Knowledge Management for the Demilitarization of Munition using Advanced XML Algorithms

A THESIS
SUBMITTED TO THE GRADUATE FACULTY
in partial fulfillment of the requirements for the degree of

MASTERS OF SCIENCE
MECHANICAL ENGINEERING

By

James Thomas Lee
Norman, Oklahoma
2006
Knowledge Management for the Demilitarization of Munition using Advanced XML Algorithms

A THESIS APPROVED FOR THE
SCHOOL OF AEROSPACE AND MECHANICAL ENGINEERING

BY

Chair: ______________________________  Dr. Kurt Gramoll

Member: ____________________________  Dr. J. David Baldwin

Member: ____________________________  Dr. Zahed Siddique
Acknowledgements

The author would like to thank Dr. Gramoll for his support and encouragement throughout this project. His knowledge in this area of research propelled the author to new levels in this ever growing topic. The author would also like to thank Siamak Faridani and Chris DiLorenzo for their help and guidance. Their expertise in the field of web-based applications helped the author reach project milestones. This project was supported by the Defense Ammunition Center and the author would like to appreciate their continued support of this project at The University of Oklahoma.

The author would not be where he is today if it wasn’t for the amazing faculty in the School Aerospace and Mechanical Engineering at The University of Oklahoma. The author thanks them immensely.
# Table of Contents

ACKNOWLEDGEMENTS ........................................................................................................ IV

LIST OF EXAMPLES ........................................................................................................... VIII

LIST OF FIGURES .............................................................................................................. X

LIST OF TABLES ................................................................................................................ XII

ABSTRACT .......................................................................................................................... XIII

CHAPTER 1: INTRODUCTION .............................................................................................. 1

1.1 DEFENSE AMMUNITION CENTER ........................................................................ 1

1.2 APPLICATION DEVELOPMENT ........................................................................... 2

1.3 PROBLEMS ENCOUNTERED .............................................................................. 8

1.4 METHODS USED ............................................................................................... 11

1.5 HIERARCHICAL XML ALGORITHM ..................................................................... 12

1.6 SIMILAR APPLICATIONS ................................................................................ 13

1.7 OVERVIEW ...................................................................................................... 14

CHAPTER 2: BACKGROUND ............................................................................................ 16

2.1 OVERVIEW OF THE PROJECT ........................................................................ 16

2.2 OVERVIEW OF THE TECHNOLOGIES USED .................................................... 16

2.2.1 Web-Based Applications ................................................................... 16

2.2.2 Macromedia Flash ................................................................................... 18
List of Examples

EXAMPLE 2.2.4.1: ODBC CONNECTION AND SQL STATEMENT ................................. 24
EXAMPLE 2.2.4.2: ACTIVEX DATA OBJECT CONNECTION .............................. 24
EXAMPLE 2.2.4.3: DIRECT CONNECTION STRING ................................................ 25
EXAMPLE 2.2.5.1: BASIC HTML FILE .................................................................. 29
EXAMPLE 2.2.5.2: BASIC XML FILE ..................................................................... 30
EXAMPLE 2.2.5.3: FLAT XML FILE ....................................................................... 31
EXAMPLE 2.2.5.4: HIERARCHICAL XML ............................................................... 31
EXAMPLE 2.2.6.1: HELLO WORLD ASP.NET ......................................................... 33
EXAMPLE 3.3.1: ASP THAT PLACES THE DATABASE DATA INTO A STRING AND

RETURNED TO BROWSER ................................................................................... 42
EXAMPLE 3.3.2: ACTIONSCRIPT FUNCTION TO PROCESS XML FROM ASP CODE .. 44
EXAMPLE 3.4.1: ASP.NET CODE THAT CREATES XML ......................................... 52
EXAMPLE 3.4.2: ASP.NET THAT CREATES XML USING THE DATASET METHOD... 53
EXAMPLE 3.4.1.1: FLASH ACTIONSCRIPT WITH IMPROVED TEXT VARIABLE .... 58
EXAMPLE 3.5.1: FACTORIAL RECURSIVE FUNCTION ........................................... 63
EXAMPLE 3.5.2: ASP.NET CODE USING THE XmlTextWriter OBJECT ............... 66
EXAMPLE 3.5.3: OUTPUT XML FILE FROM THE ASP.NET USING THE

