MarkLogic Server

Getting Started With MarkLogic Server

MarkLogic 10
May, 2019

Last Revised: 10.0-5, September, 2020

Table of Contents

Getting Started With MarkLogic Server

1.0 Introduction to MarkLogic Server

MarkLogic Server is a powerful software solution for harnessing your digital content all in a single database. MarkLogic enables you to build complex applications that interact with large volumes of JSON, XML, SGML, HTML, RDF triples, binary files, and other popular content formats. The unique architecture of MarkLogic ensures that your applications are both scalable and high-performance, delivering query results at search-engine speeds while providing transactional integrity over the underlying database.

1.1 Requirements

Before using the procedures in this guide, you must first install MarkLogic Server, including accepting the MarkLogic License Agreement and creating the initial MarkLogic user. Both a successful installation and acceptance of the license agreement are required before using the software. For instructions on installing the software, including a list of supported platforms, see Supported Platforms in the *Installation Guide*.

1.2 About this Document

This document describes how to get started using MarkLogic Server on your system. It introduces some concepts of MarkLogic Server, JavaScript, and XQuery, and is intended for first-time users of MarkLogic Server.

The list of guides available for MarkLogic can be found in the Documentation section of the *Release Notes*. For detail on other aspects of MarkLogic, see the complete documentation available at http://docs.marklogic.com/.

2.0 Getting Started with MarkLogic Cloud Services

You can leverage the power of MarkLogic in the cloud via the following service on Amazon Web Services (AWS) and Microsoft Azure. To learn more about MarkLogic cloud services, see https://a.marklogicsvc.com/ for AWS and https://z.marklogicsvc.com/ for Azure.

Service	Description
Data Hub Service	MarkLogic Data Hub Service provides a managed AWS or Azure instance in which to deploy an operational data hub created using Data Hub Framework (DHF). The service is designed to be transparent, scaleable, and easy to deploy. For more details, see https://www.marklogic.com/product/marklogic-database-overview/data-hub-service/ and https://docs.marklogic.com/cloudservices/.

3.0 Getting Started with MarkLogic Server-Side JavaScript

This section describes the following procedures to get started creating and querying documents using JavaScript in MarkLogic:

- About Query Console
- JavaScript Examples
- Module Programs and Other Considerations

3.1 About Query Console

Query Console is an interactive web-based query tool that is bundled with MarkLogic. Query Console enables you to write ad-hoc queries and view the results without using `.sjs` or `.xqy` files. You can view the results as formatted (Auto) or in plain-text (Raw) output. Query Console is designed for a modern web browser with JavaScript support.

Many of the examples in this document assume you are using Query Console. To learn more about Query Console, see the Query Console Walkthrough in the *Query Console User Guide*

3.2 JavaScript Examples

This section walks you through creating some simple documents in JavaScript and contains the following parts:

- Create a Simple JSON Document
- Creating and Searching Documents

3.2.1 Create a Simple JSON Document

To create a simple document and query it, perform the following steps:

1. Go to the following URL (substitute your hostname if MarkLogic is not running on your local machine):

   http://localhost:8000/

2. When prompted, enter a username and password. The user should have the `admin` role (you can use the same user created when you installed MarkLogic).

3. Optionally, in Query Console, create a new workspace. This will make it easier to come back to later.

4. To create a JavaScript object and return the results, enter the following in the query text area of Query Console:

```
const obj = {key1:"value1", key2:"value2"};
obj;
```

Click the Run button. You will see the object serialized in the results pane.

5. To create a document, enter the following in the query text area of Query Console:

```
declareUpdate();
const obj = {key1:"value1", key2:"value2"};
xdmp.documentInsert("/simple.json", obj);
```

Click the Run button. You will see “your query response was empty” in the results pane. But what actually happened is you created a document in the database. The `declareUpdate()` function is required everytime your program includes an update to the database (otherwise an exception is thrown—try it).

6. Look at the document you just created by entering the following in the query text area of Query Console:

```
cts.doc("/simple.json");
```

Click the Run button. You will see the JSON document you created in the results pane.

7. To see the `key1` property, enter the following and then click the Run button:

```
cts.doc("/simple.json").root.key1;
```

You will see the value for the `key1` property (`value1`) in the results pane. Note that cts.doc returns a document node, so you first have to navigate to the `root` to get to the JSON node, and then you can navigate to the `key1` property.

3.2.2 Creating and Searching Documents

To create some documents and then perform some searches against them, perform the following steps:

1. In Query Console, run the following:

```
// create some documents to search over
declareUpdate();
const phrase1 = "The quick brown fox jumps over the lazy dog.";
const fox = {fox: phrase1};
const phrase2 = "The huge white elephant walked in the mud.";
const elephant = {elephant: phrase2};
const phrase3 = "The fast electric car drove down the highway.";
const car = { car: phrase3};
const obj1 = { fox: {colors:["brown", "red", "yellow"],
                     actions:["jump", "sleep", "eat"]} ,
               elephant: {colors:["brown", "white", "green"],
                     actions:["blow horn", "sleep", "eat", "walk"]},
               car: {colors:["brown", "red", "yellow", "green", "grey"],
                     actions:["speed", "drive", "stop", "accelerate"]}
             };
const col = "my-phrases";
const perms = [xdmp.permission("qconsole-user", "read"),
               xdmp.permission("qconsole-user", "update")];
xdmp.documentInsert("/fox.json", fox, perms, col);
xdmp.documentInsert("/elephant.json", elephant, perms, col);
xdmp.documentInsert("/car.json", car, perms, col);
xdmp.documentInsert("/stuff.json", obj1, perms, col);
```

2. Click the Explore button in query console. You should see the four new documents you just created. You can click on them to see their contents.

