

California State University, San Bernardino

CSUSB ScholarWorks

Theses Digitization Project

John M. Pfau Library

2008

Offline searchable database

James Howard Klein

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/etd-project>



Part of the [Databases and Information Systems Commons](#)

Recommended Citation

Klein, James Howard, "Offline searchable database" (2008). *Theses Digitization Project*. 3582.
<https://scholarworks.lib.csusb.edu/etd-project/3582>

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.

OFFLINE SEARCHABLE DATABASE

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Business Administration

by
James Howard Klein, Jr.

June 2008

OFFLINE SEARCHABLE DATABASE

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by

James Howard Klein, Jr.

June 2008

Approved by:


Dr Frank Lin, Committee Chair,
Information and Decision Sciences

6/2/08
Date


Dr Jake Zhu


Dr Walt Stewart, Department Chair,
Information and Decision Sciences

© 2008 James Howard Klein, Jr.

ABSTRACT

Using XML, Javascript, Visual Basic .NET, and HTML, an attempt is made to create a database that can operate regardless of operating system being used. Development of the database itself will be done in several phases. The first phase, and the purpose of this project, is to show a database can work regardless of operating system. The second phase involves developing a SQL query interpreter, and the third phase involves creating a Management capable of running queries and managing multiple databases. To prove the database is possible for the first phase, data must be normalized and stored using basic table components such as data type, primary and foreign keys, auto increment, and null and not null, and default values. A search showing custom search results and a basic management program to manage data will be created to show the database functions.

ACKNOWLEDGMENTS

I would like to thank my loving wife Luisa her complete support and my son Ethan for continually retyping completed material. A very special thank you goes to Mom and Dad for their encouragement and incredible support during hard times. Without Paul and Linda Lucido, this project would have never become a reality. A thank you goes out to Dr Tapie Rohm, the fourth reader. Your assistance with this project is much appreciated. Finally, but not least, a thank you to Dr. Jake Zhu for the initial idea that got this whole project started.

DEDICATION

I would like to dedicate this project to my son Ethan. May your curiosity and interest for computers never die. But, please don't fry another motherboard of mine.

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGMENTS	iv
LIST OF TABLES	ix
LIST OF FIGURES	x
CHAPTER ONE: BACKGROUND	
Introduction	1
Purpose of the Project	1
Context of the Problem	5
Significance of the Project	6
Assumptions	6
Limitations	8
Definition of Terms	9
Organization of the Thesis	9
CHAPTER TWO: OTHER EXTENSIBLE MARKUP LANGUAGE DATABASES	
Introduction	11
Two Types of Extensible Markup Language Databases	11
Extensible Markup Language: Database Initiative	12
Extensible Markup Language Database Application Programming Interface	13
XPath	13
Summary	14

CHAPTER THREE: THE EXTENSIBLE MARKUP LANGUAGE DATABASE

Introduction	15
The Extensible Markup Language Declaration	15
Storing Data Using Extensible Markup Language	16
Data Integrity	18
Primary Keys	19
Foreign Keys	19
Column Definitions	21
An Example Table	22
Testing the Extensible Markup Language Tables	23

CHAPTER FOUR: JAVASCRIPT

Introduction	27
Opening an Extensible Markup Language Table	28
Accessing the Loaded Data	35
Returning Results	40
Why Not Use XPath?	41

CHAPTER FIVE: HYPERTEXT MARKUP LANGUAGE INTERFACE

Introduction	43
Referencing the Javascript	43
Initial Startup	43
Making a Static Page Dynamic	44
Linking to Javascript Functions	45
Automatically Changing the Search Page	46

CHAPTER SIX: VISUAL BASIC .NET ADMINISTRATION
PROGRAM

Introduction	48
Required Imports	49
Opening an Extensible Markup Language Document	50
Displaying Table Data	52
Enforcing Table Structures	53
Auto Increment	54
Varchar Data	54
Integers	55
Primary Keys	56
Foreign Keys	56
Notnull	57
Default	57
Adding and Saving Data	58
Searching Tables	62
CHAPTER SEVEN: CONCLUSIONS AND RECOMMENDATIONS	
Introduction	64
Conclusions	64
Recommendations	65
Lessons Learned	67
Summary	68
APPENDIX A: EXTENSIBLE MARKUP LANGUAGE DATABASE	70
APPENDIX B: JAVASCRIPT CODE	75
APPENDIX C: HYPERTEXT MARKUP LANGUAGE CODE	86

APPENDIX D: VISUAL BASIC .NET CODE	89
REFERENCES	100

LIST OF TABLES

Table 1. Sample Data	3
Table 2. Normalized Articles Table	4
Table 3. Normalized Authors Table	4
Table 4. ArtAuth Link Table	5
Table 5. Journal Table	5

LIST OF FIGURES

Figure 1.	Entity Relationship Diagram	3
Figure 2.	First Row of the Author Table in Extensible Markup Language	18
Figure 3.	Authors Table Definition	22
Figure 4.	The Complete Authors Extensible Markup Language Document	23
Figure 5.	Error in an Extensible Markup Language Document	25
Figure 6.	Error Free Extensible Markup Language Document	26
Figure 7.	Javascript Testing the Browser Type	30
Figure 8.	Preparing to Open an Extensible Markup Language Document with ActiveX	33
Figure 9.	Preparing to Open an Extensible Markup Language Document using Document Implementation Method	34
Figure 10.	Loading the Journal.xml Document	35
Figure 11.	Function JournalInfo	39
Figure 12.	Datagrid Control Example	49
Figure 13.	Extensible Markup Language Loading Function	52
Figure 14.	DataGridView Control Displaying Authors Table Content	53
Figure 15.	Getting the Length of a Varchar	55
Figure 16.	Preventing Duplicate Information in Primary Key Columns	56
Figure 17.	Testing For Null Values	58
Figure 18.	WriteXML Function	60

Figure 19. Saving Changes to the Authors Table	61
Figure 20. Authors Table Search Sub	63

CHAPTER ONE

BACKGROUND

Introduction

The contents of Chapter One presents an overview of the project. The contexts of the problem are discussed followed by the purpose, significance of the project, and assumptions. Next, the limitations and delimitations that apply to the project are reviewed. Finally, definitions of terms are presented.

Purpose of the Project

The purpose of the project was to develop a database that can be accessed on any platform regardless of operating system with or without an internet connection.

In November 2005, Dr Jake Zhu from California State University, San Bernardino approached me with an idea he wanted to implement for the Journal of American Society of Business and Behavioral Sciences' (ASBBS) web site. Dr. Zhu wanted to search through each volume's articles based on article title and author name. Under normal circumstances a server side language such as PHP and data storage with MySQL would be used. However, server capabilities are unknown and future maintenance needs to be considered. Those who are maintaining the ASBBS web

site do not have experience in server side languages. Since server side languages cannot be used, web pages have to be coded solely using HTML.

After looking at the ASBBS web site, some sort of database definitely needs to be used to store article information. Storage has to be compatible and accessible with client side scripting. Some way for the end user to interface with the database also has to be created.

During the initial development of the ASBBS database and interface, a couple of local companies have expressed interest in using the resulting database for marketing and possibly their own web sites. The marketing materials can possibly be mailed out to customers. Given the amount of different operating systems in use, the marketing materials have to be compatible with all operating systems. Also, there is no way to determine if user of the marketing materials has an internet connection. A single web site database turned into a database that can operate on any platform.

For this project, a sample database will be created that is modeled after the real ASBBS database using fictitious data. Table 1 shows the sample data, figure 1 shows the database entity relationship diagram, and tables 2 - 4 show the sample data in third normal form. Tables 1

- 4 and figure 1 will be used to test the different aspects of the database.

Table 1. Sample Data

Article Title	Author Name	Number of Pages	HTML Page	PDF Page
Follow The Leader	James Klein	4	follow.html	follow.pdf
Follow The Leader	Jake Zhu	4	follow.html	follow.pdf
Info-648	Jake Zhu	2	648.html	648.pdf
Computers	Kevin Howard	7	computers.html	computers.pdf
Computers	James Howard	7	computers.html	computers.pdf

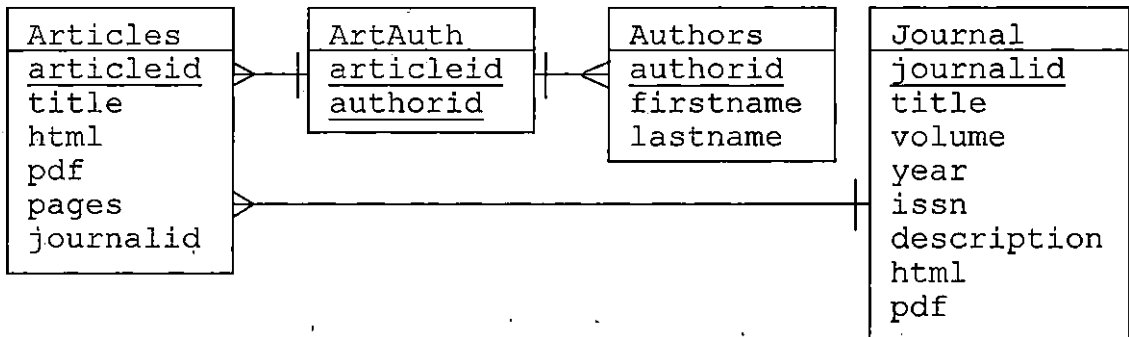


Figure 1. Entity Relationship Diagram

Table 2. Normalized Articles Table

Articles				
articleid	title	pdf	html	pages
0	Follow The Leader	follow.html	follow.pdf	4
1	Info0648	info.html	info.pdf	2
2	Computers	computer.html	computer.pdf	7

journalid
0
0
0

Table 3. Normalized Authors Table

Authors		
authorid	firstname	lastname
0	James	Klein
1	Jake	Zhu
2	Kevin	Howard
3	James	Howard

Table 4. ArtAuth Link Table

artuatuh	
arcileid	authorid
0	0
0	1
1	1
2	2
2	3

Table 5. Journal Table

journal	
journalid	0
title	ASBBS E-Journal
volume	1
number	1
Year	2005
Issn	1557-5004
description	An Official Online Journal of American Society of Business and Behavioral Sciences
Html	html\
Pdf	pdf\

Context of the Problem

The context of the problem was to address the need for searching the articles of the ASBBS journals over the

internet without using server side scripting. However, interest in the database has lead to a need for a database that will work regardless of operating system and availability of the internet. Due to time constraints and the complexity of database software in general, the goal of this project is to show such a database can exist.

Significance of the Project

The significance of the project shows complex software that runs on any platform can be developed. The database and accompanying interface could potentially spawn a way to create software that can run on all platforms. When looking at system requirements for software, there is almost always an operating system requirement. Programs may require a version of Microsoft Windows, a flavor of Linux, or a version of Mac OS. Not being tethered to a specific operating system could potentially vastly increase a software company's earnings.

Assumptions

The following assumptions were made regarding the project:

1. The database itself will be developed through several phases. The current phase, this project, is to develop a working example in order to see

if a database can operate regardless of platform.

2. Given the repetition in commands seen in testing, only the commands will be shown demonstrated in a way that leads to a complete sample function. The complete code for the entire project is in the Appendices.
3. The demonstration database must be in and maintain third normal form to be considered as a proper database.
4. To be successful, the database must work regardless of the relationships that exist between the various tables: many to many, one to one, and one to many.
5. This project does not cover how to program. The reader should know the basics of programming with Javascript, HTML, XML, and Visual Basic .NET.
6. The intentions of this project are to show if a database can function regardless of the platform the end user has. The reader is assumed to know about databases and their administration.

Limitations

During the development of the project, a number of limitations were noted. These limitations are presented in the next section.

The following limitations apply to the project:

1. Every database uses some sort of querying language. MySQL, Microsoft Access, Microsoft SQL Server, for example, all use the SQL query language to interact with a database. Due to time constraints, a query language will not be developed. The point of this project is not to develop the query language or interpreter, but to show a database can work in the requirements discussed earlier.
2. Interaction with the database will be hard coded.
3. A program to administrate any type of database using the methods within this project will not be created. That is considered the next phase of the software itself. Instead, a piece of software will be created to administrate the sample database which will be discussed in chapter 5.

Definition of Terms

The following terms are defined as they apply to the project.

ASBBS: Journal of American Society of Business and Behavioral Sciences

MySQL: Open source database storage and serving software.

PHP: Hypertext Preprocessor. A server side scripting language originally designed for producing dynamic web pages.

XML: Extensible Markup Language. A general purpose markup language that allows programmers to define their own tags.

Organization of the Thesis

The thesis portion of the project was divided into five chapters. Chapter One provides an introduction to the context of the problem, purpose of the project, significance of the project, limitations and delimitations and definitions of terms. Chapter Two consists of a review of relevant literature. Chapter Three show how a sample database is created using XML. Chapter Four details how to interface with the XML database using Javascript. Chapter Five contains the methods used to create HTML based search page. Chapter Six presents the Visual Basic .NET

management program. Chapter Seven presents conclusions and recommendations drawn from the development of the project. Project references follow Chapter Seven. The Appendices for the project consists of: Appendix A XML DATABASE; Appendix B JAVASCRIPT CODE; Appendix C HTML CODE; Appendix D VISUAL BASIC .NET CODE. Finally, the Project references.

CHAPTER TWO

OTHER EXTENSIBLE MARKUP LANGUAGE DATABASES

Introduction

Before attempting to create a database, research can tell if there have been attempts to create a database using XML to store the data, Javascript to programmatically interface with the database, and HTML to accomplish the same goals. Research has turned up interesting facts concerning XML databases.

Two Types of Extensible Markup Language Databases

Two types of XML databases exist: XML-enabled databases and Native XML Databases (NXD). XML-enabled databases maps XML to a traditional database using XML as input and output for data. NXDs use xml as a method to define data storage in a database. NXDs are not actually standalone databases and do not store data in text form.

Several well known database management systems implement native XML. Oracle, Access, MySQL, and DB2 all implement native XML. Examples of typical XML data are similar to the XML tables appearing in the Appendices. If data can be represented and stored in XML format, there is

a possibility of using XML itself to store data.