XmlTextWriter ................................................................................................. 67
EXAMPLE 3.5.3.1: SQL STORED PROCEDURE TO CREATE XML ......................... 70
EXAMPLE 3.5.3.2: SQL STORED PROCEDURE THAT USES A TEMPORARY TABLE .... 72
EXAMPLE 3.5.3.3: SQL EXECUTION STATEMENT WITH TEMPORARY TABLE .......... 72
EXAMPLE 3.5.3.4: SQL STORED PROCEDURE THAT USES ATTRIBUTES .............. 74
EXAMPLE 3.5.3.5: SQL SCALAR VALUED, USER-DEFINED FUNCTION ............... 76
EXAMPLE 3.5.3.6: SQL STORED PROCEDURE THAT CALLS THE SCALAR VALUED,
USER-DEFINED FUNCTION ........................................................................... 77
EXAMPLE 3.5.3.7: DYNAMIC STORED PROCEDURE ........................................ 79
# List of Figures

**FIGURE 1.2.1:** INITIAL VERSION OF THE TECHNOLOGY TREE APPLICATION .......... 3

**FIGURE 1.2.2:** NODE ORGANIZATION IN INITIAL TECHNOLOGY TREE APPLICATION. 4

**FIGURE 1.2.3:** CURRENT TECHNOLOGY TREE APPLICATION SCREENSHOT .............. 6

**FIGURE 1.2.4:** COLUMNS WITHIN EVERY TABLE IN THE DATABASE ....................... 7

**FIGURE 1.3.1:** TECHNOLOGY TREE THAT SHOWS THE CONFUSING LINE BETWEEN

SISTER ELEMENTS .................................................................................................................. 10

**FIGURE 1.3.2:** INFORMATION BUTTON ........................................................................... 11

**FIGURE 2.2.1.1:** WEB-BASED APPLICATION OPERATION STEPS .............................. 18

**FIGURE 2.2.2.1:** FLASH FLOWER SWF ........................................................................ 19

**FIGURE 2.2.3.2:** PART OF A TABLE WITHIN A DATABASE ........................................... 22

**FIGURE 2.2.4.1:** OPEN DATABASE CONNECTIVITY .................................................. 23

**FIGURE 2.2.5.1:** TREE REPRESENTATION OF THE XML FILE SEEN IN EXAMPLE

2.2.5.4 .................................................................................................................................... 32

**FIGURE 2.2.6.2:** HELLO WORLD SOURCE CODE ....................................................... 34

**FIGURE 2.2.6.1.1:** COMMAND PROMPT FOR INSTALLING THE .NET FRAMEWORK. 35

**FIGURE 3.1.1:** APPLICATION OPERATION STEPS I .................................................. 38

**FIGURE 3.3.1:** FLASH TESTING APPLICATION SCREENSHOT .................................. 45

**FIGURE 3.3.2:** MACROMEDIA FLASH ERROR MESSAGE .......................................... 47

**FIGURE 3.3.3:** TESTING APPLICATION TABLE LOADING TIMES ............................ 49
FIGURE 3.4.1: MODIFIED FLASH TESTING APPLICATION SCREENSHOT ................. 54
FIGURE 3.4.2: ASP AND ASP.NET AVERAGE TIME MEASUREMENTS ................. 56
FIGURE 3.4.1.1: TIME MEASUREMENTS WITH THE IMPROVED VARIABLE IN FLASH 61
FIGURE 3.6.1.1: FLASH VERSION TEST ................................................................. 82
FIGURE 5.4.1: APPLICATION OPERATION STEPS II ........................................... 91
List of Tables