3. Now in another Query Console buffer, run the following:

```
// find all documents with the word "the" in the "my-phrases"
// collection and count them
let count = 0;
const results = new Array();
for (const result of cts.search(
       cts.andQuery(["the", cts.collectionQuery("my-phrases")]))) {
  count++;
  results.push(result); };
results.push(fn.concat("Count = ", count));
results;
```

This returns the following array:

```
[
 {"car":"The fast electric car drove down the highway."},
 {"fox":"The quick brown fox jumps over the lazy dog."},
 {"elephant":"The huge white elephant walked in the mud."},
 "Count = 3"
]
```

4. Now in another Query Console buffer, run the following:

```
// find all documents in the "my-phrases" collection
// with the word "car" and count them
let count = 0;
const results = new Array();
for (const result of cts.search(cts.andQuery([
        cts.collectionQuery("my-phrases"),
        cts.wordQuery("car")])) ) {
  count++;
  results.push(result); };
results.push(fn.concat("Count = ", count));
results;
```

This returns the following array:

```
[
 {"car":"The fast electric car drove down the highway."},
 "Count = 1"
]
```

5. Try changing some of the values in the `cts.collectionQuery` or the `cts.wordQuery` to see what different results you get.

6. Now in another Query Console buffer, run the following:

```
// find all documents with the word "car" and count them
let count = 0;
const results = new Array();
for (const result of fn.collection() ) {
  let res = result.root.car;
  let x = res;
  count++;
  results.push(x); };
results.push(fn.concat("Count = ", count));
results;
```

This returns the following array:

```
[
 "The fast electric car drove down the highway.",
 null, null,
 {"colors":["brown", "red", "yellow", "green", "grey"],
  "actions":["speed", "drive", "stop", "accelerate"]},
 "Count = 4"
]
```

7. A simpler way to find the number of documents with the word “car” in it is to perform the following:

```
cts.estimate("car");
```

This is an “estimate” of the number of documents that match the query, which means it is the number of documents that the indexes indicate are a match. For many searches, this will be accurate. For details about search, see the *Search Developer’s Guide*.

8. You can also use the `jsearch` API to perform the search as follows:

```
import * as jsearch from '/MarkLogic/jsearch.mjs';
jsearch.documents()
  .where([cts.collectionQuery('my-phrases'), cts.wordQuery('car')])
  .map({snippet: true})
  .result();
```

This returns two objects, the first is the HTTP header and the second contains the response from the endpoint, which contains a report detailing what matches the search.

```
{"results":
  [{"score":108544,
    "fitness":0.670031011104584,
    "uri":"/car.json",
    "path":"fn:doc(\"/car.json\")",
    "confidence":0.497606456279755,
    "index":0,
    "matches":
      [{"path":"fn:doc(\"/car.json\")/text(\"car\")",
        "matchText":
          ["The fast electric ", {"highlight":"car"},
           " drove down the highway."]
        }]
  }],
 "estimate":1
}
```

For details about `jsearch`, see *jsearch API Documentation* and Creating JavaScript Search Applications in the *Search Developer’s Guide*.

This is just a very brief introduction to how you can search in documents. For information about the search functionality possible in MarkLogic, see the *Search Developer’s Guide*.

3.3Module Programs and Other Considerations

This section walks you through some Javascript changes that are implemented with a view towards improving Javascript performance:

- Converting Scripts to Modules
- New Mimetype for Javascript Modules

3.3.1 Converting Scripts to Modules

Evaluating a JavaScript program may generate side-effects on the JavaScript global environment; therefore, each JavaScript program is evaluated in a separate v8 context. The overhead of creating such a context is significant, and in the recent v8 version that overhead has increased by roughly 40%.

To compensate for this overhead, it is suggested that you convert your Javascript scripts to Javascript modules. A Javascript module program is one that adheres to the following:

1. A program that uses strict JavaScript syntax and can be compiled as a JavaScript module.
2. A program that contains a main module with ".mjs" extension.
3. Is any adhoc program that uses import or export syntax.

For further reading on the details of converting script programs into module programs, please see our Javascript Reference Guide.

3.3.2 New Mimetype for Javascript Modules

In order to support Javascript module programs, a new server mimetype has been created. All module URIs must conform to the new mimetype:

- name: "application/vnd.marklogic-js-module"
- Extension: "mjs"

You may view this new mimetype by navigating to the Admin UI and selecting the Mimetypes from the explorer pane.

The extension for a module URI in the import statement may be omitted. When the module URI in an import statement doesn't contain an extension, an extension mapping to any of the above MIME types must be added to resolve the specified module. For example:

```
import { square, diag } from 'lib/top'; // map to lib/top.js or lib/top.mjs
```

4.0 Getting Started with MarkLogic Using REST, Java, or Node.js

MarkLogic includes a REST Client API, a Java Client API, and a Node.js Client API. These APIs allow you to create full-featured, complex applications that create and update content in a MarkLogic database, search your content and return search results as well as facets, store JSON documents in MarkLogic, and much more. These APIs allow you to create these applications without needing to write any XQuery code.

To quickly get started, first run through Getting Started with the MarkLogic REST API in the *REST Application Developer's Guide*. You can use the REST API directly using an HTTP tool like cURL, or you can write a thin layer on top of it in your favorite scripting or programming language and then quickly use the language you are familiar with.