(Wikipedia, 2008)

Extensible Markup Language: Database Initiative

The XML: DB Initiative was a group dedicated to developing and creating standards for XML Databases. The keyword "was" is used because the latest information on the Initiative's web site is 5 years old. After extensive searching, the Initiative disappeared leaving projects and specifications behind. However, current implementations of XML databases are built on the work of the Initiative. Goals of the initiative included developing technology specifications for managing data in XML databases, community contribution to the specifications under Open Source licensing, formation of a community where vendors and users can ask questions and exchange information regarding XML databases, and promoting the use of XML database products throughout the marketplace. In support of their goals, the Initiative had three projects: XML Database API, XUpdate, and SiXDML. Goal support is programmed in Java and documented using Javadoc. The only project that is widely used is XML Database API. (XML:DB, 2003).

Extensible Markup Language Database Application Programming Interface

The XML Database API is a common set of access mechanisms to XML databases allowing the creation of applications to store, retrieve, modify, and query data stored in a XML database. Core specifications specify how XML databases are to be created in order to be compatible with the API. Two core levels exist: Core Level 0 and Core Level 1.

Core level 0 is the minimum requirements to be considered compliant with the API consisting of the API Base and XMLResource. API Base consists of the core methods used to interface with a XML document. All other API modules are built off the AI Base. XML Resource is a representation of XML data providing access to the data in XML textual, W3C DOM, or SAC ContextHandler forms. Core Level 1 consists of all Core Level 0 requirements as well as XPathQueryService. XPathQueryService is simply querying functions for XML Databases. (XML:DB, 3003).

XPath

XML is implemented in a lot of databases management systems such as Oracle, DB2, and MySQL. The implementations are only Native XML Databases were the XML is stored in a database. Even though XML is stored in a

database, the querying of the XML data is the same that will be required by this project: XPath. XPath is a querying language for XML documents that return arrays of XML data, strings, and integers (Clark).

Summary

The literature important to the project was presented in Chapter Two. Based on the research done no one has created a database whose data is solely stored in XML, or at least gone public with the database. Native XML implementations show such a database can exist.

CHAPTER THREE

THE EXTENSIBLE MARKUP LANGUAGE DATABASE

Introduction

Dynamic web pages are typically programmed using a server side scripting language such as ASP, PHP, or JSP with a database server holding data. Since the database being developed will not reside on a server, using server side scripts is not possible. The chosen method for storing the database is using XML. XML allows programmers to create custom tags. Custom tags allows for an easy way to define tables, rows, and cells. To make sure proper XML is created, the World Wide Web Consortium's (W3C) web site was consulted.

The Extensible Markup Language Declaration

The first line of code for each XML document contains the XML declaration. A declaration line is part of a well formed XML document. The typical declaration line contains up to three elements: the XML version and encoding. The version is used to tell the XML parser the conforming version of the XML document and is stated as version="1.0". Currently there is only one version of XML, version 1.0. The version portion of the declaration is in place in the event there will be future versions of XML.

Encoding, the second portion of the declaration, tells the parser how the XML document is encoded. Typical encoding is UTF-8 and is the default encoding if the encoding portion of the declaration is not specified. Other encoding types can be found at the Internet Assigned number Authority (IANA) at <http://www.iana.org>. The final part of the declaration tells the parser whether the xml document has an internet document type definition (DTD) or an external DTD by using `standalone="yes"` or `"no"`. Using `"yes"` tells the parser that the document has an internal DTD whereas `"no"` tells the parser there is another file that acts as a DTD. For purposes of storing data for a database, each document will have an internal DTD. The default standalone in the declaration is `"yes"` if standalone is not included within the declaration.

The declaration for each document making up the database will include all three portions of the declaration. The declaration for each document is `<?xml version="1.0" encoding="UTF-8"?>`.

Storing Data Using Extensible Markup Language

After the document declaration comes the opening root element. The root element could be simply `"database"` with each child element being the name of each table. In terms

of large amounts of data, the resulting XML document would be very large and memory intensive to load. Keeping the XML documents as simple as possible, each table in the tables should a separate XML document. The file name and root element for the XML document will be the table name. Using the articles table as an example, the root element for the articles table is <articles> and the file name is "articles.xml". The flexibility given by this method will become apparent when creating the articles administration program and in future phases when implementing a SQL interpreter.

Within the root element are the elements defining each row of data. The tag name of the defining elements will be simply "row". Each of the "row" element's children will be named after each of the column names. Using the authors table as an example, the author's first name tag will be <firstname>, last name will be <lastname>, and the author's unique id number will be in the <authorid> tag. Figure 2 shows the first row of data in the author table.

```
<row>
  <authorid>0</authorid>
  <firstname>James</firstname>
  <lastname>Klein</lastname>
</row>
```

Figure 2. First Row of the Author Table in Extensible Markup Language

Data Integrity

Data integrity is a huge concern for database administrators. Tools such as primary and foreign keys to help maintain referential integrity is a must. Imagine a database that has one table, such as Table 1. The database administrator wants to delete the author "Jake Zhu" or the article "Follow the Leader". Each delete would have to be done more than once. The possibilities of deleting a co-author of "Follow the Leader" increase with every delete of the article. Even making sure the correct type of data entered into a cell is important. If a column contains numbers representing 4 digit years, a database administrator wouldn't want some type of text to appear in the column. Defining primary keys and other table attributes becomes a must for administration.

Before any rows of information, the table definitions tag defines all properties for the table including primary and foreign keys, data types for columns, default data,

and whether a column can be null. The next three sections build the table definition portion of a table.

Primary Keys

All properly formed tables in databases such as Access, MySQL, and Oracle use primary keys to determine a unique trait for each row of data. In each of the tables in the sample database the column with "id" in the name signifies the primary key. Database administrators may not always use "id" within the primary key column name. There needs to be a way to determine which field is the primary key and specify the field once within the XML document. When determining how to specify the primary key, multiple occurrences of the "pk" property within the same document can lead to different primary keys being used at different times that can lead to data corruption and program bugs accessing the database. The solution is placing the "pk" tag within the "tabledef" element of each XML document. For the authors table, the "pk" element would look like "<pk>authored</pk>". Tables, such as link tables, can have multiple primary keys. Each key is encased within their own "pk" tag.

Foreign Keys

In many table relationships, one table may have a column that refers to one or more rows in another table.

The columns are foreign keys. Foreign keys typically point to the primary key of another table. Good database administrators use the same column name for both keys in both tables. There are cases when the foreign key is named differently from the primary key. There needs to be a way to tell what column of what table the foreign key is pointing to.

Defining a foreign key occurs within the "tabledef" element by creating a "fk" tag. In between the fk tag is the name of the cell that is to be foreign key. Inside the fk tag declaration are two properties: the column the foreign key points to and the table the column resides in. The foreign key for the artauth's authorid column looks like '`<fk table="authors" column="authorid">authorid</fk>`'. The foreign key column is "authorid" within the artauth table, the table being referenced is "authors" and the column being referenced in authors is "authorid". The reasoning behind having the XML file named after the table is starting to become apparent. When enforcing foreign keys, the administration program can see the table name "authors" telling the program to open "authors.xml" to look up author information based on the foreign key.

Column Definitions

Other column properties besides primary and foreign keys also need to be defined. Typical properties for columns include whether the column can have empty cells through the "notnull" parameter, the default data to put in a cell if notnull is enabled, and the data type for the column. Specifying the different attributes for each column is done within the "tabledef" element by creating a tag named after the column being defined. Contained within the column tag are properties for each of the column properties being used: datatype, notnull, and default. Datatype specifies what kind of data being stored within the column such as integer, varchar, datetime, date, char, text, and autoincrement. Notnull determines whether the column can have empty cells. Setting notnull to "yes" will not allow empty cells when data is being entered into the table, whereas "no" allows empty cells. The "default" property contains the default value for the column if no information is entered when a new row is added to the table.

All properties are required when defining a row in order to prevent errors while running the administration program, Javascript interface, and the HTML search page. Even when using auto increment for primary keys, all

properties are required. The following column definition example defines a column with auto increment as the data type: '<authorid datatype="autoincrement" notnull="yes" default="" />' Figure 3 shows the complete table definition for the authors table.

```
<tabledef>
  <pk>authorid</pk>
  <authorid datatype="autoincrement" notnull="yes" default="" />
  <firstname datatype="varchar(15)" notnull="yes" default="" />
  <lastname datatype="varchar(15)" notnull="yes" default="" />
</tabledef>
```

Figure 3. Authors Table Definition

An Example Table

The complete authors table is shown below. Refer to Appendix A for the rest of the database.

```

<?xml version="1.0" encoding="utf-8"?>
<authors>
  <tabledef>
    <pk>authorid</pk>
    <authorid datatype="autoincrement" notnull="yes" default="" />
    <firstname datatype="varchar(15)" notnull="yes" default="" />
    <lastname datatype="varchar(15)" notnull="yes" default="" />
  </tabledef>
  <row>
    <authorid>0</authorid>
    <firstname>James</firstname>
    <lastname>Klein</lastname>
  </row>
  <row>
    <authorid>1</authorid>
    <firstname>Jake</firstname>
    <lastname>Zhu</lastname>
  </row>
  <row>
    <authorid>2</authorid>
    <firstname>Kevin</firstname>
    <lastname>Howard</lastname>
  </row>
  <row>
    <authorid>3</authorid>
    <firstname>James</firstname>
    <lastname>Howard</lastname>
  </row>
</authors>

```

Figure 4. The Complete Authors Extensible Markup Language Document

Testing the Extensible Markup Language Tables

So far the XML document representing each table of the database have been hand coded. Hand coding can lead to mistakes. The easiest and fastest way to test each document's completeness and error checking is to open the XML document in a web browser. Web browsers can quickly determine errors in the xml and display the potential

cause of the errors. Knowing that a XML document works properly when having troubles coding the interface or user search page makes debugging code much easier, removing the burden of checking the XML documents has been removed. Figure x shows an example of an error and Figure x shows an error free XML document displayed in Microsoft's Internet Explorer.

