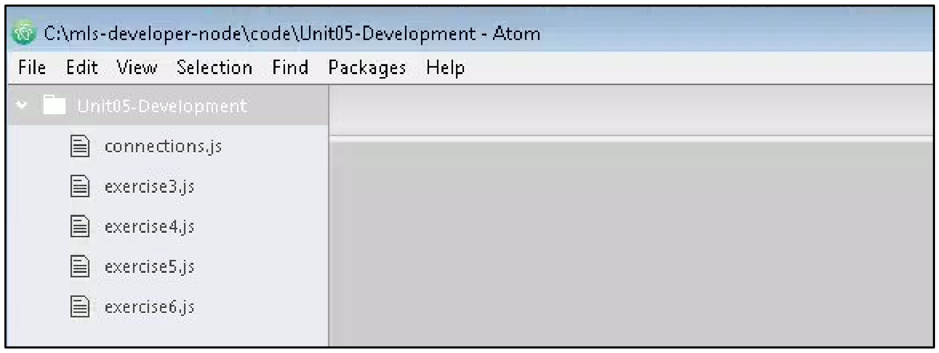
The development project is there for you to have an area for experimentation and writing your own code.

3. Open the Atom editor from your Desktop:

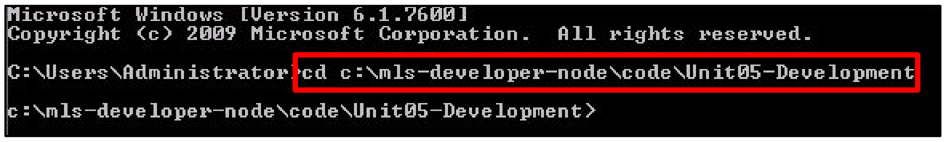
4. In the Atom editor select File→Open Folder…:


untitled - Atom
File Edit View Selection Find Packages Help
New Window Ctrl+Shift+N
New File Ctrl+N
Open File... Ctrl+O
Open Folder... Ctrl+Shift+O
Add Project Folder... Alt+Ctrl+O
Reopen Last Item Ctrl+Shift+T
Settings Ctrl+Comma

5. Open the **Unit05-Development** folder from **c:\mls-developer-node\code\**
6. You should see the following structure in place in your editor:

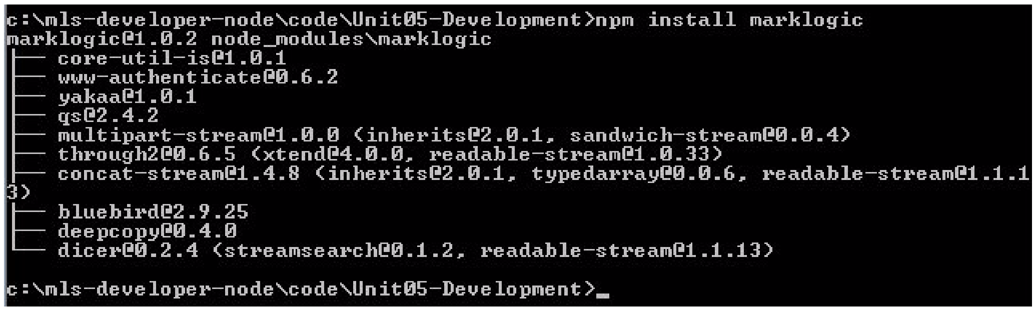

C:\mls-developer-node\code\Unit05-Development - Atom
File Edit View Selection Find Packages Help
Unit05-Development
connections.js
exercise3.js
exercise4.js
exercise5.js
exercise6.js