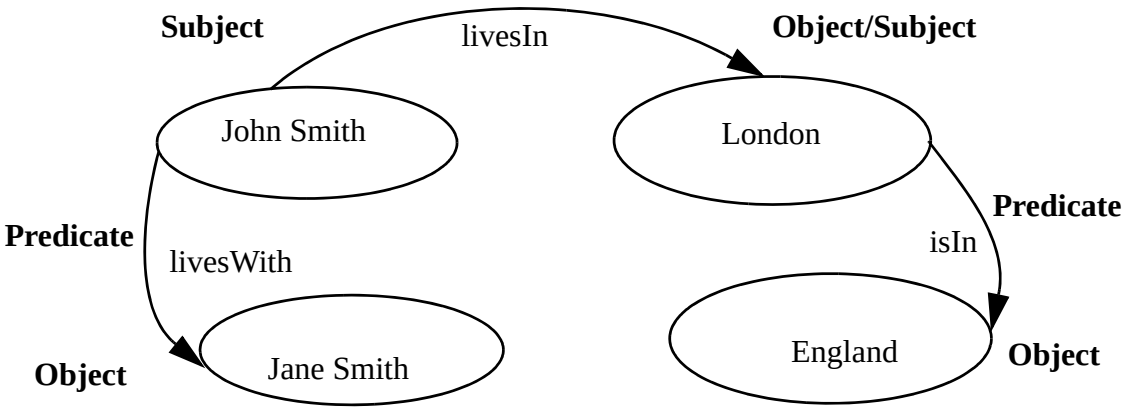
The Java Client API is built on top of the REST API. It includes the functionality of the REST API and is designed to be a familiar interface for Java developers. To get started with the Java Client API, see Getting Started in the *Java Application Developer's Guide*.

The Node.js Client API is also built on top of the REST API. It includes the functionality of the REST API and is designed to be a familiar interface for JavaScript developers using Node.js. To get started with the Node.js Client API, see Getting Started in the *Node.js Application Developer's Guide*.

5.0 Getting Started with MarkLogic Semantics Graph

MarkLogic includes tools to index and store semantic graphs in the form of RDF triples. You can search and manage those triples using native SPARQL query and SPARQL Update, and do inference to discover new facts about your data. MarkLogic can function as a document store for triples embedded in documents and as a triple store for triples in graphs. These semantic capabilities can be accessed natively or through JavaScript, XQuery, and REST.

Each RDF triple represents a fact (John Smith lives in London) in the form of a statement made up of a subject, predicate, and object. Groups of triples are stored as RDF graphs. Here is a model of a simple RDF graph that contains three triples:



Using this graph, we could ask “who lives in England?” and learn that both John Smith and Jane Smith live in England.

To get started with Semantics, see Introduction to Semantic Graphs in MarkLogic and Getting Started with Semantic Graphs in MarkLogic in the *Semantic Graph Developer’s Guide*.

6.0 Getting Started with MarkLogic XQuery Applications

This section describes the following procedures to get started using XQuery in MarkLogic Server:

- Exploring the Use Cases
- The App-Services App Server
- Sample XQuery Application that Runs Directly Against an App Server

Be sure to complete each procedure in the order presented.

6.1 Exploring the Use Cases

As part of the XQuery standard, the W3C working group has assembled a set of use cases that demonstrate how XQuery can be used to accomplish a variety of sample tasks. As part of its installation process, MarkLogic Server provides a workspace for Query Console containing working demonstrations of these use cases. The use cases provide an excellent starting point for familiarizing yourself with the XQuery language and with some important aspects of MarkLogic Server.

To explore the use cases, complete the following steps:

1. Go to the following URL (substitute your hostname if MarkLogic is not running on your local machine):

   http://localhost:8000/

2. Click the Query Console button in the top navigation bar.

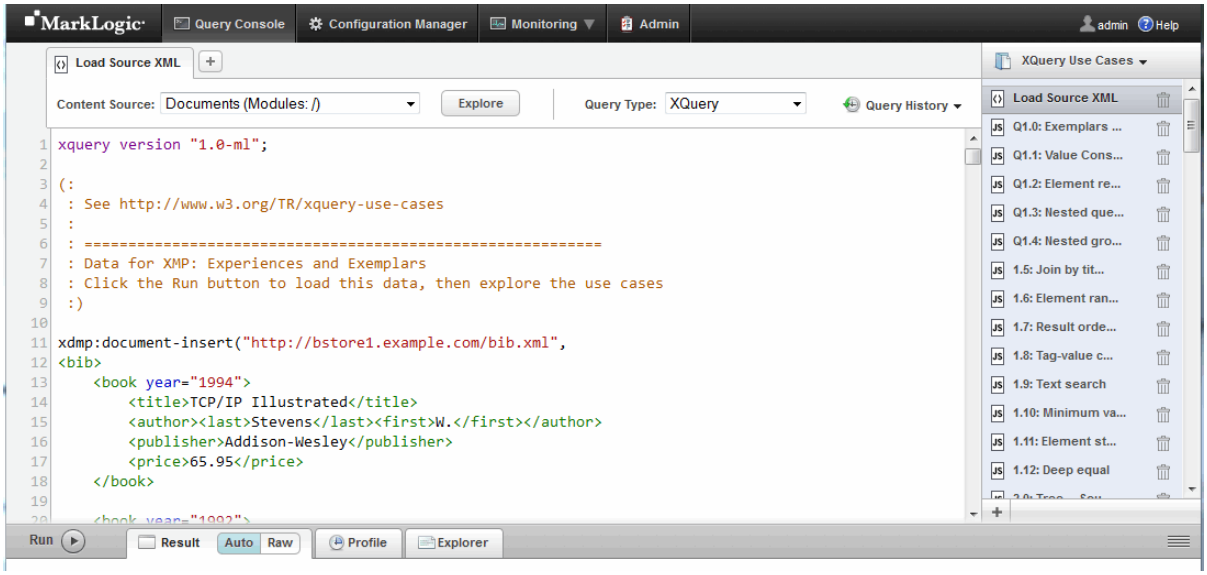
3. In Query Console, click the Workspace link (towards the upper right corner of Query Console) and select Import...