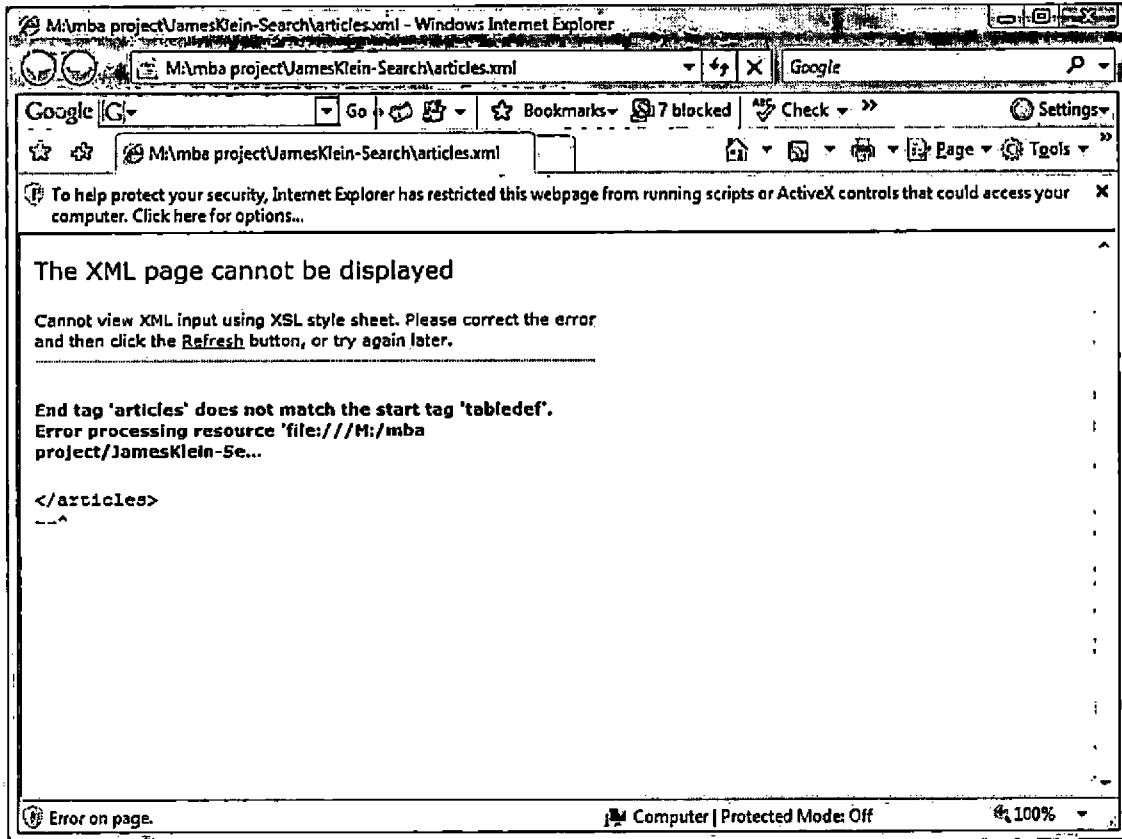


Figure 5. Error in an Extensible Markup Language Document

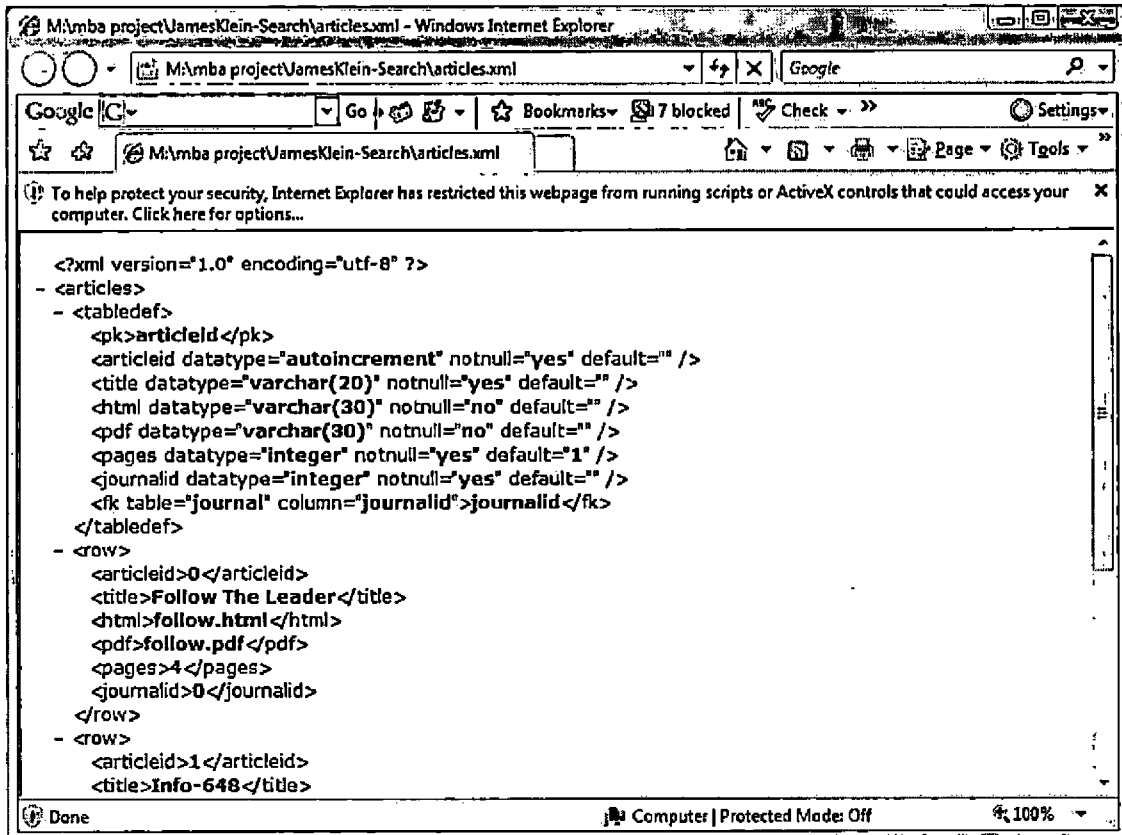


Figure 6. Error Free Extensible Markup Language Document

CHAPTER FOUR

JAVASCRIPT

Introduction

Creating a method to interface with the XML database is not hard. The issue lies in which programming language to use. The code must be able to run on all systems regardless of being compiled or in source code form. Java can interface with the database and run on any platform; however a custom database interface created by a user is not as easily created. Building a custom interface can easily help with acceptance of the database being created. Another problem with Java is the requirement for a Java Virtual Machine. The Java VM is easily obtainable, however not everyone has a Java VM installed. Having to install Java VM deters from ease of use.

The best solution that lends to being able to create custom end user interfaces is Javascript. Being compiled at run time allows Javascript to easily run on all platforms. As long as the user's computer has a web browser installed, Javascript can run. Since the mid 1990s operating systems come standard with a web browser capable of running Javascript scripts.

In this phase of the database development, the Javascript being developed is purpose coded for the sample database. However, the code should be able to operate effectively regardless of whether there are 3 records or 300 records in each table. The Javascript code must also be able to join the author and article tables using the ArtAuth link table.

Covering the entire Javascript code is pointless since most of the Javascript uses the same methods, only the methods will be discussed in a manner that builds up to the database startup portion of the Javascript code. For the complete Javascript refer to Appendix B.

Opening an Extensible Markup Language Table

Considered to be the most important part of the Javascript code, failure to open a XML table will prevent the project from working all together. Javascript does have methods for opening XML documents and feeding the opened documents through a XML parser. For security reasons that simple logic can justify Javascript cannot write XML documents. The security risks involved in letting web sites write to a client's system can dwarf current problems with malware. For that reason alone,

managing the database will be done with Visual Basic .NET, and is discussed in Chapter 5.

Opening an XML document in Javascript requires two different methods depending on the web browser the user has. Microsoft's Internet Explorer uses ActiveX controls to work with XML documents while other browsers such as Mozilla's Firefox, Netscape Communicator, Safari, and Opera use the document implementation method. The first step to open an XML document is to determine the browser being used. The easiest test is determining if ActiveX or document implementation exists. Javascript's typeof command test if something exists. The result "undefined" means the object being tested for doesn't exist. Any other result means the object being tested for does exist (typeof 2008). Testing for the browser being used becomes as simple as two if statements. The document implementation test consists of two typeof tests rather than one. The first test determines if "document.implementation" exists and the second determines if "document.implementation.createDocument" exists. The if statement is then "if ((typeof document.implementation != 'undefined') && (typeof document.implementation.createDocument != 'undefined'))". Testing for Internet Explorer requires one typeof test:

the existence of ActiveX objects. The test is simply "if (typeof window.ActiveXObject != 'undefined')".

Once the browser has been determined, the browser type needs to be stored in a variable to use later on in the script. The variable "browser" is set to "netscape" or "ie" depending on the outcome of the browser tests. Note "browser" is not a global variable. Once the XML document is open, Javascript doesn't need to know which browser is being used. Figure 7 shows the complete browser testing code.

```
var browser = "none";  
  
if ((typeof document.implementation != 'undefined') && (typeof  
document.implementation.createDocument != 'undefined'))  
{ browser = "netscape"; }  
else if (typeof window.ActiveXObject != 'undefined')  
{ browser = "ie"; }
```

Figure 7. Javascript Testing the Browser Type

Now the XML document can be opened. Both opening methods require an object to store the reference to the XML document being opened. For the startup portion of the Javascript code, the XML document object is called "xmlDoc". Tests show there has to be a unique object for each XML document being opened, regardless of whether or not the currently opened XML document being stored in the

"xmlDoc" object is going to be used again. Even if the "xmlDoc" object is cleared and then set to another XML document, the "xmlDoc" object becomes empty. The "xmlDoc" object has to be a global object. The reasoning for being global will be clear later.

After determining the browser type, the "xmlDoc" object must be prepared to open a XML document. For Internet Explorer, the first task is to create a new ActiveX object by using "xmlDoc = new ActiveXObject ('Microsoft.XMLDOM');". Next comes the first problem with accessing a XML document, or any document, is whether to synchronously or asynchronously access the document. Asynchronous access allows retrieved information from a file to be worked with before the entire file has been read. During testing working with the retrieved XML data was faster than retrieving the data creating synchronization errors. Synchronous access waits for the data to finish retrieving from the file before allowing the Javascript to work with the retrieved data. When using synchronous access, no errors occurred and any extra execution time searching through the retrieved data was not noticeable. To achieve synchronous access us "xmlDoc.async = false;".

The next step is telling the "xmlDoc" object what to do once the XML document has been loaded. Using the state change function of the "xmlDoc" object determines the next step after loading the XML document. The statement

```
"xmlDoc.onreadystatechange = function () { if  
(xmlDoc.readyState == 4) JournalInfo() };"
```

 tells the "xmlDoc" object to run a new function when the state of the object changes. The object can have four states. State 4 occurs when the object has loaded the XML document and the data is ready to be worked with. The new function tests for the object state of 4 then runs another function to work with the data. A separate function is used for two reasons. The first reason being each method of opening the XML document needs to access the same code, the second reason is keeping the source code as clean as possible. Earlier the "xmlDoc" is required to be global. The function that works with the retrieved data contained in "xmlDoc" needs to have access to the object. If the function can't access the object, no data can be returned. Figure 8 shows the complete Internet Explorer ActiveX Method.

```
xmlDoc = new ActiveXObject('Microsoft.XMLDOM');
xmlDoc.async = false;
xmlDoc.onreadystatechange = function ()
{ if (xmlDoc.readyState == 4) JournalInfo() };
```

Figure 8. Preparing to Open an Extensible Markup Language Document with ActiveX

Preparing to open an XML document using the document implementation method is slightly different than ActiveX; however the principles are the same. A new object is created using "xmlDoc = document.implementation.createDocument ("", "doc", null);". Telling the document implementation what to do with the retrieved data is done by using "xmlDoc.onload = JournalInfo;". "JournalInfo" is the same function that the ActiveX method was told to load once the XML document has been fully read. Remember, once the XML document has been fully loaded, both methods can use the same code to interact with the loaded data. Figure 9 shows the complete document implementation method for preparing to open a XML document.

```
xmlDoc = document.implementation.createDocument("", "doc", null);
xmlDoc.onload = JournalInfo;
```

Figure 9. Preparing to Open an Extensible Markup Language Document using Document Implementation Method

Once the "xmlDoc" object has been prepared, the XML document can be loaded. The load command is the same regardless of browser. A switch command is used to determine the correct preparation method based on the contents of the "browser" variable. After the switch statement, 'xmlDoc.load ("journal.xml");' is ran to load a XML file. The startup Javascript pulls information from the "Journal" table. In the preceding example, journal.xml is loaded. Any other XML file name can be used. When considering the location of the XML database in relation to the Javascript code, the file location always starts at the same location the search page is opened from. If the search page is opened from "e:\", the starting location of the Javascript code is also "e:\". For example, if the XML database resides in a folder called "data" that is located in "e:\" giving the complete folder path of "e:\data\", loading journal.xml would be 'xmlDoc.load ("data\journal.xml");'. The complete switch statement with loading the XML document is shown in figure 10.


```

switch (browser)
{
  case "ie":
  {
    xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
    xmlDoc.async = false;
    xmlDoc.onreadystatechange = function ()
    { if (xmlDoc.readyState == 4) JournalInfo()    };
    break
  }
  case "netscape":
  {
    xmlDoc = document.implementation.createDocument("", "doc", null);
    xmlDoc.onload = JournalInfo;
    break
  }
}
xmlDoc.load("journal.xml");

```

Figure 10. Loading the Journal.xml Document

Note from the figure above that preparation requires using the name of the object that will be used to store the XML data. Each different XML document being loaded must have its own unique object variable as discussed earlier. Each object must be prepared separately. Generally, all objects are prepared before any data is worked with effectively creating cleaner code and making debugging changes to the Javascript code much easier.

Accessing the Loaded Data

Once a XML document is loaded, the XML data needs to be accessed. Javascript uses the same methods to access XML data regardless of the browser being used. XML tags within a XML document are organized in arrays. Accessing

the tags is as simple as accessing information stored in an array. The first step is to expose the XML data. Creating an array and then setting the array to a collection of XML data is the only way to expose the data. The very first object in the array is the XML document's root element. Using the Journal table that is needed during startup as an example, the initial object in the data array is root element "journal". The command that exposes everything with the "journal" tag is `'var journal = xmlDoc.getElementsByTagName("journal")[0].getElementsByTagName("row");'`. A new variable called "journal" is created storing all of the information within the "journal" tag in a multidimensional array.

Accessing the child tags is just as easy, only the data does not have to be saved to a variable. The "journal" variable already stores the information in a manner that is easily accessible. To access all of the elements within the root "journal" tag the first object in the "journal" variable needs to be accessed by using `"journal[0]"`. The next part of accessing the child elements should look familiar. Using `'getElementsByTagName("title");'` retrieves all of the information stored in every element that had a tag name of "title" in the format of an array. To get the only "title" element for each row

'journal[n]. getElementsByTagName("title")[0]' is used where "n" is the n + 1 row to get the title information from. Since there is only one row in the journal table, n will always be 0. In cases such as the authors table, n can be between 0 and 3 since the table has 4 rows of data.

All that has been done so far is to expose the methods for accessing the information within the first "title" element. "journal" is not just a multidimensional array, but is also a reference to the "xmlDoc" object that can call the methods associated with the object to retrieve data. Looking at the journal.xml in Appendix A, there is only one "title" child element within the XML document. In order to actually see the information between the title element "firstChild.nodeValue" must be used. "nodeValue" returns the information contained with the element being accessed. The full line of code needed to access the title of the journal is 'journal[0]. getElementsByTagName("title")[0]. firstChild.nodeValue', all on one line. Accessing other elements with names other than "title" is as simple as replacing "title" in the line above with the name of the element, such as "year" or "number". In the case of the XML table, "title" would be replaced with the name of the desired row being retrieved. In future phases of the database development, SQL

statement parsing will be implemented. Within SQL statements are column names. Just as "title" is a column name, the column names that appear in the SQL statements can be inserted in place of "title". Figure x shows the complete code to retrieve all of the journal information using the function "JournalInfo". Recall "JournalInfo" is called as soon as journal.xml has loaded.

```

function JournalInfo()
{
    var journal = xmlDoc.getElementsByTagName("journal");

    if (journal.length > 0 )
    {
document.getElementById('jtitle').innerHTML =
journal[0].getElementsByTagName("title")[0].firstChild.nodeValue +
"&nbsp;&nbsp;&nbsp;";
        jjtitle = journal[0].getElementsByTagName("title")[0].firstChild.nodeValue;
        jvolume = journal[0].getElementsByTagName("volume")[0].firstChild.nodeValue;
        jnumber = journal[0].getElementsByTagName("number")[0].firstChild.nodeValue;
        jyear = journal[0].getElementsByTagName("year")[0].firstChild.nodeValue;
        document.getElementById('volume').innerHTML = "Volume " + jvolume + ", No. "
+ jnumber + ", " + jyear + "&nbsp;&nbsp;&nbsp;";
        document.getElementById('issn').innerHTML = "ISSN: " +
journal[0].getElementsByTagName("issn")[0].firstChild.nodeValue +
"&nbsp;&nbsp;&nbsp;";
        document.getElementById('description').innerHTML =
journal[0].getElementsByTagName("description")[0].firstChild.nodeValue +
"&nbsp;&nbsp;&nbsp;";
        htmllocation = journal[0].getElementsByTagName("html")[0].firstChild.nodeValue;
        pdflocation = journal[0].getElementsByTagName("pdf")[0].firstChild.nodeValue;
    }
    else
    {
        document.getElementById('jtitle').innerHTML = "<strong>Error: journal has no
info!</strong>"
    }
}

```

Figure 11. Function JournalInfo

After looking at figure 11, some of the retrieved information is manipulated. Generally the information retrieved is in the form of a string or integer. Both can be appended to strings and appended to by strings, but only integers can be manipulated with mathematical operations. As seen in most of the Javascript code the string data that is being added to the retrieved data is

HTML code. The reasoning behind this will be apparent later on in the next chapter. There is also an "if" statement. The "if" statement tests to make sure there are child elements to retrieve data from. The result of no child elements returns an error message preventing confusing errors being shown to users.

Looping through the rows of a table is necessary and not hard to do. The number of rows to loop through is found by using 'journal.length'. Not only does the journal variable have access to XML commands, but also array commands as well. The "length" command returns the number of elements within the journal array which is also the number of rows within the table. The rest of the loop is standard Javascript loop: 'for (\$i=0; \$i<journal.length, \$i++) { }'. Each run through the loop is moving down one row through the table.