7. Open a command prompt.
8. Enter the highlighted command to navigate to the **Unit05-Development** folder:



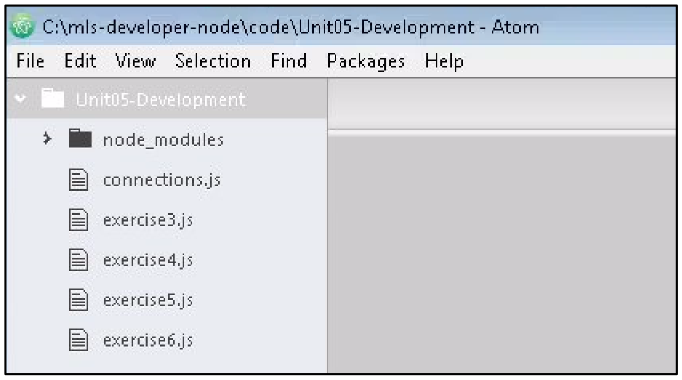
```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Administrator>cd c:\mls-developer-node\code\Unit05-Development

c:\mls-developer-node\code\Unit05-Development>
```

9. Type **npm install marklogic** and press enter to install the MarkLogic Node.js client API:



```
c:\mls-developer-node\code\Unit05-Development>npm install marklogic
marklogic@1.0.2 node_modules\marklogic
├── core-util-is@1.0.1
├── www-authenticate@0.6.2
├── yakaa@1.0.1
├── qs@2.4.2
├── multipart-stream@1.0.0 (inherits@2.0.1, sandwich-stream@0.0.4)
├── through2@0.6.5 (xtend@4.0.0, readable-stream@1.0.33)
├── concat-stream@1.4.8 (inherits@2.0.1, typedarray@0.0.6, readable-stream@1.1.1
3)
├── bluebird@2.9.25
├── deepcopy@0.4.0
└── dicer@0.2.4 (streamsearch@0.1.2, readable-stream@1.1.13)

c:\mls-developer-node\code\Unit05-Development>_
```

10. Notice that the **/node_modules/** directory is now a part of your project:



C:\mls-developer-node\code\Unit05-Development - Atom

```
Unit05-Development
  node_modules
  connections.js
  exercise3.js
  exercise4.js
  exercise5.js
  exercise6.js
```

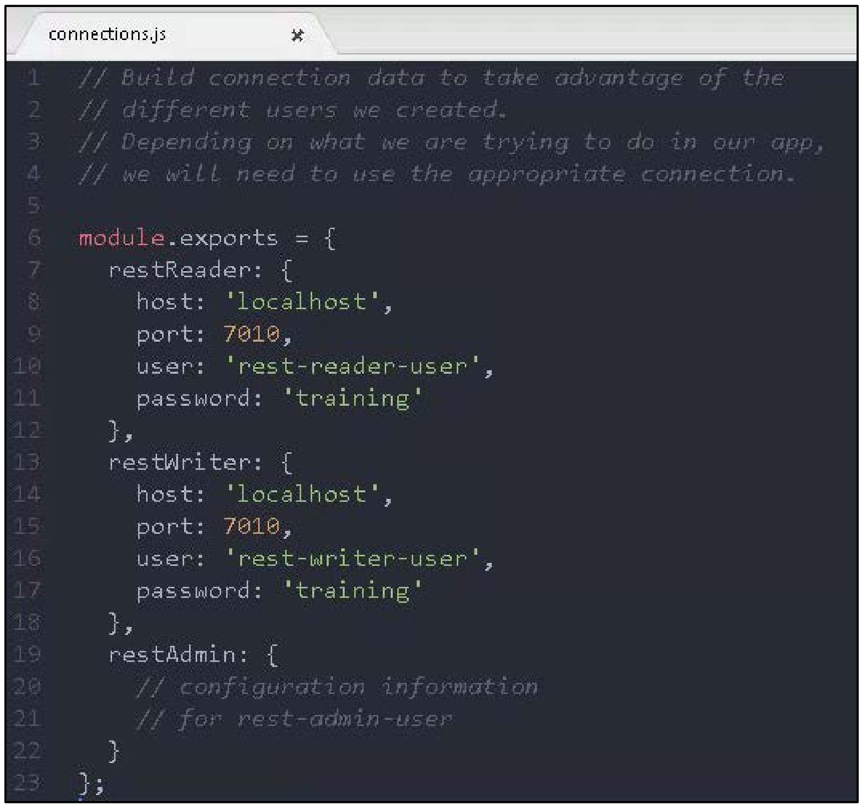
Exercise 2: Define Project Database Connections

In this exercise you will setup database connection details for each of the three user accounts that you created during the security configuration exercises.