4. Navigate to `<marklogic-dir>/Samples/w3c-use-cases.xml`.

   Note: If MarkLogic is running on the same machine as your browser, you can navigate to this directory (for example, `c:/Program Files/MarkLogic/Samples` on Windows or `/opt/MarkLogic/Samples` on Linux), but if MarkLogic is installed on a remote server, then you must copy this file from the server to the machine in which your browser is running, and then navigate to the directory where you copied the file.

5. Click Import. The W3C XQuery use cases are imported into Query Console.

6. Click Load Source XML query buffer in the workspace pane to display the query into Query Console.



7. Click the Run button.

   A confirmation message displays indicating the documents have been loaded.

8. The use cases are divided into the following categories, with each category having several queries:

   - 1.0: Exemplars
   - 2.0: Tree
   - 3.0: Sequence
   - 4.0: Relational
   - 5.0: SGML
   - 6.0: String Search
   - 7.0: Namespace
   - 8.0: Recursive
   - 9.0: Strong

At this point, you have different options to explore the use cases. We recommend you use the following procedures as examples to maximize your experience with the use cases:

- Viewing the Source XML
- Viewing a Use Case
- Editing a Use Case

You can complete these procedures in any order.

6.1.1 Viewing the Source XML

Note: This procedure focuses on the first use case topic, “1.0: Exemplars.” You can view the source XML for any of the use cases.

To view the source XML, complete the following steps:

1. Click the 1.0 Source XML query buffer in the workspace pane to display the query into Query Console.
2. View the source code in the query editor tab.
3. Click Run.
4. View the source XML at the bottom in the XQuery Results pane.

6.1.2 Viewing a Use Case

Note: This procedure focuses on the first use case in “Exemplars.” You can view any use case in other topics as well.

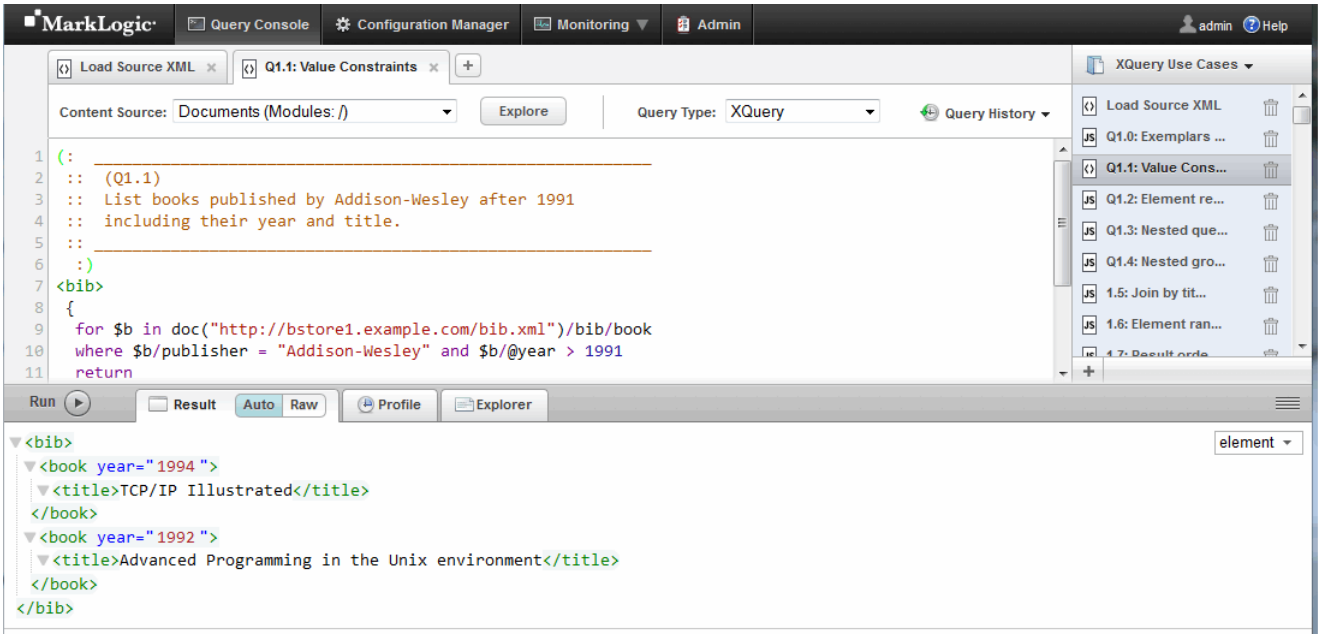
To view the use case, complete the following steps:

1. Click the 1.1 Value constraints query buffer in the workspace pane to display the query into Query Console.
2. View the source code in the query editor tab.

   ```
   Lists books published by Addison-Wesley after 1991 including their year
   and title.
   ```

   Notice that XQuery comments are wrapped in “smiley faces”: `(: comment :)`
3. Click Run.
4. View your results.

The following shows the results:



6.1.3 Editing a Use Case

> **Note:** This procedure focuses on the first use case in "Exemplars." You can edit any use case in the list.