Returning Results

Have an interface that can retrieve data becomes fairly useless if the data can't be seen by the user. Throughout the Javascript code "document.getElementById('divid').innerHTML" is seen. Javascript is using divider tags within the HTML search page to dynamically change the page's contents. Setting up

the divider tags within the HTML page is discussed in the next chapter. Of course, "divid" needs to be changed to the appropriate divider tag id.

Why Not Use XPath?

No one can deny the usefulness of XPath or XQuery. Consider for a moment not all browsers are created equal. Microsoft Internet Explorer tends to be the exception. Being the only browser to support ActiveX, special cases have to be made when programming Javascript. Even with Internet Explorer 8 being released soon which is supposed to be compatible with the document namespace, quick internet searches show Microsoft to not follow web standards set forth by the W3C. Not everyone will be quick to switch to the new version of Internet Explorer. Tons of internet users are still using version 6 of the browser while version 7 has been around for over two years now.

Even though XPath works in all browsers, Internet Explorer is different enough to make XPath implementation more difficult than directly accessing the XML data. Take for instance a simple XPath query to retrieve the first journal in the journals table. The query would look like `"/journals/row[1]"` for most browsers, however Microsoft saw fit to not follow the W3C standards for querying.

Microsoft's query would look like `"/journals/row[0]"`. The W3C states an array of retrieved XML records starts with `"1"`, not `"0"`.

Not just the query statements are different, even the querying command is different. Document implementation's command is `"document.evaluate(xpath, xmldoc, nameset, result type, result)"` where `xpath` is the query, `xmldoc` is the document object holding the actual xml file, `namespace` is used in case namespaces occur in the XML document (generally null is specified instead), `result type` is the type of result desired (integer, string, etc), and `result` is the result set to append to (generally set to null). Internet Explorer uses `"xmlDoc.selectNodes(xpath)"` where `xmldoc` is the ActiveX object containing the XML document and `xpath` is the query.

The differences in XPath implementation can easily double the amount of Javascript required to make the database functional. Instead of having to worry about which browser is being used just for opening an XML document, the entire Javascript code would have to worry about the browser being used. Complexity of code will also increase. Trouble areas that may normally require one area to fix would require two areas to fix. During the SQL interpreter phase of the database itself, XPath may be re-evaluated.

CHAPTER FIVE

HYPertext MARKUP LANGUAGE INTERFACE

Introduction

The Javascript interface is designed to run only when required. To get the Javascript running HTML is used to call functions such as "initialize" in startup.js. HTML will also be used to search the database and display search results.

Referencing the Javascript

In interest of keeping source code simple, Javascript is kept separate from the HTML search page. So the search page can execute functions referenced within the page, a link to the Javascript source files needs to add to the search page's head. The link line is simple: `<script language="javascript" SRC="startup.js"></script>`. Each Javascript source file is referenced in the same manner, except replace "startup.js" with the other file names needing to be referenced.

Initial Startup

The demo search page contains elements that come from the database, such as journal information and author's first names, which require retrieving from the database upon opening the search page. Getting all of the

information from the database when the search page is opened is a simple modification to the body tag. In startup.js there is a function called "initialize". Initialize does everything required to show the initial search page. Starting initialize when the page loads is done by adding 'OnLoad="initialize()"' within the opening body tag. Instead of the opening body tag being "<body>", the opening body tag becomes '<body OnLoad="initialize()">'. If a different function needs to be referenced, change "initialize()" to the required startup function.

Making a Static Page Dynamic

HTML pages by nature are static. Once written, information within the page does not change. To have dynamic pages, a server side script is generally used. Since server side scripts can't be used, another way needs to be found to make a static page change based on user input. The HTML tag that allows static pages to become dynamic is "<div>". The divider tag breaks HTML pages up into sections. When given an id, Javascript can reference the tags while the page is being viewed and change the contents within the tag using HTML content. Content within the division can be blank or already have information.

Chapter 4 shows the command used to change the divider's content. The typical divider tag used in the search page look like `<div id="jtitle">`, replacing "jtitle" with the desired tag id. Divider tags can so much more than just separate portions of HTML code. For purposes of a search page, the basic function shown gets the job done.

What can be considered a good side effect of using a divider tag is the inability of a browser to show the source code of a divider tag once the information within the tag has been changed. All viewing source will do is show the original source code of the HTML file.

Linking to Javascript Functions

The search page has a link "Reset Search" to clear search results. The link references the Javascript function "initialize", which puts the search back into the condition the page was in when the search page was first loaded. In effect the function "initialize" has two functions: the first is to show the initial information for the search page and the second is to clear search results. Referencing a function in a link is done by putting `javascript: initialize()` within the href portion of a link tag in html. Replace "initialize()" with any other desired function. The complete reset link in the

search page is `Reset Search`.

Automatically Changing the Search Page

As the user starts searching for articles by selecting an author's first name, a drop down box for the author's last name automatically shows up. Once a last name is selected, a list of articles written by the author is shown. The search page automatically changes to user input without having to click on a button. Each of the two dropdown controls contains a piece of code that tells the search page what to do once the control has been changed. The code is simply

`onchange="createLastNameSelect(this.form)"`. This line of code and is one of the properties in the opening select tag in the HTML code. Without `onchange`, drop down box would do nothing when a first name is selected.

`createLastNameSelect` is a javascript function that creates another drop down box with a list of last names once a first name is selected. `this.form` passes into the `createLastName` function the entire form that the first name dropdown box belongs to, allowing Javascript to access the selected first name. The complete opening tag for the first name drop down box is `<select`

```
name="firstname"  
onChange="createLastNameSelect(this.form)"  
id="firstname">', all on one line.
```

There are two other properties of the select tag that makes changing the search page possible: "id" and "name". Both "id" and "name" are set to the same string "firstname". This is done for both dropdown boxes. Both properties are required for Javascript to function properly, and are used for compatibility of different browsers. Javascript uses the name property to be able to interact with the control through the form and the id property is used to identify the control when a reference to the control is not available, such as through "this.form". Some browsers, such as Internet Explorer, reference the control through the name property within the document namespace. For example using "document.firstname" allows javascript to access the properties for the first name dropdown in Internet Explorer. Browsers such as Firefox and Opera use the property "getElementById" discussed earlier.

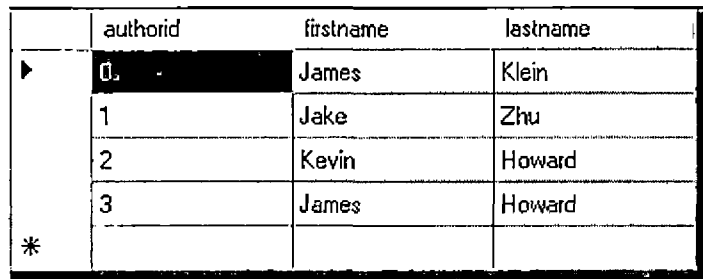
CHAPTER SIX

VISUAL BASIC .NET ADMINISTRATION PROGRAM

Introduction

In one way or another, the database needs to be administrated. The search page developed earlier can't administrate data, just display search results. Creating a program using Javascript and HTML isn't possible. Javascript cannot write to files, just read them. As discussed in the background, the security risks involved in allowing Javascript can be worse than damage caused by malware. In keeping with the requirement to operate regardless of operating system would lead to using Java. Since Java source is compiled at run time, source code can run as long as a Java Virtual Machine is installed. The problem with Java is the lack of one of the required controls: a datagrid. Datagrids are required to view the contents of each table. An example of a datagrid control is what's used when looking at the contents of a table within Microsoft Access, or the grid information and formulas are type into in Microsoft Excel. Datagrids are available for JAVA, but for a fee. The only freely available datagrid for JAVA would require special licensing for this project, which would take longer than

the time available to receive proper licensing. Given the issues with a datagrid control for JAVA, Microsoft Visual Basic .NET (VB) will be used. Figure 12 shows an example of a datagrid displaying the authors table.



	authorid	firstname	lastname
▶	0	James	Klein
	1	Jake	Zhu
	2	Kevin	Howard
	3	James	Howard
*			

Figure 12. Datagrid Control Example

The organization of this chapter is in sections, each detailing the steps required to complete specific tasks.

Required Imports

Visual Basic .NET uses the "imports" command to expose namespaces to a form's root namespace. One of the imports used to make working with XML files easier is "System.Data". Without the import, any command and object within the System.Data requires System.Data to precede the command. For instance the dataset object would have to be accessed through System.Data.dataaset rather than just plain dataset. Any other object and command that works with datasets such as datarow and datatable would also

require the full namespace to be accessed. Importing the namespace makes coding projects much easier and allows the source code to be much cleaner. The imports used in this project are System, System.Data, and System.Data.SqlClient.

Opening an Extensible Markup Language Document

The method to opening a XML document is done through a function. The function requires the name of the xml file and returns a dataset containing the contents of the xml file. The first part of the function is setting up the appropriate variables for the dataset and filestream. Creating a dataset is as simple as creating a new variable set to be a dataset. A new filestream is slightly more complex. Not only does a new variable have to be created, the XML file has to be opened in read only mode. Read only mode is used to prevent accidental changes to the data. The command to create the filestream is "Dim fsReadXml As New System.IO.FileStream(xmlfile, IO.FileMode.Open, IO.FileAccess.Read)". "xmlfile" is string containing the location of the file. If "xmlfile" only contains the name of the XML document being opened then the assumed folder the document resides in is the folder as the management program itself. If "xmlfile" contains a folder as well,

the exact folder name is used. For instance, if "c:\file.xml" is used the XML document needs to be located in the root folder of the "c" drive. "IO.FileMode.Open" tells the filestream that the file "xmlfile" is to be opened. "IO.FileAccess.Read" tells the filestream to only read from "xmlfile", but not write to the file.

At this point "xmlfile" is not opened, let alone been read. One command takes care of both tasks at the same time. "DataSet.readXML(xmlfile)" opens xmlfile and reads the entire contents of the XML document. Once the xml file has been fully read, the dataset contains several tables representing the XML data. One table represents the primary keys, another representing all the rows, and several tables representing the properties of each column. Methods for accessing the data and adding data are discussed throughout this chapter. The final task is to close the XML file using "FileStream.Close()" (Walkthrough, 2008). Figure 13 shows the complete XML File loading function.

```

Private Function OpenXML(ByVal xmlfile As String)
    Dim ds As New DataSet()
    Dim fsReadXml As New System.IO.FileStream(xmlfile, IO.FileMode.Open,
    IO.FileAccess.Read)
    Try
        ds.ReadXml(fsReadXml)
    Catch ex As Exception
        MessageBox.Show(ex.ToString())
        Return Nothing
    Exit Function
    Finally
        fsReadXml.Close()
    End Try
    Return ds
End Function

```

Figure 13. Extensible Markup Language Loading Function

Displaying Table Data

At this point all XML documents are opened and available in various datasets. Now the data needs to be displayed visually. Visual Basic can easily link a dataset to a DataGridView control. Using "DataGridView.DataSource = DataSet" tells the DataGridView control to use a dataset for a source of data to display. Setting a data source is not enough; the DataGridView control needs to know which portion of the XML data to display. Each XML document contains two main sections of XML: the table definition and the table data. Using 'DataGridView.DataMember = "row"'. "row" is the name of the group of XML tags being displayed in the DataGridView. The column header of the DataGridView control will show the column names that occur between each "row" tag. Using the authors table as an

example, the column header would contain "authorid", "firstname", and "lastname". Below the header, all data in each of the rows is shown. Figure 14 shows a DataGridView control displaying the Authors table.

	authorid	firstname	lastname
▶	0	James	Klein
	1	Jake	Zhu
	2	Kevin	Howard
	3	James	Howard
*			

Figure 14. DataGridView Control Displaying Authors Table Content

Enforcing Table Structures

Table structures need to be enforced as part of maintaining data integrity. Each column's settings are stored in the DataSet in a table named after each column. To retrieve the data type for the Authors table firstname column, 'DataSet.Tables ("firstname").rows(0).item("datatype")' is used. The row that each property is stored in is row 0. No other row number is used since each property table contains only one row. This section shows how each of a table's properties are maintained and used within the management program.

Auto Increment

Auto Increment allows a column of data to increase in number by one for each row of data added. Maintaining the increment is simply done by adding one to the largest number found in the auto increment column.

Programmatically the largest number is in the last row of data in the dataset. Getting the last number used is done by using `'dataset.tables("row").rows.count - 1'`. One is subtracted from the total number of rows to get the correct programmatic row number. Accessing rows in a dataset table starts with 0 and ends with $n - 1$ rows, similar to accessing data in an array. Creating the next number to be used in the auto increment is thus `'(dataset.tables("row").rows.count - 1) + 1'`.

Varchar Data

Varchar data is simply text with a certain length. DataGridView controls do not have a way to set the maximum length of characters that can be typed in for each cell. However data entry contains TextBox controls that can have maximum character lengths set. First the varchar data type needs to be retrieved before getting the length of the varchar string. Retrieving the data type was shown above. The next step is to get the length of the varchar. The best method for determining the length of the data is

testing for an open parenthesis starting from the end of the string "varchar(x)". Once the number is has been found, the maximum length of the TextBox control being used is set to the found number. Figure 15 shows the function used to get the varchar length.

```
Private Function GetVarcharLength(ByVal varchar As String)
    Dim startat As Integer = Len(varchar) - 1
    Dim foundnumber As String = varchar.Substring(startat, 1)
    Dim teststring As String
    startat -= 1
    Dim found = False
    While Not found
        teststring = varchar.Substring(startat, 1)
        If teststring = "(" Then
            found = True
        Else
            foundnumber = teststring + foundnumber
        End If
    End While
    Return Convert.ToInt16(foundnumber)s
End Function
```

Figure 15. Getting the Length of a Varchar

Integers

Integers are simply numbers. Enforcing only numbers in a column or textbox cannot be easily done without creating custom controls. With little knowledge in creating custom controls, another way has to be found to enforce integers. Visual Basic has a convert namespace that has commands to convert strings to integers. If 'Convert.ToInt16' causes an error, the information that

was entered is not a number. An error message can be displayed letting the user know to enter numbers only.