This will enable you to write project code using the appropriate database connection for the task that you are trying to perform.

For example, if your goal is to search the database you would use the connection associated with the user and role that has read permissions. If your goal is to insert new data or update existing data, you would use the connection associated with the user and role that has update permissions.

1. Within the **Unit05-Development** project in your editor, view **connections.js**:



connections.js

```
// Build connection data to take advantage of the
// different users we created.
// Depending on what we are trying to do in our app,
// we will need to use the appropriate connection.

module.exports = {
  restReader: {
    host: 'localhost',
    port: 7010,
    user: 'rest-reader-user',
    password: 'training'
  },
  restWriter: {
    host: 'localhost',
    port: 7010,
    user: 'rest-writer-user',
    password: 'training'
  },
  restAdmin: {
    // configuration information
    // for rest-admin-user
  }
};
```

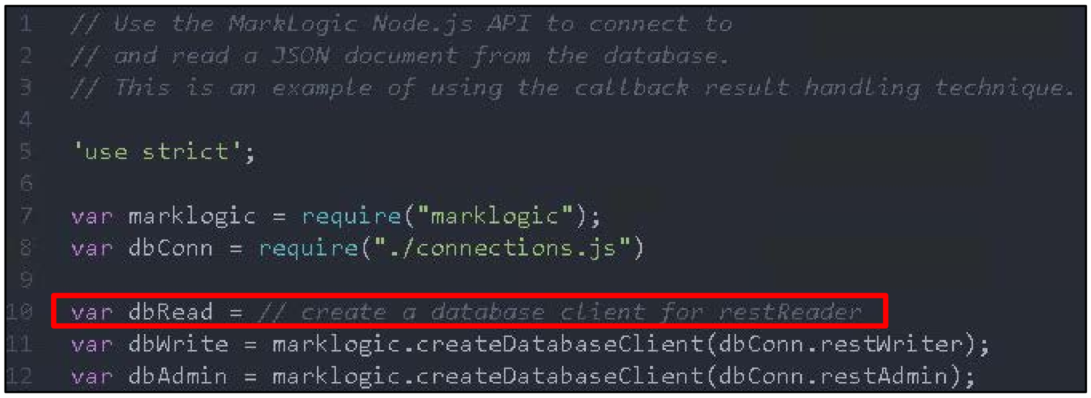
2. The restReader and restWriter have already been setup. Locate the incomplete setup for **restAdmin**. Follow the pattern used to define restReader and restWriter but substituting the appropriate account information for the rest-admin-user that you created earlier.

3. If you are unsure of what to do, you may consult your instructor or reference the solution project to see the complete example.

Exercise 3: Read a JSON Document Using a Callback

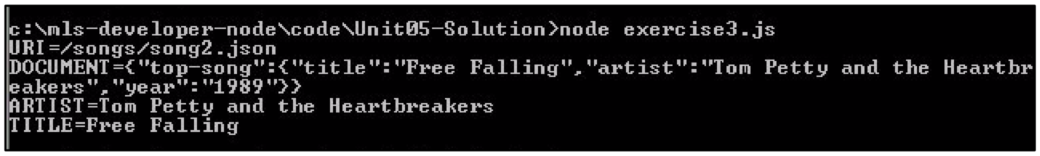
In this exercise you will create a database client using the connection information that you just configured in order to read a JSON document from the database and output information from that document. You will use a callback function to handle the document that is returned from the database.