To edit a use case, complete the following steps:

1. Click the 1.1 Value constraints query buffer in the workspace pane to display the query into Query Console.

2. View the source code in the query editor tab.

3. Change the following source from:

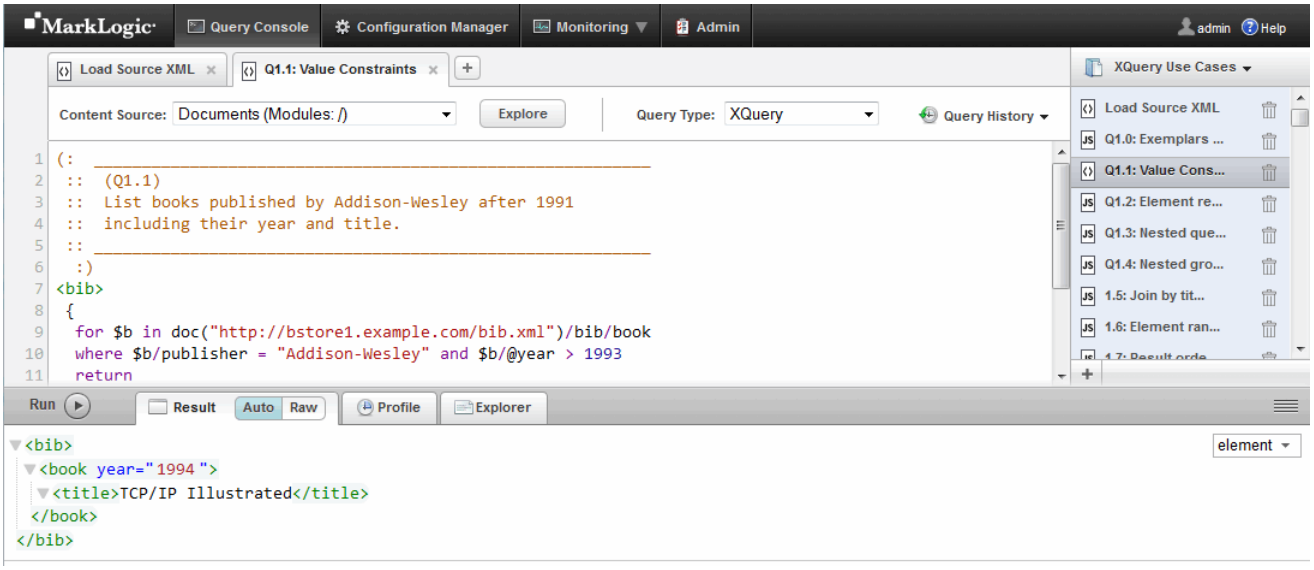
```
where $b/publisher = "Addison-Wesley" and $b/@year > 1991
```

   to:

```
where $b/publisher = "Addison-Wesley" and $b/@year > 1993
```

4. Click Run.

5. View the new query results.



Note: You may change the source as much as you like. Explore and customize each use case as thoroughly as possible. You can reload the workspace from the Sample directory to restore it.

6.2 The App-Services App Server

The `App-Services` App Server at port 8000 hosts Query Console and other MarkLogic applications. The `App-Services` App Server also serves as an HTTP App Server, an XDBC App Server, and as a REST API instance. XQuery, Node.js, XCC, and REST applications can be run directly on the `App-Services` App Server.

By default, the `App-Services` App Server uses the `Documents` database. The MarkLogic APIs provide mechanisms for changing the database when necessary.

The Sample XQuery Application described below is run on the `App-Services` App Server.

6.3 Sample XQuery Application that Runs Directly Against an App Server

In this section, you create a sample XQuery application. This is a simple browser-based application that runs against an HTTP App Server, and that allows you to load and modify some data. First you will create the App Server, and then create the application. This procedure includes the following parts:

- Creating a New App Server
- Creating the Sample XQuery Application

6.3.1 Creating a New App Server

In this section, you create a new HTTP App Server. An App Server is used to evaluate XQuery code against a MarkLogic database and return the results to a browser. This App Server uses the Documents database, which is installed as part of the MarkLogic Serverinstallation process. In “Sample XQuery Application that Runs Directly Against an App Server” on page 18, you use this App Server to run a sample XQuery application.

To create a new App Server, complete the following steps:

1. Open a new browser window or tab.

2. Open the Admin Interface by navigating to the following URL (substitute your hostname if MarkLogic is not running on your local machine):

   http://localhost:8001/

3. Log in with your admin username and password.

4. Click the Groups icon on the left.

5. Click on the Default icon within the Groups branch.

6. Click on the App Servers icon within the Default group.

7. Click the Create HTTP tab.

8. Go to the HTTP Server Name field and enter `TestServer`.

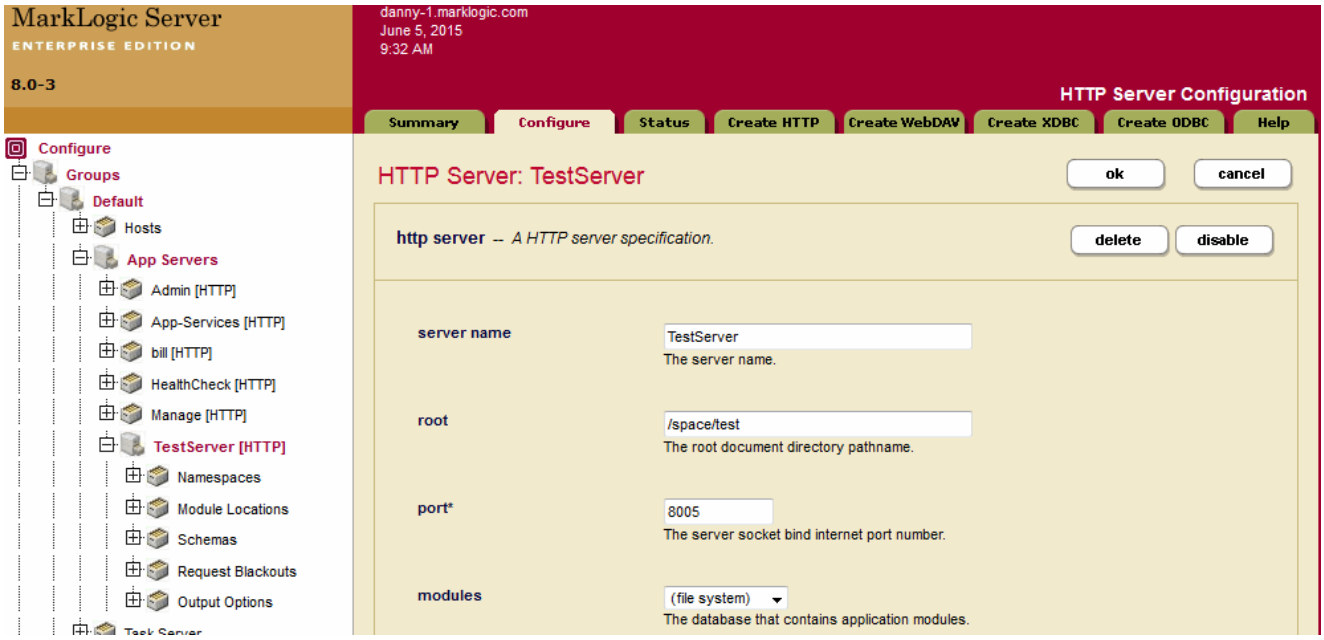
   This is the name that the Admin Interface uses to reference your server on display screens and in user interface controls.