TABLE 2.2.3.1: SQL DIALECTS ............................................................................. 21
TABLE 3.1.1: APPLICATION PROCESS I ................................................................. 37
TABLE 3.1.2: BROWSER AND SERVER SPEED COMPARISONS (SECONDS) .......... 39
TABLE 3.3.1: TESTING TABLE CHARACTERISTICS .............................................. 46
TABLE 3.3.2: TIME TRIALS FOR THE ASP THAT CREATES FLAT XML (SECONDS) .. 47
TABLE 3.4.1: AVERAGE TIME MEASUREMENTS I (SECONDS) .............................. 55
TABLE 3.4.3: TIMES AFTER EACH RECORD I ......................................................... 57
TABLE 3.4.1.1: TIMES AFTER EACH RECORD II .................................................... 59
TABLE 3.4.1.2: AVERAGE TIME MEASUREMENTS II (SECONDS) ......................... 60
TABLE 3.4.1.1: DATASET COMPARISONS ............................................................ 62
TABLE 3.5.3.1: SQL STORED PROCEDURE EXECUTION TIME USING A RECURSIVE
FUNCTION ............................................................................................................. 75
TABLE 3.5.3.2: STORED PROCEDURE EXECUTION TIME USING A RECURSIVE
FUNCTION ............................................................................................................. 77
TABLE 3.6.1.1: MOVIE CLIP CREATION IN DIFFERENT FLASH VERSIONS (SEC) .... 81
TABLE 5.4.1: APPLICATION OPERATION STEPS II .............................................. 90
Abstract

The Defense Ammunition Center’s web-based application that visually displays the steps and procedures to demilitarize ammunition in a tree structure is slow and costs its employees valuable time. In this research, new methods, technologies, and software were develop and implemented to improve the loading time as well as the graphical user interface of the application. The implementation of ASP.NET, SQL Server 2005, and XML have led to numerous new capabilities for improving the performance of the application. The technologies and methods developed and implemented are unique and improve the efficiency of retrieving and organizing hierarchical data. The most notable advancement within this study is the use of an ASP.NET page that accesses a stored procedure within SQL Server 2005 that calls a recursive function that creates hierarchical XML and then sends it to Macromedia Flash. The recursive function finds all of the child/parent relationships within the data. The previous algorithm for the application used over 200 lines of ActionScript code to find the parent/child relationships within Macromedia Flash. The current method is a stored procedure with a recursive function that is twenty-three lines long. Not only was a new algorithm developed to find the parent/child relationships, but the application’s technologies were upgraded and this caused the application to be over 250 times faster.
The recursive function developed in this research has the potential to enhance the efficiency of sending other structured data through the Internet. The general public will see the capabilities of this new algorithm with the upcoming release of Microsoft Office 2007, where all of the Word, Excel, and PowerPoint files will be XML files. The release of the new Office will cause more applications to use Extensible Markup Language (XML) to store data. XML files will become a common format to store data and as more software programs implement XML the need for XML tools will only increase.
Chapter 1: Introduction

1.1 Defense Ammunition Center

The Defense Ammunition Center (DAC) in McAlester, Oklahoma, has a project called the “Demilitarization Knowledge Management Application for Transitioning the Ammunition Stockpile.” The value of this project is its life saving potential through the timely distinction of old and unstable munitions. The purpose of this project is to develop an application that will visually display the steps and procedures to demilitarize ammunition. This involves detailed steps to ensure a safe demilitarization operation. This application will primarily be used by the Defense Ammunition Center and the United States Army, but will also be released in the future to other institutions across the world.

The development of this application, known as Technology Trees, a web-based knowledge management of ammunition demilitarization, has taken place at the University of Oklahoma. Over the past three years numerous improvements have been made to the application. Previous designs have been shown to the management of the Defense Ammunition Center and also to upper level army officials. The potential of this application and the goals it will achieve for the Defense Ammunition Center have been acknowledged by upper management levels in the demilitarization community.
The Defense Ammunition Center (DAC) is part of the U.S. Army Joint Munitions Command (JMC). The mission of the DAC is to, “Provide the military services timely ammunition training, demilitarization technology, explosives safety, engineering, career management, and technical assistance through logistics support.” [1] The DAC manages an ongoing program called Munitions Items Disposition Action System (MIDAS) and receives programmatic support from the Argonne National Laboratory. MIDAS consists of a database that provides a central source of the most accurate information on the structure and composition of conventional munitions [2].