1. From your editor in the **Unit05-Development** project open **exercise3.js**
2. Since the objective is to read a document, the first thing you will need to do is create a database client with the permissions to read documents.
3. Study the first part of the code. Locate the incomplete **dbRead** variable and edit it to create the database client for the restReader connection details defined in the prior exercise. Use the dbWrite and dbAdmin variables as examples, or consult the solution project if you get stuck:



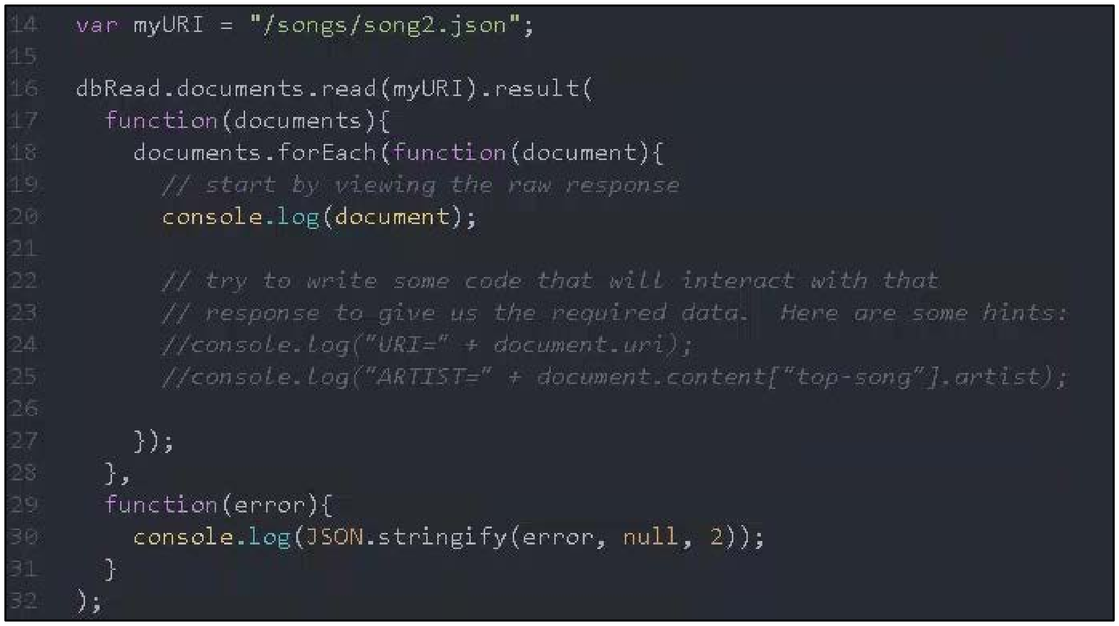
```
// Use the MarkLogic Node.js API to connect to
// and read a JSON document from the database.
// This is an example of using the callback result handling technique.

'use strict';

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

var dbRead = // create a database client for restReader
var dbWrite = marklogic.createDatabaseClient(dbConn.restWriter);
var dbAdmin = marklogic.createDatabaseClient(dbConn.restAdmin);
```

4. Next, let's think about what we are trying to do. The objective is to output information about the document that we read from the database. Specifically we want the document URI, the entire document content, and the value of the title property and the value of the artist property. The end result should look like this:



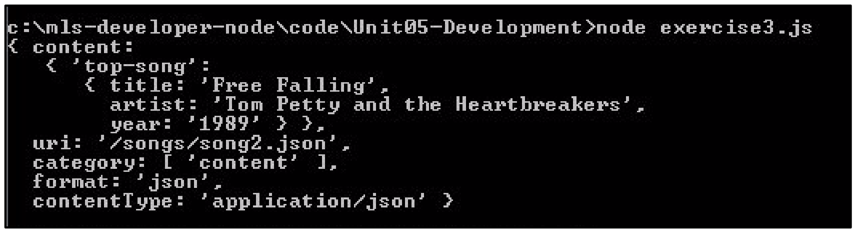
```
c:\mls-developer-node\code\Unit05-Solution>node exercise3.js
URI=/songs/song2.json
DOCUMENT={"top-song":{"title":"Free Falling","artist":"Tom Petty and the Heartbr
eakers","year":"1989"}}
ARTIST=Tom Petty and the Heartbreakers
TITLE=Free Falling
```