9. Go to the Root directory field and enter `/space/test` (or whatever directory you want for your App Server root, for example `c:/space/test` on a Windows system).

   By default, the software looks for this directory in your MarkLogic Server program directory, as specified in the *Installation Guide*. But it is much better practice to specify an absolute path (such as `C:\space\test` on a Windows platform or `/space/test` on a Linux platform).

10. Go to the Port field and enter `8005` (or whatever port you want to use for this App Server).

    The following screen shows an HTTP server with these values:



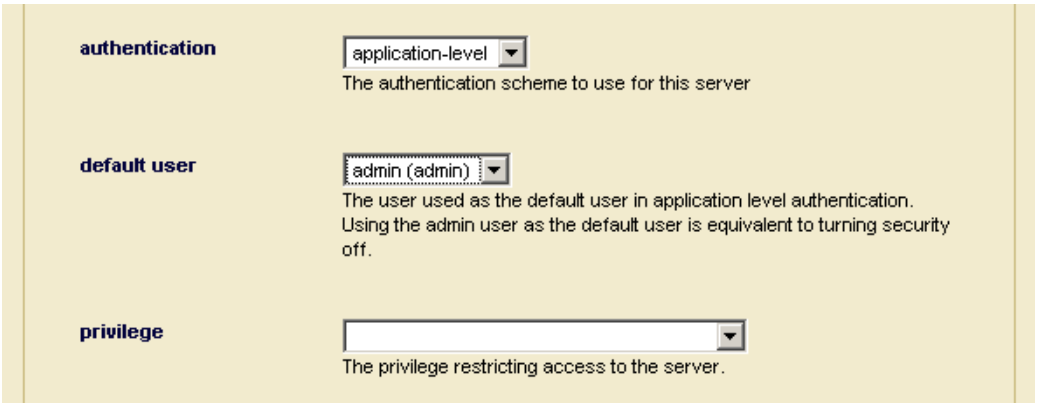
11. Scroll down to Authentication and select `application-level`.

    Note: This makes it so you do not need to enter a username or password against this App Server. If you want to enter a username and password, leave this setting as `digest`.

12. Choose an admin user (it has the word `admin` in parenthesis) as the Default User.

13. Leave the privilege field blank.

    The following screen shows an HTTP server with these values



14. Scroll to the top or bottom and click OK.

15. See that TestServer is added to the HTTP Server branch.

6.3.2 Creating the Sample XQuery Application

To create and run the sample XQuery application, complete the following steps:

1. This procedure assumes you are using the application directory `/space/test`. If you are using a different directory (for example, `c:/space/test`), just make everything relative to that directory.

2. Create a directory on the host in which MarkLogic is running named `/space/test` (or whatever directory you want).

3. Open a text editor and create a new file called `load.xqy` in the `/space/test` directory.

4. Copy and save the following code into this `.xqy` file:

```
xquery version "1.0-ml";
(: load.xqy :)
xdmp:document-insert("books.xml",
  <books xmlns="http://www.marklogic.com/ns/gs-books">
    <book bookid="1">
      <title>A Quick Path to an Application</title>
      <author>
        <last>Smith</last>
        <first>Jim</first>
      </author>
      <publisher>Scribblers Press</publisher>
      <isbn>1494-3930392-3</isbn>
      <abstract>This book describes in detail the power of how to use
XQuery to build powerful web applications that are built on the
MarkLogic Server platform.</abstract>
    </book>
  </books>
),

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Database loaded</title>
  </head>

  <body>
<b>Source XML Loaded</b>
<p>The source XML has been successfully loaded into the database</p>
  </body>
</html>
```

5. Create another file called `dump.xqy` in the `/space/test` directory.

6. Copy and save the following code into this `.xqy` file:

```
xquery version "1.0-ml";
(: dump.xqy :)
declare namespace bk = "http://www.marklogic.com/ns/gs-books";

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Database dump</title>
  </head>
  <body>
  <b>XML Content</b>
{
    for $book in doc("books.xml")/bk:books/bk:book
    return
    <pre>
        Title: { $book/bk:title/text() }
        Author: { ($book/bk:author/bk:first/text(), " ",
        $book/bk:author/bk:last/text()) }
        Publisher: { $book/bk:publisher/text() }
    </pre>
}
    <a href="update-form.xqy">Update Publisher</a>
  </body>
</html>
```