Primary Keys

Primary keys denote that each piece data in a particular column is unique. This project makes use of auto increment for primary keys except for the ArtAuth link table. To make sure each row in the ArtAuth table is unique, a loop is used to cycle through all rows in the table. Each row is compared to the data being entered into the table. If a row matches the data being entered, the new row I will not be add to the table. Figure 16 shows the loop to prevent duplicate primary keys.

```
For Each pkrow As DataRow In dsAuthors.Tables("row").Rows
  If pkrow.Item("authorid") = dgvAuthors.Item(0, nextrow).Value _
  And pkrow.Item("articleid") = Me.IsbArticles.SelectedIndices(counter) Then
    inpk = True
    Exit For
  End If
Next
```

Figure 16. Preventing Duplicate Information in Primary Key Columns

Foreign Keys

Foreign keys denote a column whose values reference the primary key column of another table. Data input forms uses combo boxes listing the information in the order the

information appears in the tables. Since the tables in the sample database use auto increment to create primary keys, the combo box indices match the primary key numbers in the combo box's supporting table. When storing a foreign key, simply store the selected index of the combo box containing the referenced table's information.

Notnull

Notnull determines if a column can contain null values. The management program makes use of this setting when a textbox control is changed or a row is added to one of the DataGridView controls. Testing for a column's notnull attribute is done by using

```
'DataSet.Tables("firstname").rows(0).item("notnull") = "yes"`. Changing "yes" to "no" will determine if notnull is turned off.
```

Default

The default table property determines what is inserted into each cell of a particular column is nothing is specified. In the management program the default data is shown in the data entry forms and when new rows are added to one of the DataSets. Getting the default settings is done by using

```
'DataSet.Tables("firstname").rows(0).item("default")'. A blank default means no default data is required. The
```

opposite is also true. A non-empty result means there is default data specified and the default data should be present in any forms used to enter data into the database.

Adding and Saving Data

Saving changes to data is an absolute must when managing a database. The first step to adding data is to make sure all values that are not null contain information. Figure 17 shows how to test for a null value.

```
If dsAuthors.Tables("firstname").Rows(0).Item("notnull") = "no" Then
    If Me.txtFirstName.Text = "" Then
        MsgBox("A first name is required")
        Exit Sub
    End If
End If
```

Figure 17. Testing For Null Values

Once all not null conditions are met, a method needs to be used to add the data to the XML document and to the DataGridView. Adding a row to the DataGridView programmatically is not allowed since the DataGridView control is bound to a data source. The next option is to change the data source. Changing the data source requires adding a new row to the "row" table in the associated DataSet. Creating a new row is done by using 'dim dsrow as new DataRow'. Populating the new row is as simple as

mapping the values of each control in the entry form to the heading of each column in the DataGridView control. Adding a first name to the new row for the authors table is done by using the following line of code:

```
'dsrow("firstname") = Me.txtFirstName.Text'. Once the DataRow has been filled with data, the row needs to be added to the corresponding Dataset using 'dsAuthors.Tables("row").Rows.Add(dsrow)'.
```

A new row of data has been added and the appropriate Dataset has been updated. At this point there is only two more tasks left: save the data to the XML file and update the datagridview. Saving to the XML file is done by a function called "WriteXML". WriteXML takes in a filename and a DataSet and writes the Dataset to the XML file. A function is used to prevent having to maintain three portions of identical code. The first step to writing to a XML file is to create a FileStream that holds the information necessary to access the desired XML file. Next a XMLTextWriter is created as an interface between the FileStream and DataSet. Finally write the XML data using the DataSet's "WriteXML" method and then close the XML file (DataSet.WriteXML, 2008). Figure 18 shows the complete WriteXML function.

```
Private Sub WriteXML(ByVal xmlfile As String, ByVal ds As DataSet)
    Dim myFileStream As New System.IO.FileStream(xmlfile,
System.IO.FileMode.Create)
    Dim myXMLWriter As New System.Xml.XmlTextWriter(myFileStream,
System.Text.Encoding.Unicode)
    ds.WriteXml(myXMLWriter)
    myXMLWriter.Close()
End Sub
```

Figure 18. WriteXML Function

The final task is to update the DataGridView. Two simple lines of code borrowed from opening an XML document will finish this task off. 'Me.dgvArtAuth.DataSource = dsArtAuth' and 'Me.dgvArtAuth.DataMember = "row"' refresh the DataGridView and complete adding and saving changes to XML files. Figure 19 shows the typical adding and saving data routine.

```

If dsAuthors.Tables("firstname").Rows(0).Item("notnull") = "no" Then
    If Me.txtFirstName.Text = "" Then
        MsgBox("A first name is required")
        Exit Sub
    End If
End If
If dsAuthors.Tables("lastname").Rows(0).Item("notnull") = "no" Then
    If Me.txtLastName.Text = "" Then
        MsgBox("A last name is required")
        Exit Sub
    End If
End If
Dim dsrow As DataRow
dsrow = dsAuthors.Tables("row").NewRow
Dim counter = 0
Dim nextrow As Integer = Me.dgvAuthors.Rows.Count - 1
dsrow("authorid") = dgvAuthors.Item(0, nextrow - 1).Value + 1
dsrow("firstname") = Me.txtFirstName.Text
dsrow("lastname") = Me.txtLastName.Text
dsAuthors.Tables("row").Rows.Add(dsrow)
WriteXML("authors.xml", Me.dsAuthors)
Me.dgvAuthors.DataSource = dsAuthors
Me.dgvAuthors.DataMember = "row"
While counter < Me.IsbArticles.SelectedItems.Count
    dsrow = dsArtAuth.Tables("row").NewRow
    Dim inpk As Boolean = False
    For Each pkrow As DataRow In dsAuthors.Tables("row").Rows
        If pkrow.Item("authorid") = dgvAuthors.Item(0, nextrow).Value _
            And pkrow.Item("articleid") = Me.IsbArticles.SelectedIndices(counter) Then
            inpk = True
            Exit For
        End If
    Next
    If Not inpk Then
        dsrow("articleid") = Me.IsbArticles.SelectedIndices(counter)
        dsrow("authorid") = dgvAuthors.Item(0, nextrow).Value
        dsArtAuth.Tables("row").Rows.Add(dsrow)
    End If
    counter += 1
End While
WriteXML("artauth.xml", Me.dsArtAuth)
Me.dgvArtAuth.DataSource = dsArtAuth
Me.dgvArtAuth.DataMember = "row"

```

Figure 19. Saving Changes to the Authors Table

Searching Tables

One "minor" but useful function of database management programs is the feature to be able to search data. Since the purpose of searching through data is to show search results, search will be done against the DataGridView controls. Once the search criteria are met, the row containing the data can be highlighted. To get started with the search, the column being search against is stored in a variable. The order the column header names appear in the ComboBox controls used to select which column is being searched corresponds to order the column headers appear in the DataGridView control. A selected index of 2 in a ComboBox is the same as the column number in the associated DataGridView. The variable containing the ComboBox selected index becomes the column numbers when retrieving the contents of a cell in the DataGridView.

The next to searching is looping through each cell in the selected column. Since two search criteria appear in the database management program, there are two conditions to be met. The first condition is if the first search criteria is found. The second condition is if the second search criteria is met as long as there is a second search criteria. If all of the search criteria are met, the

current row is selected. Once a row is selected, there is no reason to continue searching. The sub doing the search can be exited, or the loop exited, to allow continued use of the program. Figure 20 shows the sub used to search the authors table.

```
Private Sub cmdAuthorsFind_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdAuthorsFind.Click
    Cursor = Cursors.WaitCursor
    Dim columntouse As Integer = Me.cboAuthorsColumn.SelectedIndex
    Dim columnandtouse As Integer = Me.cboAuthorsColumnAnd.SelectedIndex
    For Me.authorscounter = 0 To Me.dgvAuthors.RowCount - 1
        If dgvAuthors.Item(columntouse, authorscounter).Value =
Me.txtAuthorsLookFor.Text Then
            If Me.txtArticlesLookForAnd.Text = "" Or Me.txtArtAuthLookForAnd.Text =
dgvAuthors.Item(columnandtouse, authorscounter).Value Then
                dgvAuthors.ClearSelection()
                dgvAuthors.Rows(authorscounter).Selected = True
                Me.cmdAuthorsFindNext.Enabled = True
                Cursor = Cursors.Arrow
                Exit Sub
            End If
        End If
    Next
    Cursor = Cursors.Arrow
    MsgBox("Can't find anything that matches the search criteria.")
End Sub
```

Figure 20. Authors Table Search Sub

CHAPTER SEVEN

CONCLUSIONS AND RECOMMENDATIONS

Introduction

Included in Chapter Seven was a presentation of the conclusions gleaned as a result of completing the project. Further, the recommendations extracted from the project are presented. Next lessons learned from the project. Lastly, the Chapter concludes with a summary.

Conclusions

The conclusions extracted from the project follows.

1. A database that can work regardless of platform is very possible. With further development the database might be able to operate without requiring a web browser.
2. The database has shown usefulness in other areas other than an academic journal's web site. During the development of this project, another XML database modeled after the database developed in this project was created for a local mortgage broker. After 2 months of use, the database is working excellently. The database doesn't store valuable information, but

does provide information regarding file storage and who interacts with the storage.

3. Even though the database works as required, the management program is considered a partial failure. Even though the Visual Basic .NET program works well with the database, the program does not work on any platform. Java can run regardless of platform as long as a Java virtual machine is installed. During future development, working out the datagrid control problem with Java is a must.

Recommendations

The recommendations resulting from the project follows.

1. The success of this project reflects the success of only one phase of the database. Even though this phase of the database is completed, development should not stop.
2. The project's readers and campus advisors preached working on the project a little at a time rather than all at once in a rush. There are two sides to every argument. Those giving the advice are correct in some aspects. Rushing

is stressful and can produce shoddy work.

However, the best work done on this project was done during the last minute rush. Mistakes made throughout developing the project were caught during the rush before the project committee could voice concerns. Out of respect for the committee, the last minute rush is truly unfair to the committee being able to give sound criticism and advice concerning the project.

3. SQL, XQuery, and XPath compatibility should also be a top priority for future development. Even though the management program does allow for limited searches, being able to run custom queries using SQL, XQuery, and XPath will allow for development of more feature filled programs built on top of the database and more user friendly data management software.
4. The data contained within the database is not encrypted for security; hence the acknowledgement that the data stored in the database can be viewed by anyone and doesn't contain any company sensitive data. Creating an encryption algorithm will help create more

utility for the database, adding to the database's potential success.

Lessons Learned

The project itself is a lesson learned. Using XML to store data, Javascript to retrieve data, and HTML to display search results to effectively create a database that works on any platform was never considered. After a little command search on the W3C web site and some testing, a simple search page turned into a database system.

While testing the operation of the Javascript interface, a problem came up that kept the search page from operating correctly. When the search page initially loaded, the journal table information displayed just fine. The drop down box containing a list of author's first names was empty. After two days of trying to figure out the cause of the problem, the program seemed to have a problem that is impossible to solve. All attempts to find the culprit failed. Message boxes showed all variables contained the correct information. With all the searching and code rewriting, the problem wasn't being solved. Out of everything tried, using separate XML objects for each XML file being opened was never tried. Once authors.xml

was opened in a separate XML object from that of journal.xml, the search page behaved correctly.

Originally the same XML object was being used for both files. Once journal.xml was finished being used, the XML object was recreated to work with authors.xml. This was done to save on memory requirements. Javascript doesn't like recycling these types of objects. Once the XML object is set to a XML file, there is no changing the file the object works with. Why Javascript behaves in this fashion is unclear. Searching turned up no answers. Being required to use multiple XML objects is actually the better way to go. When linking multiple tables together, multiple XML objects are required. Even if a single object was planned to be reused for each file, linking tables together would prevent reuse of the object.

Summary

Chapter Five reviewed the conclusions extracted from the project. Next, the recommendations derived from the project were presented. Lastly, the lessons learned from the project.

A successful attempt was made to create a database that could work without the requirement of a specific platform. Data is stored in a normalized format and is

capable of being retrieved in any order and format. Utilizing functions from the Javascript interface, the user interface is easily created using HTML. Since Javascript cannot write files to a client's system, and Java does not contain a standard datagrid control, Visual Basic .NET was used to create the management program. The management program provides an easy way to manage the XML database without having to directly edit the XML files. This phase of the database development is done, paving the way to the next phase of the database: a SQL interpreter.