5. To get to this end result, we need to learn to interact with the data that comes back to us.
6. Take a minute to study the code in your **Unit05-Development** project for **exercise3.js**:
7. Note that you are using a callback to process the result, and in the callback you are iterating over the results (in case you returned multiple documents, even though this example only returns one):



```
var myURI = "/songs/song2.json";

dbRead.documents.read(myURI).result(
  function(documents){
    documents.forEach(function(document){
      // start by viewing the raw response
      console.log(document);

      // try to write some code that will interact with that
      // response to give us the required data.  Here are some hints:
      //console.log("URI=" + document.uri);
      //console.log("ARTIST=" + document.content["top-song"].artist);

    });
  },
  function(error){
    console.log(JSON.stringify(error, null, 2));
  }
);
```

8. Let's start by running this code to see the output of **console.log(document)**
9. In a command prompt, navigate to your **Unit05-Development** project directory.
10. Key the command **node exercise3.js** and press enter to see the response:



```
c:\mls-developer-node\code\Unit05-Development>node exercise3.js
{ content:
   { 'top-song':
      { title: 'Free Falling',
        artist: 'Tom Petty and the Heartbreakers',
        year: '1989' } },
  uri: '/songs/song2.json',
  category: [ 'content' ],
  format: 'json',
  contentType: 'application/json' }
```

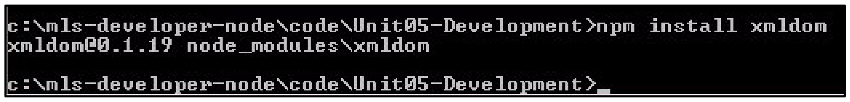
11. The response is JSON, and it's this response that we need to work with to meet the objective.
12. Take some time and try to write code that will work with this response to output the URI, the full document content, and the value of the title and the value of the artist.
13. If you get stuck, please consult your instructor or reference the solution project.

Exercise 4: Read an XML Document Using a Callback

In this exercise you will read an XML document from the database and output information from the document.

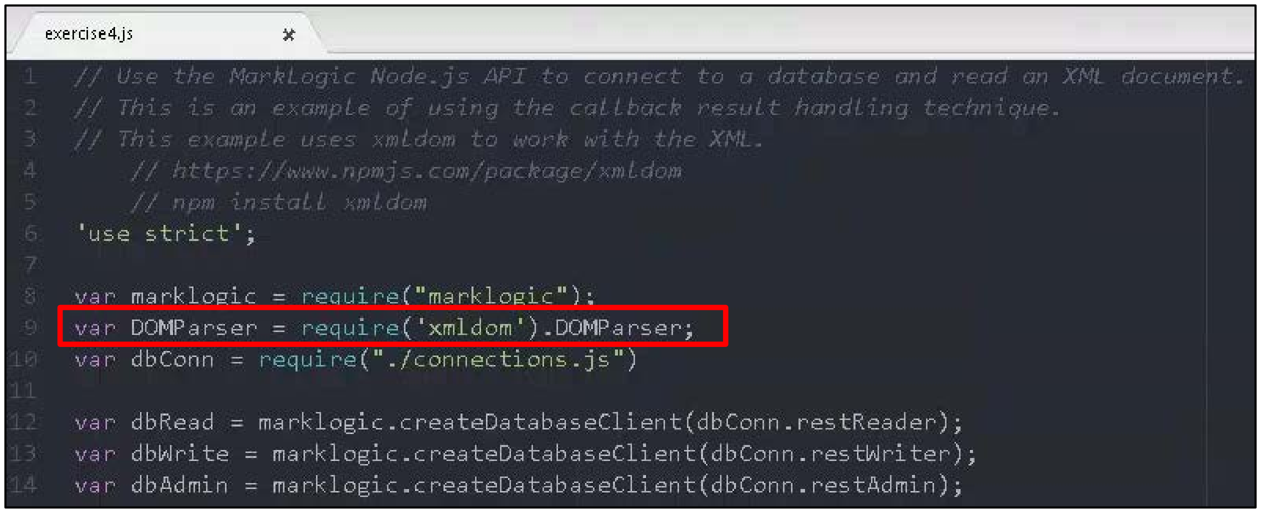
Working with JSON in JavaScript is easy – they were made to work together. In order to use XML, you need to do a little more work in parsing the data using DOM. In order to accomplish that goal, we will install and use another NPM package designed for that purpose.