7. Create another file called `update-form.xqy` in the `/space/test` directory.

8. Copy and save the following code into this `.xqy` file:

```
xquery version "1.0-ml";
(: update-form.xqy :)
declare namespace bk="http://www.marklogic.com/ns/gs-books";

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Change Publisher</title>
  </head>
  <body>
    {
      let $book := doc("books.xml")/bk:books/bk:book[1]
      return
      <form action="update-write.xqy">
        <input type="hidden" name="bookid" value="{ $book/@bookid }"/>
        <p><b>
Change publisher for book <i>{ $book/bk:title/text() }</i>:
        </b></p>
        <input type="text" name="publisher"
            value="{ $book/bk:publisher/text() }"/>
        <input type="submit" value="Update publisher"/>
      </form>
  }
  </body>
</html>
```

9. Create another file called `update-write.xqy` in the `/space/test` directory.

10. Copy and save the following code into this `.xqy` file:

```
xquery version "1.0-ml";
(: update-write.xqy :)
declare namespace bk="http://www.marklogic.com/ns/gs-books";

declare function local:updatePublisher()
{
  if (doc("books.xml")) then
    let $bookid := xdmp:get-request-field("bookid")
    let $publisher := xdmp:get-request-field("publisher")
    let $b := doc("books.xml")/bk:books/bk:book[@bookid = $bookid]
  return
      if ($b) then
    (
      xdmp:node-replace($b/bk:publisher,
            <bk:publisher>{ $publisher }</bk:publisher>)
,
       xdmp:redirect-response("dump.xqy")
    )
      else
    <span>Could not locate book with bookid { $bookid }.</span>
      else
    <span>Unable to access parent XML document.</span>
};

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Update In Process</title>
  </head>

  <body>
Attempting to complete update and redirect browser to detail page.
    <p>
If you are seeing this page, either the redirect has failed
or the update has failed.  The update has failed if there is
a reason provided below:
      <br/>
      { local:updatePublisher() }
    </p>
  </body>
</html>
```

11. Confirm that you have the following new files in your `Test` directory:

- `load.xqy`
- `dump.xqy`
- `update-form.xqy`
- `update-write.xqy`

12. Confirm that all files end with the `.xqy` extension, not the `.txt` extension.

13. Using these files, continue to the following procedures:

    - Loading the Source XML
    - Generating a Simple Report
    - Submitting New Information

    Be sure to complete these procedures in order.

6.3.2.1 Loading the Source XML

To load the source XML, complete the following procedure:

1. Open a new browser window or tab.

2. Go to the following URL (substitute your hostname if MarkLogic is not running on your local machine):

   http://localhost:8005/Test/load.xqy

   MarkLogic Server runs the new `load.xqy` file.

3. View the following confirmation message:

   Source XML Loaded

   The source XML has been successfully loaded into the database

6.3.2.2 Generating a Simple Report

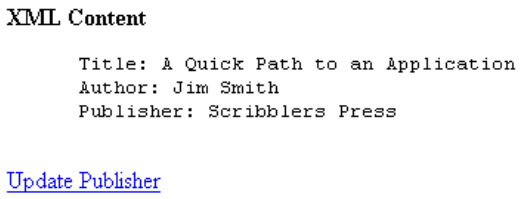
To generate a simple report from the newly loaded XML, complete the following steps:

1. Go to the following URL (substitute your hostname if MarkLogic is not running on your local machine):

   http://localhost:8005/Test/dump.xqy

   MarkLogic Server runs the `dump.xqy` file.

2. View the new HTML-compatible report:



6.3.2.3 Submitting New Information

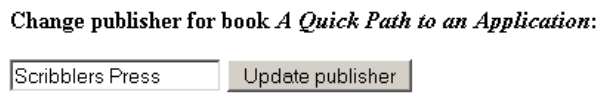
To submit new information to the database, complete the following steps:

1. Go to the following URL (substitute your hostname if MarkLogic is not running on your local machine):

   http://localhost:8005/Test/update-form.xqy

   MarkLogic Server runs the new `update-form.xqy` file.

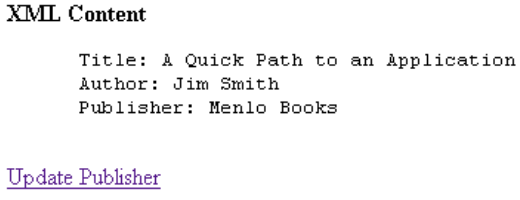
2. View the simple form to update a publisher:



3. Enter “Menlo Books” as the new publisher.

4. Click Update publisher.

   This action automatically calls `update-write.xqy`, which updates the publisher element in the database, and then redirects the browser to `dump.xqy` which displays the updated book information.



5. (Optional) Continue exploring your new files or write your own code to create new XQuery applications.

7.0 Getting Started with XQuery

This chapter describes how to use Query Console to interactively create and execute XQuery code. This chapter includes the following sections::

7.1 XQuery Mini Tutorial

This section includes the following procedures to demonstrate how to use Query Console to take a closer look at the XQuery code in the modules from "Sample XQuery Application that Runs Directly Against an App Server" on page 18:

7.1.1 Creating and Inserting a Document into the Database

This section examines a simple XQuery program that creates a document. First, run the code as follows:

1. Open Query Console in a browser and create a new query.

2. Cut and paste the following code into the XQuery source editor.

```
xquery version "1.0-ml";
xdmp:document-insert("books.xml",
  <books xmlns="http://www.marklogic.com/ns/gs-books">
    <book bookid="1">
      <title>A Quick Path to an Application</title>
      <author>
        <last>Smith</last>
        <first>Jim</first>
      </author>
      <publisher>Scribblers Press</publisher>
      <isbn>1494-3930392-3</isbn>
      <abstract>
          This book describes in detail the power of how
          to use XQuery to build powerful web applications
          that are built on the MarkLogic Server platform.
      </abstract>
    </book>
  </books>
)
```

3. Click Run. Query Console indicated that your query returned the empty sequence.

Taking a closer look at this code, we see it uses the `xdmp:document-insert` function to insert a `books` node into a document with the URI, `books.xml`. The `books` node is in the `http://www.marklogic.com/ns/gs-books` namespace, which is specified with the following namespace declaration:

```
xmlns="http://www.marklogic.com/ns/gs-books"
```

For details on the use of namespaces, see Understanding XML Namespaces in XQuery in the *XQuery and XSLT Reference Guide*.