APPENDIX A
EXTENSIBLE MARKUP LANGUAGE DATABASE

journal.xml

```
<?xml version="1.0" encoding="utf-8"?>
<journal>
  <tabledef>
    <pk>journalid</pk>
    <journalid datatype="autoincrement" notnull="yes" default="" />
    <title datatype="varchar(20)" notnull="yes" default="title" />
    <volume datatype="integer" notnull="yes" default="1" />
    <number datatype="integer" notnull="yes" default="1" />
    <year datatype="integer" notnull="yes" default="2005" />
    <issn datatype="varchar(10)" notnull="yes" default="0000-0000" />
    <description datatype="text" notnull="yes" default="description" />
    <html datatype="varchar(30)" notnull="no" default="" />
    <pdf datatype="varchar(30)" notnull="no" default="" />
  </tabledef>
  <row>
    <journalid>0</journalid>
    <title>ASBBS E-Journal</title>
    <volume>1</volume>
    <number>1</number>
    <year>2005</year>
    <issn>1557-5004</issn>
    <description>An Official Online Journal of American Society of Business and
Behavioral Sciences</description>
    <html>html</html>
    <pdf>pdf</pdf>
  </row>
</journal>
```

articles.xml

```
<?xml version="1.0" encoding="utf-8"?>
<articles>
  <tabledef>
    <pk>articleid</pk>
    <articleid datatype="autoincrement" notnull="yes" default="" />
    <title datatype="varchar(20)" notnull="yes" default="" />
    <html datatype="varchar(30)" notnull="no" default="" />
    <pdf datatype="varchar(30)" notnull="no" default="" />
    <pages datatype="integer" notnull="yes" default="1" />
    <journalid datatype="integer" notnull="yes" default="" />
    <fk table="journal" column="journalid">journalid</fk>
  </tabledef>
  <row>
    <articleid>0</articleid>
    <title>Follow The Leader</title>
    <html>follow.html</html>
    <pdf>follow.pdf</pdf>
    <pages>4</pages>
    <journalid>0</journalid>
  </row>
  <row>
    <articleid>1</articleid>
    <title>Info-648</title>
    <html>648.html</html>
    <pdf>648.pdf</pdf>
    <pages>2</pages>
    <journalid>0</journalid>
  </row>
  <row>
    <articleid>2</articleid>
    <title>Computers</title>
    <html>computers.html</html>
    <pdf>computers.pdf</pdf>
    <pages>7</pages>
    <journalid>0</journalid>
  </row>
</articles>
```

authors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<authors>
  <tabledef>
    <pk>authorid</pk>
    <authorid datatype="autoincrement" notnull="yes" default="" />
    <firstname datatype="varchar(15)" notnull="yes" default="" />
    <lastname datatype="varchar(15)" notnull="yes" default="" />
  </tabledef>
  <row>
    <authorid>0</authorid>
    <firstname>James</firstname>
    <lastname>Klein</lastname>
  </row>
  <row>
    <authorid>1</authorid>
    <firstname>Jake</firstname>
    <lastname>Zhu</lastname>
  </row>
  <row>
    <authorid>2</authorid>
    <firstname>Kevin</firstname>
    <lastname>Howard</lastname>
  </row>
  <row>
    <authorid>3</authorid>
    <firstname>James</firstname>
    <lastname>Howard</lastname>
  </row>
</authors>
```

artauth.xml

```
<?xml version="1.0" encoding="utf-8"?>
<artauth>
  <tabledef>
    <pk>articleid</pk>
    <pk>authorid</pk>
    <articleid datatype="integer" notnull="yes" default="" />
    <authorid datatype="integer" notnull="yes" default="" />
    <fk table="authors" column="authorid">authorid</fk>
    <fk table="articles" column="articleid">articleid</fk>
  </tabledef>
  <row>
    <articleid>0</articleid>
    <authorid>0</authorid>
  </row>
  <row>
    <articleid>0</articleid>
    <authorid>1</authorid>
  </row>
  <row>
    <articleid>1</articleid>
    <authorid>1</authorid>
  </row>
  <row>
    <articleid>2</articleid>
    <authorid>2</authorid>
  </row>
  <row>
    <articleid>2</articleid>
    <authorid>3</authorid>
  </row>
</artauth>
```


APPENDIX B
JAVASCRIPT CODE

startup.js

```
var firstnames = new Array();
var authartXML;
var jvolume = "";
var jyear = "";
var jnumber = "";
var jjtitle = "";
var htmllocation = "";
var pdflocation = "";
var xmlDoc;
var fnameXML;
var fname;
var lname;
var authIdXML;
var authID;
var articlesXML;
var articles = new Array();
var lnameXML;
var fnameForm;

function initialize()
{
    document.getElementById('lastNameSearch').innerHTML = "";

    var browser = "none";

    if ((typeof document.implementation != 'undefined') && (typeof
document.implementation.createDocument != 'undefined'))
        { browser = "netscape"; }
    else if (typeof window.ActiveXObject != 'undefined')
        { browser = "ie"; }

    switch (browser)
    {
        case "ie":
        {
            xmlDoc = new ActiveXObject('Microsoft.XMLDOM');
            xmlDoc.async = false;
            xmlDoc.onreadystatechange = function ()
                { if (xmlDoc.readyState == 4) JournalInfo() };
            fnameXML = new ActiveXObject('Microsoft.XMLDOM');
            fnameXML.async = false;
            fnameXML.onreadystatechange = function ()
                { if (fnameXML.readyState == 4) LoadFirstName() };
            authartXML = new ActiveXObject('Microsoft.XMLDOM');
            authartXML.async = false;
            articlesXML = new ActiveXObject('Microsoft.XMLDOM');
            articlesXML.async = false;
            articlesXML.onreadystatechange = function ()
                { if (articlesXML.readyState == 4) listArticles() };
            break
        }
    }
}
```

```

case "netscape":
{
    xmlDoc = document.implementation.createDocument("", "doc", null);
    xmlDoc.onload = JournalInfo;
    fnameXML = document.implementation.createDocument("", "doc", null)
    fnameXML.onload = LoadFirstName;
    authartXML = document.implementation.createDocument("", "doc", null)
    articlesXML = document.implementation.createDocument("", "doc", null)
    articlesXML.onload = listArticles;
    break
}
}
xmlDoc.load("journal.xml");
fnameXML.load("authors.xml");
authartXML.load("artauth.xml");
articlesXML.load("articles.xml");
}

function isInArray(testArray, lookFor)
{
    var returns = false;
    for (var j = 0; j < testArray.length; j++)
    {
        if (testArray[j] == lookFor) returns = true;
    }
    return returns;
}

function JournalInfo()
{
    var journal =
xmlDoc.getElementsByTagName("journal")[0].getElementsByTagName("row");

    if (journal.length > 0 )
    {
        document.getElementById('jtitle').innerHTML =
journal[0].getElementsByTagName("title")[0].firstChild.nodeValue + "&nbsp;&nbsp;&nbsp;";
        jjtitle = journal[0].getElementsByTagName("title")[0].firstChild.nodeValue;
        jvolume = journal[0].getElementsByTagName("volume")[0].firstChild.nodeValue;
        jnumber = journal[0].getElementsByTagName("number")[0].firstChild.nodeValue;
        jyear = journal[0].getElementsByTagName("year")[0].firstChild.nodeValue;
        document.getElementById('volume').innerHTML = "Volume " + jvolume + ", No. " +
jnumber + ", " + jyear + "&nbsp;&nbsp;&nbsp;";
        document.getElementById('issn').innerHTML = "ISSN: " +
journal[0].getElementsByTagName("issn")[0].firstChild.nodeValue + "&nbsp;&nbsp;&nbsp;";
        document.getElementById('description').innerHTML =
journal[0].getElementsByTagName("description")[0].firstChild.nodeValue + "&nbsp;&nbsp;&nbsp;";
        htmllocation = journal[0].getElementsByTagName("html")[0].firstChild.nodeValue;
        pdflocation = journal[0].getElementsByTagName("pdf")[0].firstChild.nodeValue;
    }
    else
    {

```

```

        document.getElementById('jtitle').innerHTML = "<strong>Error: journal has no
info!</strong>"
    }
}

function LoadFirstName()
{
    var fnames =
fnameXML.getElementsByTagName("authors")[0].getElementsByTagName("row");
    for (var i = 0; i < fnames.length; i++)
    {
        var testfor = fnames[i].getElementsByTagName("firstname")[0].firstChild.nodeValue;
        if (isArray(firstnames, testfor) == false)
firstnames.push(fnames[i].getElementsByTagName("firstname")[0].firstChild.nodeValue);
    }
    firstnames.sort();
    var searcher = "<form name='searchArticles' method='get' action='\">";
    searcher += "First Name: <select name='firstname'
onChange='createLastNameSelect(this.form)' id='firstname'>";
    searcher += "<option></option>";
    for (var i = 0; i < firstnames.length; i++) searcher += "<option value='\" + firstnames[i] +
'\">\" + firstnames[i] + "</option>";
    searcher += "</select>";
    searcher += "</form>";
    document.getElementById('firstNameSearch').innerHTML = searcher;
}

function listArticles()
{
    var color = 0;
    var articles =
articlesXML.getElementsByTagName("articles")[0].getElementsByTagName("row");
    var combotable =
authartXML.getElementsByTagName("artauth")[0].getElementsByTagName("row");
    var names =
fnameXML.getElementsByTagName("authors")[0].getElementsByTagName("row");

    var searcher = "<table borderColor='\"#c0c0c0' cellSpacing='1' width='100%'
border='2'><tr><td align='left'>";

    for (var i = 0; i < articles.length; i++)
    {
        searcher += "<tr><td align='left'";
        if (color == 0)
        {
            searcher += ">";
            color++;
        } else {
            searcher += " bgcolor='\"#ffffc1'>";
            color--;
        }
    }

    searcher += "\"<a href='\" + htmllocation;

```

```

searcher += articles[i].getElementsByTagName("html")[0].firstChild.nodeValue;
searcher += "\>";
searcher += articles[i].getElementsByTagName("title")[0].firstChild.nodeValue;
searcher += "</a>\". <strong>Author(s):</strong> ";

var articleid = articles[i].getElementsByTagName("articleid")[0].firstChild.nodeValue;
var authids = new Array();

for (j = 0; j < combotable.length; j++)
{
    var testid =
combotable[j].getElementsByTagName("articleid")[0].firstChild.nodeValue;
    if (testid == articleid)
    {
authids.push(combotable[j].getElementsByTagName("authorid")[0].firstChild.nodeValue); }
    }

for (j = 0; j < names.length; j++)
{
    var testid = names[j].getElementsByTagName("authorid")[0].firstChild.nodeValue;

    for (k = 0; k<authids.length; k++)
    {
        if (testid == authids[k])
        {
            searcher +=
names[j].getElementsByTagName("firstname")[0].firstChild.nodeValue;
            searcher += " ";
            searcher +=
names[j].getElementsByTagName("lastname")[0].firstChild.nodeValue;
            if (k < authids.length - 1)
            {
                searcher += ", ";
            } else if (k == authids.length - 1) {
                searcher += ". ";
            }
        }
    }
}

searcher += "<strong><i>" + jjtitle + "</i></strong>, ";
searcher += jyear + ", Vol. " + jvolume + " No. ";
searcher += jnumber + ", ";
searcher += articles[i].getElementsByTagName("pages")[0].firstChild.nodeValue;
searcher += "p<br>";
searcher += "<a href=\"\" + pdflocation;
searcher += articles[i].getElementsByTagName("pdf")[0].firstChild.nodeValue;
searcher += "\>PDF Version</a></td></tr>";
}
searcher += "</table>";
document.getElementById('articleList').innerHTML = searcher;
}

```

lastname.js

```
function createLastNameSelect(nameForm)
{
    fnameForm = nameForm;
    var browser = "none";

    if ((typeof document.implementation != 'undefined') && (typeof
document.implementation.createDocument != 'undefined'))
    { browser = "netscape"; }
    else if (typeof window.ActiveXObject != 'undefined')
    { browser = "ie"; }

    switch (browser)
    {
        case "ie":
        {
            lnameXML = new ActiveXObject("Microsoft.XMLDOM");
            lnameXML.async = false;
            lnameXML.onreadystatechange = function ()
                { if (lnameXML.readyState == 4) LoadLastName() };
            break
        }
        case "netscape":
        {
            lnameXML = document.implementation.createDocument("", "doc", null)
            lnameXML.onload = LoadLastName;
            break
        }
    }
    lnameXML.load("authors.xml");
}

function LoadLastName()
{
    var lastnames = new Array();
    var lnames =
lnameXML.getElementsByTagName("authors")[0].getElementsByTagName("row");
    for (var i = 0; i < lnames.length; i++)
    {
        var testfor = lnames[i].getElementsByTagName("firstname")[0].firstChild.nodeValue;
        if (testfor == fnameForm.firstname.value)
        {
            testfor = lnames[i].getElementsByTagName("lastname")[0].firstChild.nodeValue;
            if (isArray(lastnames, testfor) == false)
lnameXML.getElementsByTagName("lastname")[0].firstChild.nodeValue);
        }
    }
    lastnames.sort();
    var searcher = "<form name='searchLastName' method='get' action='\">";
    searcher += "Last Name: <select name='lastname' onChange='createArticleList()'\"
id='lastname'>";
    searcher += "<option></option>";
}
```

```
for (var i = 0; i < lastnames.length; i++)
{
    searcher += "<option value=\" + lastnames[i] + "\"> + lastnames[i] + "</option>";
}
searcher += "</select>";
searcher += "</form>";
document.getElementById('lastNameSearch').innerHTML = searcher;
}
```

search.js

```
function createArticleList()
{
    document.getElementById("articleList").innerHTML = "";
    fname = "";
    lname = "";
    articles.length = 0;
    fname = document.getElementById("firstname").value;
    lname = document.getElementById("lastname").value;

    var browser = "none";

    if ((typeof document.implementation != 'undefined') && (typeof
document.implementation.createDocument != 'undefined'))
    { browser = "netscape"; }
    else if (typeof window.ActiveXObject != 'undefined')
    { browser = "ie"; }

    switch (browser)
    {
        case "ie":
        {
            authIdXML = new ActiveXObject('Microsoft.XMLDOM');
            authIdXML.async = false;
            authIdXML.onreadystatechange = function ()
                { if (authIdXML.readyState == 4) getAuthorID() };
            authartXML = new ActiveXObject('Microsoft.XMLDOM');
            authartXML.async = false;
            authartXML.onreadystatechange = function ()
                { if (authartXML.readyState == 4) getArticleID() };
            articlesXML = new ActiveXObject('Microsoft.XMLDOM');
            articlesXML.async = false;
            articlesXML.onreadystatechange = function ()
                { if (articlesXML.readyState == 4) showArticles() };
            break
        }
        case "netscape":
        {
            authIdXML = document.implementation.createDocument("", "doc", null)
            authIdXML.onload = getAuthorID;
            authartXML = document.implementation.createDocument("", "doc", null)
            authartXML.onload = getArticleID;
            articlesXML = document.implementation.createDocument("", "doc", null)
            articlesXML.onload = showArticles;
            break
        }
    }
    authIdXML.load("authors.xml");
    authartXML.load("artauth.xml");
    articlesXML.load("articles.xml");
}
```



```

function getAuthorID()
{
    var authors =
    authIdXML.getElementsByTagName("authors")[0].getElementsByTagName("row");

    for (var i = 0; i < authors.length; i++)
    {
        var nodeFName =
        authors[i].getElementsByTagName("firstname")[0].firstChild.nodeValue;
        var nodeLName =
        authors[i].getElementsByTagName("lastname")[0].firstChild.nodeValue;
        if ((nodeFName == fname) && (nodeLName == lname))
        {
            authID = authors[i].getElementsByTagName("authorid")[0].firstChild.nodeValue;
            break
        }
    }
}

function getArticleID()
{
    var authart =
    authartXML.getElementsByTagName("artauth")[0].getElementsByTagName("row");

    for (var i = 0; i < authart.length; i++)
    {
        var nodeAuthID =
        authart[i].getElementsByTagName("authorid")[0].firstChild.nodeValue;
        if (nodeAuthID == authID)
        {
            articles.push(authart[i].getElementsByTagName("articleid")[0].firstChild.nodeValue);
        }
    }
    articles.sort();
}

function showArticles()
{
    var resultsHTML = "";
    var color = 0;
    var articleNodes =
    articlesXML.getElementsByTagName("articles")[0].getElementsByTagName("row");
    var authart =
    authartXML.getElementsByTagName("artauth")[0].getElementsByTagName("row");
    var authorsNames =
    authIdXML.getElementsByTagName("authors")[0].getElementsByTagName("row");

    var v = "<table borderColor=\\'#c0c0c0\\' cellSpacing=\\'1\\' width=\\'100%\\'
border=\\'2\\'><tr><td align=\\'left\\'>";
    //loop through the articleXML
    for (var i = 0; i < articleNodes.length; i++)
    {

```