1. At the command line navigate to your **Unit05-Development** project directory.
2. Key in the command **npm install xmldom** and press enter. When completed you should see:



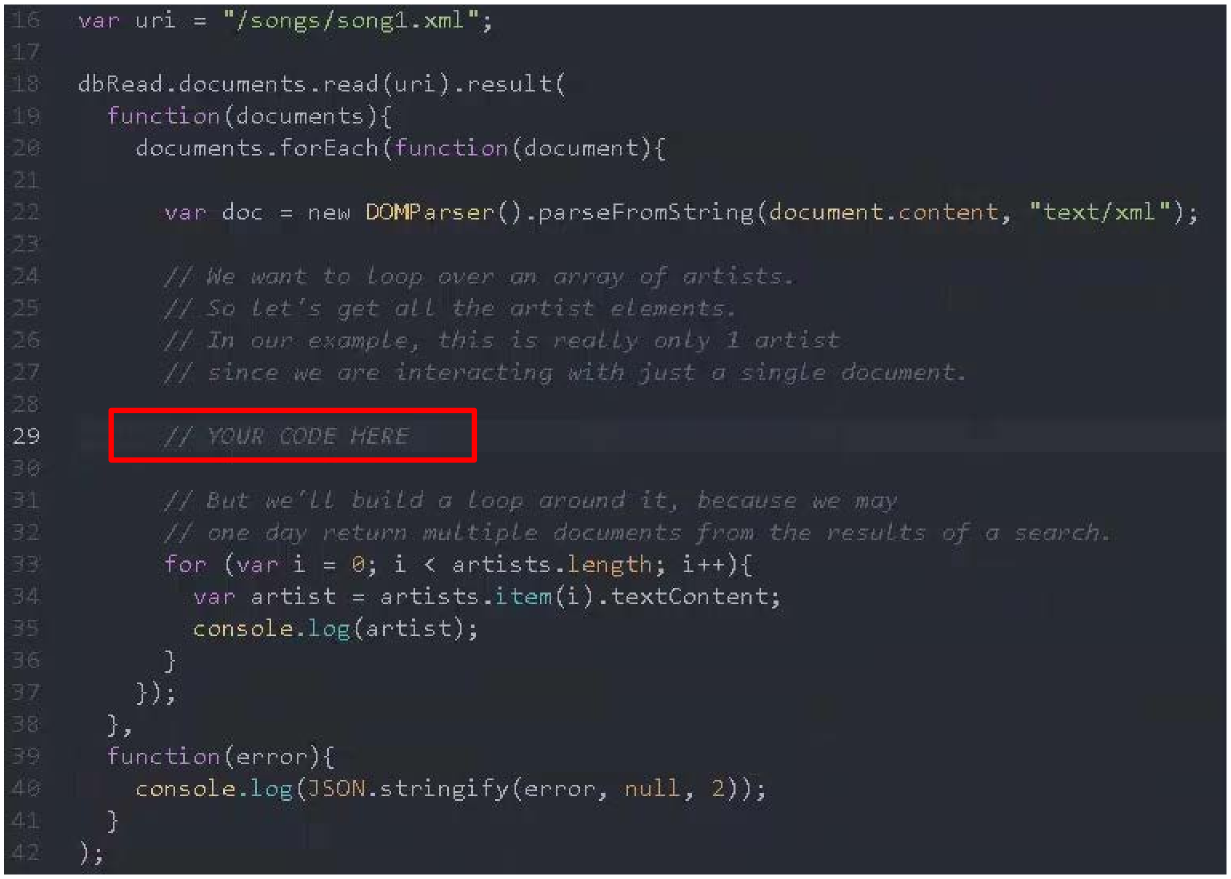
```
c:\mls-developer-node\code\Unit05-Development>npm install xmldom
xmldom@0.1.19 node_modules\xmldom

c:\mls-developer-node\code\Unit05-Development>_
```