1.2 Application Development

The Technology Tree application has seen several versions of development over the last three years. One of the first versions of development had an active server page that accessed a database and returned this data back to the active server page. Flash then displayed the data in a graphical tree structure. The database used was MySQL, an open source database. Once the data was obtained by the active server page, all of the data was placed into a string. This string was then passed to Flash where it was processed by several ActionScript, a built-in programming language in Flash, loops to find the parent/child relationships of the data. Then Flash application then produced a visual representation of the data in a tree structured. A screen capture of one of the
initial applications is shown in Figure 1.2.1. The node shown in Figure 1.2.1 is the parent node and when it is clicked on, the children of that node will appear below the parent node. Then the children nodes can be clicked on to view their children and so on.

![Figure 1.2.1: Initial version of the Technology Tree application](image)

Figure 1.2.1: Initial version of the Technology Tree application

Figure 1.2.2 shows what was displayed when the “All” button was clicked inside the info box in the upper right hand corner. This version was a good start, but there were many problems with the tree structure. One of the major problems
was that there were lines within the tree that were connected that should not have been connected. The tree had a line that disappeared off the page and left the user to follow that line until they reached the sister node. The information within the nodes did not fit within the space provided. These were just of few of the problems with the structure of the early version.

*Figure 1.2.2: Node organization in initial Technology Tree application*
Later versions of the Technology Tree application used new methods to achieve a better graphic output of the Technology Trees [3]. The revised version had visually pleasing colors and more controls to edit and add parts to the tree structure. Figure 1.2.3 shows a screenshot of the revised Technology Tree application. The visual representation of a simple example tree structure is shown in the screenshot. The tree structure shown in Figure 1.2.3 was created solely for testing purposes and represents a typical table of information. Testing designs of the application was an important task, and this simple tree allowed for easy testing parameters, considering each node is labeled by its position (row11, row21, etc…), but actual tree structures can have over a thousand nodes.
The current Technology Tree application shown in Figure 1.2.3 uses MySQL 4, which is a structured query language database management system. MySQL 4 is free database software under the GNU General Public License. MySQL 4 has been used over the last three years and has proved to be an efficient tool and beneficial to the Technology Tree application.

The management of several databases in MySQL 4 often requires an experienced database administrator. MSQQL 4 uses queries to complete administrative tasks, like adding a user and giving permissions. The use of a
graphical administration tool allows the administrator to click a button to add a user instead of having to write and execute a query, but behind the scenes the query is actually being executed in the graphical administration tool. This allows for easy management of several databases. To help manage the several MySQL databases, EMS MySQL Manager 3 was used. This program is a graphical administrative tool, for MySQL 4, that allows most of the administrative tasks to be completed.

![Table - dbo.aa](image)

*Figure 1.2.4: Columns within every table in the database*

The organization of data that comes from a database is a task that is tedious and time consuming. During this age of technological advancements programs are becoming complex and are interacting dynamically with the customers on the Internet. The Defense Ammunition Center’s project is a good example of a web-based dynamic application that sends data to the customer. The
best way to send structured data is by converting the data to Extensible Markup Language (XML). The current Technology Tree application does not implement XML and has a scripting language (ASP) that does not have adequate XML capabilities. A new hierarchical XML algorithm has been created as a simple way to dynamically send structured data to the customer via the Internet. The new algorithm consists of a stored procedure and a recursive function in SQL Server 2005 that will find the parent child/relationships. This new algorithm allows the Flash ActionScript code to be reduced by nearly 200 lines and can be used worldwide to develop fast and short applications with little programming experience. The newly developed algorithm has the potential to be a standard way of sending structured data on the web.

The advantages of storing structured data is also understood by other organizations, like Microsoft. They have decided to implement XML into its new Office 2007. “Microsoft Office will use XML as the default file format. According to Microsoft, this file format, which is compressed, will be up to 75% smaller than the current file formats [4].”

1.3 Problems Encountered

This research started with analyzing the problems encountered within the current Technology Tree application. The current Technology Tree application had already undergone three years of work, but times had changed and new
technologies are available. After analyzing the application the following problems were identified.

1. The algorithm within Flash is able to obtain the data from the database starting with the parent and then proceeds to obtain data from the children nodes. This process causes the user to wait while all of the data is being retrieved. The data needs to be reorganized in a structured manner that allows the data to know the parent/child relationships. A new algorithm needs to be developed to improve the loading time of the application.