```

var nodeArtID =
articleNodes[i].getElementsByTagName("articleid")[0].firstChild.nodeValue;

for (var j = 0; j < articles.length; j++)
{
var arttestid = articles[j];
if (nodeArtID == arttestid)
{
resultsHTML += "<tr><td align='left'";
if (color == 0)
{
resultsHTML += ">";
color++;
} else {
resultsHTML += " bgcolor='\"#ffffcc\">";
color--;
}
resultsHTML += "<a href='\"";
resultsHTML +=
articleNodes[i].getElementsByTagName("html")[0].firstChild.nodeValue;
resultsHTML += "\">";
resultsHTML +=
articleNodes[i].getElementsByTagName("title")[0].firstChild.nodeValue;
resultsHTML += "</a>. Author(s): ";

for (var k = 0; k < authart.length; k++)
{
var nodeArticles =
authart[k].getElementsByTagName("articleid")[0].firstChild.nodeValue;

if (nodeArticles == nodeArtID)
{
var nodeAuthorID =
authart[k].getElementsByTagName("authorid")[0].firstChild.nodeValue;

for (var l = 0; l < authorsNames.length; l++)
{
var nodeAuthorNameID =
authorsNames[l].getElementsByTagName("authorid")[0].firstChild.nodeValue;

if (nodeAuthorNameID == nodeAuthorID)
{
resultsHTML +=
authorsNames[l].getElementsByTagName("firstname")[0].firstChild.nodeValue;
resultsHTML += " "
resultsHTML +=
authorsNames[l].getElementsByTagName("lastname")[0].firstChild.nodeValue;
resultsHTML += ". ";
}
}
}
}
}
}

```


APPENDIX C
HYPERTEXT MARKUP LANGUAGE CODE

index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Projects</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link href="format.css" rel="stylesheet" type="text/css">
</head>
<script language="javascript" SRC="startup.js"></script>
<script language="javascript" SRC="lastname.js"></script>
<script language="javascript" SRC="search.js"></script>
<body OnLoad="initialize()">
<table bordercolor="#111111" height="2" cellspacing="0" cellpadding="0" width="100%"
border="0">
  <tr>
    <td align="left" bgcolor="#ffffcc" rowspan="2" width="200">
      
    </td>
    <td align="center" bgcolor="#ffffcc"><h1>
      <div id="jtitle" align="center"></div>
    </h1></td>
    <td bgcolor="#ffffcc" width="200" valign="bottom"><div id="volume"
align="right"></div></td>
  </tr>
  <tr>
    <td bgcolor="#ffffcc" align="center"><h2><div align="center"
id="description"></div></h2>
    </td>
    <td bgcolor="#ffffcc"><div id="issn" align="right"></div></td>
  </tr>
  <tr>
    <td align="left" valign="top">
      <table bgcolor=#ffffcc width="100%">
        <tr>
          <td>
            <p><font size="4"><strong>Article Search:</strong></font></p>
            <div id="firstNameSearch"></div>
            <div id="lastNameSearch"></div>
            <p><a href="javascript: initialize()">Reset Search</a></p>
          </td>
        </tr>
      </table>
    </td>
    <td colspan="3" align="center" valign="top">
      <table borderColor="#3366cc" cellspacing="1" width="100%" border="3">
        <tr>
          <td valign="top" align="left" bgcolor="#FFFFFF">
            <div id="articleList"><p align="center">LOADING<br></p></div>
          </td>
        </tr>
      </table>
    </td>
  </tr>
</table>
```

```
        </table>
      </td>
    </tr>
  </table>

</body>
</html>
```

APPENDIX D
VISUAL BASIC .NET CODE

frmMain.vb

```
Imports System
Imports System.Data
Imports System.Data.SqlClient

Public Class frmMain
    Dim dsAuthors As New DataSet()
    Dim dsArticles As New DataSet()
    Dim dsArtAuth As New DataSet()
    Dim dsJournal As New DataSet()
    Dim authorscounter As Integer = 0
    Dim articlescounter As Integer = 0
    Dim artauthcounter As Integer = 0
    Dim journalcounter As Integer = 0

    Private Function OpenXML(ByVal xmlfile As String)
        Dim ds As New DataSet()
        Dim fsReadXml As New System.IO.FileStream(xmlfile, IO.FileMode.Open,
IO.FileAccess.Read)
        Try
            ds.ReadXml(fsReadXml)
        Catch ex As Exception
            MessageBox.Show(ex.ToString())
            Return Nothing
        Exit Function
        Finally
            fsReadXml.Close()
        End Try
        Return ds
    End Function

    Private Function GetVarcharLength(ByVal varchar As String)
        Dim startat As Integer = Len(varchar) - 1
        Dim foundnumber As String = varchar.Substring(startat, 1)
        Dim teststring As String
        startat -= 1
        Dim found = False
        While Not found
            teststring = varchar.Substring(startat, 1)
            If teststring = "(" Then
                found = True
            Else
                foundnumber = teststring + foundnumber
            End If
        End While
        Return Convert.ToInt16(foundnumber)
    End Function

    Private Sub WriteXML(ByVal xmlfile As String, ByVal ds As DataSet)
        Dim myFileStream As New System.IO.FileStream(xmlfile, System.IO.FileMode.Create)
        Dim myXMLWriter As New System.Xml.XmlTextWriter(myFileStream,
System.Text.Encoding.Unicode)
```



```

        ds.WriteXml(myXMLWriter)
        myXmlWriter.Close()
    End Sub

Private Sub frmMain_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles MyBase.Load
        dsAuthors = OpenXML("authors.xml")
        dsArticles = OpenXML("articles.xml")
        dsArtAuth = OpenXML("artauth.xml")
        dsJournal = OpenXML("journal.xml")

        Me.dgvArtAuth.DataSource = dsArtAuth
        Me.dgvArticles.DataSource = dsArticles
        Me.dgvAuthors.DataSource = dsAuthors
        Me.dgvJournal.DataSource = dsJournal

        Me.dgvArtAuth.DataMember = "row"
        Me.dgvArticles.DataMember = "row"
        Me.dgvAuthors.DataMember = "row"
        Me.dgvJournal.DataMember = "row"

        Me.dgvAuthors.Columns(0).ReadOnly = True
        Me.dgvArticles.Columns(0).ReadOnly = True
        Me.dgvJournal.Columns(0).ReadOnly = True

        For Each column As DataGridViewColumn In dgvArticles.Columns
            Me.cboArticlesColumn.Items.Add(column.HeaderText)
            Me.cboArticlesColumnAnd.Items.Add(column.HeaderText)
        Next
        For Each column As DataGridViewColumn In dgvAuthors.Columns
            Me.cboAuthorsColumn.Items.Add(column.HeaderText)
            Me.cboAuthorsColumnAnd.Items.Add(column.HeaderText)
        Next
        For Each column As DataGridViewColumn In dgvJournal.Columns
            Me.cboJournalColumn.Items.Add(column.HeaderText)
            Me.cboJournalColumnAnd.Items.Add(column.HeaderText)
        Next
        For Each column As DataGridViewColumn In dgvArtAuth.Columns
            Me.cboArtAuthColumn.Items.Add(column.HeaderText)
            Me.cboArtAuthColumnAnd.Items.Add(column.HeaderText)
        Next

        For Each xmlrow As DataGridViewRow In dgvAuthors.Rows
            If Not IsDBNull(xmlrow.Cells(1).Value) Or xmlrow.Cells(1).Value <> "" Then
                Me.IsbAuthors.Items.Add(xmlrow.Cells(1).Value + " " + xmlrow.Cells(2).Value)
            End If
        Next
        For counter As Integer = 0 To dgvArticles.Rows.Count - 1
            If dgvArticles.Item(1, counter).Value <> "" Then
                Me.IsbArticles.Items.Add(dgvArticles.Item(1, counter).Value)
            End If
        Next
    End Sub

```

```

For counter As Integer = 0 To dgvJournal.Rows.Count - 1
    If dgvJournal.Item(1, counter).Value <> "" Then
        Me.cboJournal.Items.Add(dgvJournal.Item(1, counter).Value)
    End If
Next

Me.cmdArtAuthFindNext.Enabled = False
Me.cmdArticlesFindNext.Enabled = False
Me.cmdAuthorsFindNext.Enabled = False
Me.cmdJournalFindNext.Enabled = False
End Sub

Private Sub cmdAuthorsSave_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdAuthorsSave.Click
    Cursor = Cursors.WaitCursor
    WriteXML("authors.xml", Me.dgvAuthors.DataSource)
    Cursor = Cursors.Arrow
End Sub

Private Sub dgvAuthors_CellDoubleClick(ByVal sender As Object, ByVal e As
System.Windows.Forms.DataGridViewCellEventArgs) Handles dgvAuthors.CellDoubleClick
    Dim currentrow As Integer = dgvAuthors.CurrentRow.Index
    Dim numrows As Integer = dgvAuthors.RowCount - 1
    Dim lastnumber As Integer = dgvAuthors.Item(0, numrows - 1).Value
    If currentrow = numrows Then
        dgvAuthors.Item(0, numrows).Value = lastnumber + 1
    End If
End Sub

Private Sub dgvArticles_CellDoubleClick(ByVal sender As Object, ByVal e As
System.Windows.Forms.DataGridViewCellEventArgs) Handles dgvArticles.CellDoubleClick
    Dim currentrow As Integer = dgvArticles.CurrentRow.Index
    Dim numrows As Integer = dgvArticles.RowCount - 1
    Dim lastnumber As Integer = dgvArticles.Item(0, numrows - 1).Value
    If currentrow = numrows Then
        dgvArticles.Item(0, numrows).Value = lastnumber + 1
    End If
End Sub

Private Sub dgvJournal_CellDoubleClick(ByVal sender As Object, ByVal e As
System.Windows.Forms.DataGridViewCellEventArgs) Handles dgvJournal.CellDoubleClick
    Dim currentrow As Integer = dgvJournal.CurrentRow.Index
    Dim numrows As Integer = dgvJournal.RowCount - 1
    Dim lastnumber As Integer = dgvJournal.Item(0, numrows - 1).Value
    If currentrow = numrows Then
        dgvJournal.Item(0, numrows).Value = lastnumber + 1
    End If
End Sub

Private Sub cmdArticlesSave_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdArticlesSave.Click
    Cursor = Cursors.WaitCursor

```

```

        WriteXML("articles.xml", Me.dgvArticles.DataSource)
        Cursor = Cursors.Arrow
    End Sub

Private Sub cmdArtAuthSave_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdArtAuthSave.Click
    Cursor = Cursors.WaitCursor
    WriteXML("artauth.xml", Me.dgvArtAuth.DataSource)
    Cursor = Cursors.Arrow
End Sub

Private Sub cmdJournalSave_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdJournalSave.Click
    Cursor = Cursors.WaitCursor
    WriteXML("journal.xml", Me.dgvJournal.DataSource)
    Cursor = Cursors.Arrow
End Sub

Private Sub cmdAuthorsFind_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdAuthorsFind.Click
    Cursor = Cursors.WaitCursor
    Dim columntouse As Integer = Me.cboAuthorsColumn.SelectedIndex
    Dim columnandtouse As Integer = Me.cboAuthorsColumnAnd.SelectedIndex
    For Me.authorscounter = 0 To Me.dgvAuthors.RowCount - 1
        If dgvAuthors.Item(columntouse, authorscounter).Value = Me.txtAuthorsLookFor.Text
    Then
        If Me.txtArticlesLookForAnd.Text = "" Or Me.txtArtAuthLookForAnd.Text =
dgvAuthors.Item(columnandtouse, authorscounter).Value Then
            dgvAuthors.ClearSelection()
            dgvAuthors.Rows(authorscounter).Selected = True
            Me.cmdAuthorsFindNext.Enabled = True
            Cursor = Cursors.Arrow
            Exit Sub
        End If
    End If
    Next
    Cursor = Cursors.Arrow
    MsgBox("Can't find anything that matches the search criteria.")
End Sub

Private Sub cmdArticlesFind_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdArticlesFind.Click
    Cursor = Cursors.WaitCursor
    Dim columntouse As Integer = Me.cboArticlesColumn.SelectedIndex
    Dim columnandtouse As Integer = Me.cboArticlesColumnAnd.SelectedIndex
    For Me.articlescounter = 0 To Me.dgvArticles.RowCount - 1
        If dgvArticles.Item(columntouse, articlescounter).Value = Me.txtArticlesLookFor.Text
    Then
        If Me.txtArticlesLookForAnd.Text = "" Or Me.txtArticlesLookForAnd.Text =
dgvArticles.Item(columnandtouse, articlescounter).Value Then
            dgvArticles.ClearSelection()
            dgvArticles.Rows(articlescounter).Selected = True
            Me.cmdArticlesFindNext.Enabled = True