7.1.2 Querying the Document

This section examines a simple XQuery program that queries the document you previously. First, run the code as follows:

1. Create another Query Console query.

2. Cut and paste the following code into Query Console in the new query editor. This code is from the `dump.xqy` module in “Sample XQuery Application that Runs Directly Against an App Server” on page 18.

```
xquery version "1.0-ml";
(: dump.xqy :)
declare namespace bk = "http://www.marklogic.com/ns/gs-books";

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Database dump</title>
  </head>
  <body>
  <b>XML Content</b>
{
    for $book in doc("books.xml")/bk:books/bk:book
    return
    <pre>
        Title: { $book/bk:title/text() }
        Author: { ($book/bk:author/bk:first/text(), " ",
        $book/bk:author/bk:last/text()) }
        Publisher: { $book/bk:publisher/text() }
    </pre>
}
  </body>
</html>
```

3. Click on the `HTML` button to view the output in HTML.

4. Click Run and examine the results.

Taking a closer look at this query, we see it binds the `bk` prefix to the namespace in which the inserted `books` node resides.

```
declare namespace bk = "http://www.marklogic.com/ns/gs-books";
```

This `bk` prefix can now be used throughout the query to reference the namespace.

The query also defines a `for` clause with an XPath expression that iterates through each child element of the `book` element. Note that the `books` and `book` elements are in the namespace bound by the `bk` prefix:

```
for $book in doc("books.xml")/bk:books/bk:book
```

For details on the `for` clause, see FLWOR Expressions in the *XQuery and XSLT Reference Guide*.

The following lines include simple XPath expressions that return the text node inside each specified element. Note that each element is in the namespace bound by the `bk` prefix:

```
Title: { $book/bk:title/text() }
Author: { ($book/bk:author/bk:first/text(), " ",
           $book/bk:author/bk:last/text()) }
Publisher: { $book/bk:publisher/text() }
```

For details on XPath, see XPath Quick Reference in the *XQuery and XSLT Reference Guide*.

7.1.3 Modifying the Document

Create another Query Console query. Cut and paste the following code into the new query to change the text in the publisher element:

```
xquery version "1.0-ml";

declare namespace bk = "http://www.marklogic.com/ns/gs-books";

xdmp:node-replace(doc("books.xml")//bk:publisher,
  <bk:publisher>Pirate's Press</bk:publisher>)
```

After running this query, rerun the query in “Querying the Document” on page 28 and notice the changed content.

7.1.4 Adding a New Element to the Document

Create another query in Query Console. Cut and paste the following code into the new query to add another `book` to the `books` element described in “Creating and Inserting a Document into the Database” on page 27:

```
xquery version "1.0-ml";

declare namespace bk = "http://www.marklogic.com/ns/gs-books";
```

```
xdmp:node-insert-child(doc("books.xml")/bk:books,
  <book bookid="2" xmlns="http://www.marklogic.com/ns/gs-books">
    <title>An Alternate Path to an Application</title>
    <author>
      <last>Smith</last>
      <first>Jim</first>
    </author>
    <publisher>Scribblers Press</publisher>
    <isbn>3491-3234352-1</isbn>
    <abstract>This book describes another way to use XQuery to build
powerful web applications that are built on the MarkLogic Server
platform.</abstract>
  </book> )
```

Taking a closer look at this query, we see it uses the `xdmp:node-insert-child` function to insert the `book` element as a child of the existing `books` element. To construct the `book` element in the same namespace as the `books` element, we explicitly declare the inserted `book` element to be in the same namespace as the `books` element:

```
<book bookid="2" xmlns="http://www.marklogic.com/ns/gs-books">
```

After running this query, rerun the query in “Querying the Document” on page 28.

7.2 What Next?

To learn more about MarkLogic Server, take a closer look at the MarkLogic documentation on docs.marklogic.com, including the following:

- For more detail on the XQuery language, see the *XQuery and XSLT Reference Guide*.
- For a complete list of standard XQuery functions and MarkLogic Server enhanced functions, see the *MarkLogic XQuery and XSLT Function Reference*.
- For details on how to build XQuery applications, see the *Application Developer’s Guide* and the *Search Developer’s Guide*.
- For details on the features of Query Console, see *Query Console User Guide*.

8.0 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 technical questions, we encourage you to ask your question on Stack Overflow.

9.0 Copyright

MarkLogic Server 10.0 and supporting products.
Last updated: February, 2022

This technology is protected by U.S. Patent No. 7,127,469B2, U.S. Patent No. 7,171,404B2, U.S. Patent No. 7,756,858 B2, and U.S. Patent No 7,962,474 B2, US 8,892,599, and US 8,935,267.

The MarkLogic software is protected by United States and international copyright laws, and incorporates certain third party libraries and components which are subject to the attributions, terms, conditions and disclaimers set forth below.

For all copyright notices, including third-party copyright notices, see the Combined Product Notices for your version of MarkLogic.