3. From your editor in the **Unit05-Development** project open **exercise4.js**
4. Study the first section of the code, noting the database connection information.
5. Also note that a variable called **DOMParser** is created using the **xmldom** package that we just installed:



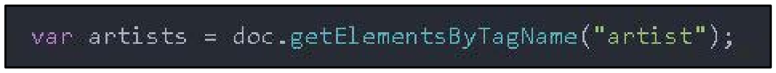
```
// Use the MarkLogic Node.js API to connect to a database and read an XML document.
// This is an example of using the callback result handling technique.
// This example uses xmldom to work with the XML.
    // https://www.npmjs.com/package/xmldom
    // npm install xmldom
'use strict';

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

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

6. Take a minute to study the remaining code and comments.
7. Take note that we are parsing the response data as XML so that we can then go out and get various elements by name.
8. Find the comment that says // **YOUR CODE HERE**



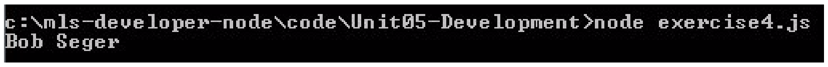
```
var uri = "/songs/song1.xml";

dbRead.documents.read(uri).result(
  function(documents){
    documents.forEach(function(document){

      var doc = new DOMParser().parseFromString(document.content, "text/xml");

      // We want to loop over an array of artists.
      // So let's get all the artist elements.
      // In our example, this is really only 1 artist
      // since we are interacting with just a single document.

      // YOUR CODE HERE

      // But we'll build a loop around it, because we may
      // one day return multiple documents from the results of a search.
      for (var i = 0; i < artists.length; i++){
        var artist = artists.item(i).textContent;
        console.log(artist);
      }
    });
  },
  function(error){
    console.log(JSON.stringify(error, null, 2));
  }
);
```

9. Replace the commented line with the following code. This code will get the <artist> element data from the XML data we have created (var doc) out of the response from the database (document.content):



```
var artists = doc.getElementsByTagName("artist");
```

10. Save your code and test.
11. From your project directory at the command line enter **node exercise4.js**.
12. You should receive the following response:



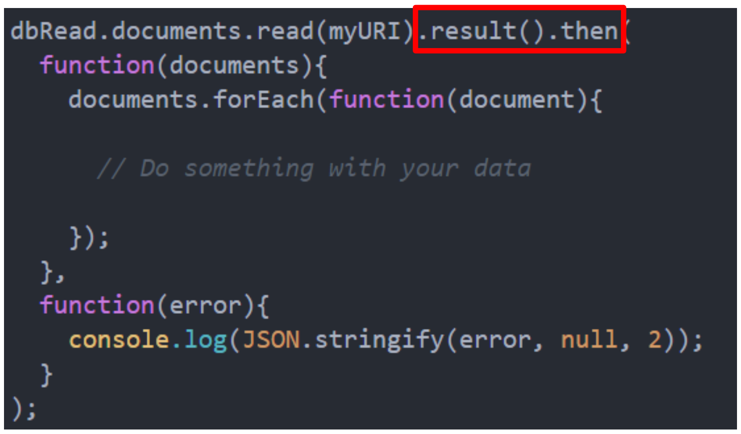
```
c:\mls-developer-node\code\Unit05-Development>node exercise4.js
Bob Seger
```

Exercise 5: Read Documents Using Promises

In this exercise you will adapt the code example that reads a JSON document using a callback. You will implement the same functionality using a promise.

Promises are important when you want to synchronize events. The promise enables you to dictate that one event must finish before the event occurs.

1. In your **Unit05-Development** project open **exercise5.js**
2. Study the comments and code. Note that the code is currently written using a callback result handling technique.
3. The objective is to modify this code to use a promise.
4. Think back to the promise result handling pattern that we discussed during the presentation materials for this unit. The example we discussed showed the use of a promises "**then**" method, which means **after** the result is returned, **then** do the callback function:



```
dbRead.documents.read(myURI).result().then(
  function(documents){
    documents.forEach(function(document){

      // Do something with your data

    });
  },
  function(error){
    console.log(JSON.stringify(error, null, 2));
  }
);
```

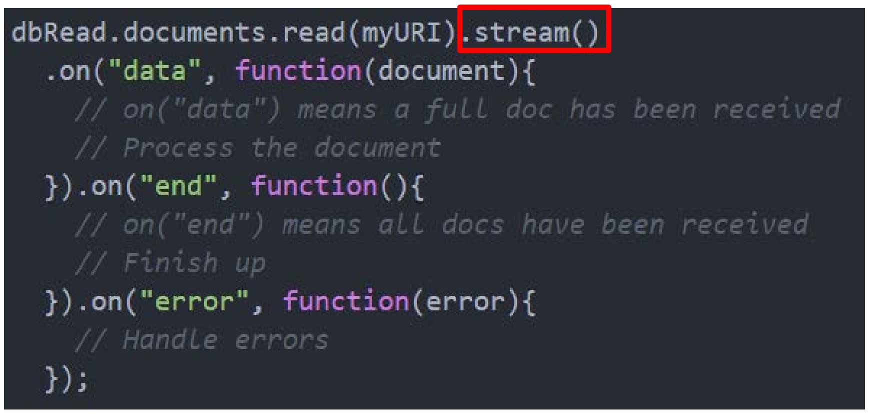
5. Take a few minutes to read more about the promise result handing pattern in the documentation: **http://docs.marklogic.com/guide/node-dev/intro#id_94923**
6. Modify the code in **exercise5.js** to use a promise.
7. Test your code changes (**node exercise5.js** from your project directory at the command line).
8. If you get stuck, please consult your instructor or use the solution project as a reference.

Exercise 6: Read Documents Using Streams

In this exercise you will adapt the code example that reads a JSON document using a callback. You will implement the same functionality using a stream.

Streams are important when working with large documents or large sets of results. The stream enables you to incrementally process the results.

1. In your **Unit05-Development** project open **exercise6.js**
2. Study the comments and code. Note that the code is currently written using a callback result handling technique.
3. The objective is to modify this code to use a stream.
4. Think back to the stream result handling pattern that we discussed during the presentation materials for this unit. The example we discussed showed the use of streams and their **.on** "**data**", "**end**" and "**error**" callback functions:



```
dbRead.documents.read(myURI).stream()
  .on("data", function(document){
    // on("data") means a full doc has been received
    // Process the document
  }).on("end", function(){
    // on("end") means all docs have been received
    // Finish up
  }).on("error", function(error){
    // Handle errors
  });
```

5. Take a few minutes to read more about the stream result handing pattern in the documentation: **http://docs.marklogic.com/guide/node-dev/intro#id_80029**
6. Modify the code in **exercise6.js** to use a stream.
7. Test your code changes (**node exercise6.js** from your project directory at the command line).
8. If you get stuck, please consult your instructor or use the solution project as a reference.

DIY: Create a Database Client

In this exercise you will make some changes to the "**hello**" project that you created back in Unit 1. Going forward this project will be used as a way to do some development against the Star Wars data.

1. In your editor, open the "**hello**" project that you created back in Unit 1 (**c:\hello**).
2. Update the project so that the database client connects to the **star-wars-content** database (using the REST instance you created called **star-wars** on **port 5002**). We no longer want this project to connect to the Samplestack database.
3. In the next Unit you will use this connection to load data into the Star Wars database.