```

```

        Cursor = Cursors.Arrow
        Exit Sub
    End If
End If
Next
Cursor = Cursors.Arrow
MsgBox("Can't find anything that matches the search criteria.")
End Sub

Private Sub cmdArtAuthFind_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdArtAuthFind.Click
    Cursor = Cursors.WaitCursor
    Dim columntouse As Integer = Me.cboArtAuthColumn.SelectedIndex
    Dim columnandtouse As Integer = Me.cboArtAuthColumnAnd.SelectedIndex
    For Me.artauthcounter = 0 To Me.dgvArtAuth.RowCount - 1
        If dgvArtAuth.Item(columntouse, artauthcounter).Value = Me.txtArtAuthLookFor.Text
Then
            If Me.txtArtAuthLookForAnd.Text = "" Or Me.txtArtAuthLookForAnd.Text =
dgvArtAuth.Item(columnandtouse, artauthcounter).Value Then
                dgvArtAuth.ClearSelection()
                dgvArtAuth.Rows(artauthcounter).Selected = True
                Me.cmdArtAuthFindNext.Enabled = True
                Cursor = Cursors.Arrow
                Exit Sub
            End If
        End If
    Next
    Cursor = Cursors.Arrow
    MsgBox("Can't find anything that matches the search criteria.")
End Sub

Private Sub cmdJournalFind_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdJournalFind.Click
    Cursor = Cursors.WaitCursor
    Dim columntouse As Integer = Me.cboJournalColumn.SelectedIndex
    Dim columnandtouse As Integer = Me.cboJournalColumnAnd.SelectedIndex
    For Me.journalcounter = 0 To Me.dgvJournal.RowCount - 1
        If dgvJournal.Item(columntouse, journalcounter).Value = Me.txtJournalLookFor.Text
Then
            If Me.txtJournalLookForAnd.Text = "" Or Me.txtJournalLookForAnd.Text =
dgvJournal.Item(columnandtouse, journalcounter).Value Then
                dgvJournal.ClearSelection()
                dgvJournal.Rows(artauthcounter).Selected = True
                Me.cmdJournalFindNext.Enabled = True
                Cursor = Cursors.Arrow
                Exit Sub
            End If
        End If
    Next
    Cursor = Cursors.Arrow
    MsgBox("Can't find anything that matches the search criteria.")
End Sub

```

```

Private Sub txtJournalLookFor_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles txtJournalLookFor.TextChanged
    Me.cmdJournalFindNext.Enabled = False
End Sub

Private Sub txtArtAuthLookFor_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles txtArtAuthLookFor.TextChanged
    Me.cmdArtAuthFindNext.Enabled = False
End Sub

    Private Sub txtAuthorsLookFor_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles txtAuthorsLookFor.TextChanged
        Me.cmdAuthorsFindNext.Enabled = False
    End Sub

Private Sub txtArticlesLookFor_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles txtArticlesLookFor.TextChanged
    Me.cmdArticlesFindNext.Enabled = False
End Sub

Private Sub cboAuthorsColumnAnd_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles cboAuthorsColumnAnd.SelectedIndexChanged
    If Me.cboAuthorsColumn.SelectedIndex = Me.cboAuthorsColumnAnd.SelectedIndex Then
        Me.cboAuthorsColumnAnd.SelectedValue = ""
    End If
End Sub

Private Sub cboArticlesColumnAnd_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles cboArticlesColumnAnd.SelectedIndexChanged
    If Me.cboArticlesColumn.SelectedIndex = Me.cboArticlesColumnAnd.SelectedIndex Then
        Me.cboArticlesColumnAnd.SelectedValue = ""
    End If
End Sub

Private Sub cboArtAuthColumnAnd_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles cboArtAuthColumnAnd.SelectedIndexChanged
    If Me.cboArtAuthColumn.SelectedIndex = Me.cboArtAuthColumnAnd.SelectedIndex Then
        Me.cboArtAuthColumnAnd.SelectedValue = ""
    End If
End Sub

Private Sub cboJournalColumn_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles cboJournalColumn.SelectedIndexChanged
    If Me.cboJournalColumn.SelectedIndex = Me.cboJournalColumnAnd.SelectedIndex Then
        Me.cboJournalColumnAnd.SelectedValue = ""
    End If
End Sub

Private Sub cboAuthorsColumn_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles cboAuthorsColumn.SelectedIndexChanged
    If Me.cboAuthorsColumn.SelectedIndex = Me.cboAuthorsColumnAnd.SelectedIndex Then
        Me.cboAuthorsColumnAnd.SelectedValue = ""
    End If

```

End Sub

```
Private Sub cboArticlesColumn_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles cboArticlesColumn.SelectedIndexChanged
    If Me.cboArticlesColumn.SelectedIndex = Me.cboArticlesColumnAnd.SelectedIndex Then
        Me.cboArticlesColumn.SelectedValue = ""
    End If
End Sub
```

```
Private Sub cboArtAuthColumn_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles cboArtAuthColumn.SelectedIndexChanged
    If Me.cboArtAuthColumn.SelectedIndex = Me.cboArtAuthColumnAnd.SelectedIndex Then
        Me.cboArtAuthColumn.SelectedValue = ""
    End If
End Sub
```

```
Private Sub cboJournalColumnAnd_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles cboJournalColumnAnd.SelectedIndexChanged
    If Me.cboJournalColumnAnd.SelectedIndex = Me.cboJournalColumnAnd.SelectedIndex Then
        Me.cboJournalColumnAnd.SelectedValue = ""
    End If
End Sub
```

```
Private Sub mnuFileExit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuFileExit.Click
    Me.Close()
End Sub
```

```
Private Sub cmdAuthorAdd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdAuthorAdd.Click
    If dsAuthors.Tables("firstname").Rows(0).Item("notnull") = "no" Then
        If Me.txtFirstName.Text = "" Then
            MsgBox("A first name is required")
            Exit Sub
        End If
    End If
    If dsAuthors.Tables("lastname").Rows(0).Item("notnull") = "no" Then
        If Me.txtLastName.Text = "" Then
            MsgBox("A last name is required")
            Exit Sub
        End If
    End If
```

```
Dim dsrow As DataRow
dsrow = dsAuthors.Tables("row").NewRow
Dim counter = 0
Dim nextrow As Integer = Me.dgvAuthors.Rows.Count - 1
```

```
dsrow("authorid") = dgvAuthors.Item(0, nextrow - 1).Value + 1
dsrow("firstname") = Me.txtFirstName.Text
dsrow("lastname") = Me.txtLastName.Text
dsAuthors.Tables("row").Rows.Add(dsrow)
WriteXML("authors.xml", Me.dsAuthors)
```

```

Me.dgvAuthors.DataSource = dsAuthors
Me.dgvAuthors.DataMember = "row"
While counter < Me.IsbArticles.SelectedItems.Count
    dsrow = dsArtAuth.Tables("row").NewRow
    Dim inpk As Boolean = False
    For Each pkrow As DataRow In dsAuthors.Tables("row").Rows
        If pkrow.Item("authorid") = dgvAuthors.Item(0, nextrow).Value _
            And pkrow.Item("articleid") = Me.IsbArticles.SelectedIndices(counter) Then
                inpk = True
                Exit For
            End If
    Next
    If Not inpk Then
        dsrow("articleid") = Me.IsbArticles.SelectedIndices(counter)
        dsrow("authorid") = dgvAuthors.Item(0, nextrow).Value
        dsArtAuth.Tables("row").Rows.Add(dsrow)
    End If
    counter += 1
End While

WriteXML("artauth.xml", Me.dsArtAuth)
Me.dgvArtAuth.DataSource = dsArtAuth
Me.dgvArtAuth.DataMember = "row"
End Sub

Private Sub cmdArticleAdd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdArticleAdd.Click
    If dsArticles.Tables("articleid").Rows(0).Item("notnull") = "no" Then
        If Me.txtTitle.Text = "" Then
            MsgBox("An article title is required")
            Exit Sub
        End If
    End If
    If dsArticles.Tables("html").Rows(0).Item("notnull") = "no" Then
        If Me.txtHTMLFile.Text = "" Then
            MsgBox("A html file name is required")
            Exit Sub
        End If
    End If
    If dsArticles.Tables("pdf").Rows(0).Item("notnull") = "no" Then
        If Me.txtPDFFile.Text = "" Then
            MsgBox("An pdf file name is required")
            Exit Sub
        End If
    End If
    If dsArticles.Tables("pages").Rows(0).Item("notnull") = "no" Then
        If Me.txtHTMLFile.Text = "" Then
            MsgBox("The number of pages the article covers is required")
            Exit Sub
        End If
    End If

    Dim dsrow As DataRow

```

```

dsrow = dsArticles.Tables("row").NewRow
Dim counter = 0
Dim nextrow As Integer = Me.dgvArticles.Rows.Count - 1

dsrow("authorid") = dgvArticles.Item(0, nextrow - 1).Value + 1
dsrow("title") = Me.txtTitle.Text
dsrow("html") = Me.txtHTMLFile.Text
dsrow("pdf") = Me.txtPDFFile.Text
dsrow("pages") = Me.txtPages.Text
dsrow("journalid") = Me.cboJournal.SelectedIndex
dsArticles.Tables("row").Rows.Add(dsrow)
WriteXML("articles.xml", Me.dsArticles)
Me.dgvArticles.DataSource = dsArticles
Me.dgvArticles.DataMember = "row"
While counter < Me.lsbAuthors.SelectedItems.Count
    dsrow = dsArtAuth.Tables("row").NewRow
    Dim inpk As Boolean = False
    For Each pkrow As DataRow In dsArticles.Tables("row").Rows
        If pkrow.Item("authorid") = dgvArticles.Item(0, nextrow).Value _
            And pkrow.Item("articleid") = Me.lsbAuthors.SelectedIndices(counter) Then
            inpk = True
            Exit For
        End If
    Next
    If Not inpk Then
        dsrow("articleid") = Me.lsbAuthors.SelectedIndices(counter)
        dsrow("authorid") = dgvArticles.Item(0, nextrow).Value
        dsArtAuth.Tables("row").Rows.Add(dsrow)
    End If
    counter += 1
End While

WriteXML("artauth.xml", Me.dsArtAuth)
Me.dgvArtAuth.DataSource = dsArtAuth
Me.dgvArtAuth.DataMember = "row"
End Sub

Private Sub cmdJournalAdd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdJournalAdd.Click
    If dsJournal.Tables("articleid").Rows(0).Item("notnull") = "no" Then
        If Me.txtJournalTitle.Text = "" Then
            MsgBox("A journal title is required")
            Exit Sub
        End If
    End If
    If dsJournal.Tables("number").Rows(0).Item("notnull") = "no" Then
        If Me.txtNumber.Text = "" Then
            MsgBox("A journal number is required")
            Exit Sub
        End If
    End If
    If dsJournal.Tables("year").Rows(0).Item("notnull") = "no" Then
        If Me.txtYear.Text = "" Then

```



```

        MsgBox("A journal publication year is required")
        Exit Sub
    End If
End If
If dsJournal.Tables("issn").Rows(0).Item("notnull") = "no" Then
    If Me.txtISSN.Text = "" Then
        MsgBox("The journal ISSN is required")
        Exit Sub
    End If
End If
If dsJournal.Tables("html").Rows(0).Item("notnull") = "no" Then
    If Me.txtHTMLFolder.Text = "" Then
        MsgBox("The folder containing the journal's html files is required")
        Exit Sub
    End If
End If
If dsJournal.Tables("pdf").Rows(0).Item("notnull") = "no" Then
    If Me.txtPDFFolder.Text = "" Then
        MsgBox("The folder containing the journal's pdf files is required")
        Exit Sub
    End If
End If
If dsJournal.Tables("description").Rows(0).Item("notnull") = "no" Then
    If Me.txtDescriptions.Text = "" Then
        MsgBox("The folder containing the journal's pdf files is required")
        Exit Sub
    End If
End If

Dim dsrow As DataRow
dsrow = dsJournal.Tables("row").NewRow
Dim counter = 0
Dim nextrow As Integer = Me.dgvJournal.Rows.Count - 1

dsrow("journalid") = dgvJournal.Item(0, nextrow - 1).Value + 1
dsrow("number") = Me.txtJournalTitle.Text
dsrow("year") = Me.txtYear.Text
dsrow("issn") = Me.txtISSN.Text
dsrow("description") = Me.txtDescriptions.Text
dsrow("html") = Me.txtHTMLFolder
dsrow("pdf") = Me.txtPDFFolder
dsJournal.Tables("row").Rows.Add(dsrow)
WriteXML("journal.xml", Me.dsJournal)
Me.dgvJournal.DataSource = dsJournal
Me.dgvJournal.DataMember = "row"
End Sub
End Class

```

REFERENCES

- Bos, B. (July 11, 1997) XML Representation of a Relational Database. April 12, 2008, from <http://www.w3.org/XML/RDB.html>
- Bray, T., Maler, E., Paoli, J., Sperberg-McQueen, C. M., Yergeau, F. (2006, August 16) Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C Recommendation. Retrieved March 29, 2008, from <http://www.w3.org/TR/REC-xml/>
- Chamnerlin, D., & Saracco, C. M. (April 6, 2006) Query DB2 XML data with XQuery. April 17, 2008, from <http://www.ibm.com/developerworks/db2/library/techarticle/dm-0604saracco/>
- Clark, J., DeRose, S. (November 16, 1999) XML Path Language (XPath). April 12, 2008 from <http://www.w3.org/TR/xpath>
- Kristjánsson, M. (September 2004) Building with Oracle XML Database. April 12, 2008, from <http://www.oracle.com/technology/oramag/oracle/04-sep/o54xml.html>
- Microsoft. (2008a) DataSet.WriteXml Method. April 19, 2008 from, [http://msdn2.microsoft.com/en-us/library/system.data.dataset.writexml\(VS.71\).aspx](http://msdn2.microsoft.com/en-us/library/system.data.dataset.writexml(VS.71).aspx)
- Microsoft. (2008b) Walkthrough: Reading XML Data into a Dataset. April 19, 2008, from [http://msdn2.microsoft.com/en-us/library/ekw4dh3f\(VS.71\).aspx](http://msdn2.microsoft.com/en-us/library/ekw4dh3f(VS.71).aspx)
- Refsnes Data. (2008) XML Parser. March 15, 2008, from http://www.w3schools.com/XML/xml_parser.asp
- Sun Microsystems. (2008) XML Functions. April 17, 2008, from <http://dev.mysql.com/doc/refman/5.1/en/xml-functions.html>
- typeof [JavaScript operator]. April 5, 2008 from <http://www.adp-gmbh.ch/web/js/operators/typeof.html>
- Wikipedia. (April 6, 2008) XML database. April 12, 2008 from http://en.wikipedia.org/wiki/XML_database

XML:DB Initiative. (May 7, 2003) XML:DB Initiative for XML Databases. Retrieved April 17, 2008, from <http://xml-db-org.sourceforge.net/>