2. Active Server Pages (ASP), also known as classic ASP, was implemented into the application two years ago. ASP is slower than newer languages and does not support advanced XML data organization.

3. The current Technology Tree application is difficult to navigate to sibling nodes in large tree structures. Figure 1.3.1 shows the problem with lines that connect sibling nodes. These lines that go off of the screen, could keep going across several screens. The line between the Electronics node and the Battery node in Figure 1.3.1 is acceptable because they both can be seen on the same screen, but the distance between the Guidance and Control node and its sibling node cannot be seen on one screen which is not acceptable. If the sibling element is
over a screen distance away, then the user must drag the screen over until the sibling node is reached. The navigation problem between sibling nodes in large tree structures is unacceptable.

4. All data obtained from the database is sent to Flash in the current Technology Tree application, but not all data is actually used. The notes column is the largest record size in the database, but the notes are only used if the info button on the respective node is clicked on. Figure 1.3.2 shows where the info button is located on each node. Sending data that is not being used is causing the Technology Tree application to be slower than it could be.
This research will explain how these problems were solved and what other problems arose when solving these. This work will also explain why there is a need for web-based applications in the engineering industry along with examples.

In summary, the most important issue with the application is the speed. This research will focus on the speed issues and will explain in detail the speed improvements that were made.

1.4 Methods Used

There are several different ways to find parent/child relationships in data. One way is to have many loops with if statements, but this would only complicate the code and would probably cause an increase in loading time instead of a decrease. Another way to find the parent/child relationships in the data is to use a recursive function. A recursive function is a function that is defined by itself, and calls itself in the function and continues to do so until a condition is met. Using a recursive function will find the parent/child relationships in the data and save
Flash from having to find these relationships. The use of the recursive function will decrease the loading time because the data will already be structured when Flash retrieves the data.

A recursive function and a stored procedure are accessed by an ASP.NET file. The stored procedure then calls the recursive function. Both the stored procedure and the recursive function use a SELECT statement to create hierarchical XML that is then sent to Flash. Flash then reads the structured data and displays the tree structure. This next generation, database driven, web-based application made major improvements over the previous version. SQL Server, ASP.NET, and XML are being used to their full abilities to achieve the success that this application has had. The use of these new features has led to improvements that are over 250 times faster and still have the capabilities to be even faster.

1.5 Hierarchical XML Algorithm

The research done for the Defense Ammunition Center’s web-based application led to a new algorithm to create hierarchical XML. Hierarchical XML will be used in future software programs such as Word Processing, CAD, DBMSs, and many more. As more software companies are starting to use XML as the default file format, there will be a need to convert all old documents to
XML and then be saved in a database. This has already started to take place with the development of native XML databases [5].

Native XML databases are being developed to store XML files and have the capabilities to edit the XML file structure. One of the benefits for using a native XML database is that the XML data does not have to be mapped to another data structure. This allows XML to be inserted into the database and XML to be retrieved.

1.6 Similar Applications

Companies are using Flash to develop interactive web pages to advertise their products, including Sony, Microsoft, Toyota, and BMW [6]. Flash Earth uses satellite photos as a way to show maps of the United States and does an effective job of having controls to zoom in and out which are similar to the tools available in the Technology Tree application [7]. Pandora.com is a flash application that allows users to search vocal artists or songs and then listen to similar types of music. Not only are Flash database driven applications being developed but databases are becoming more and more popular in companies. These companies are using the databases to store electronic data about their customers or for many other business purposes. Companies need and will invest in a web-based application to access the companies database so their employees may store their information with ease no matter where they are in the world. In
the past, database management systems (DBMSs) were expensive and few people knew how to work with them, but now new, open source, database management systems are available and are causing SQL Server, Oracle, DB2, and other DBMSs to improve their programs to keep their customers from converting to the open source DBMSs.

Peoples’ views of databases will change in the next year with the release of Windows Vista. Windows Vista uses a database similar to SQL Server to organize and store files of the operating system. Databases are continuing to grow and develop. More and more databases are storing files and as time goes on files will get larger. Files may be stored in databases using different methods. The actual binary data from the file may be stored in the database or the links to the files on the server may be stored in the database. Both are effective ways to access files using a database.

1.7 Overview

The Defense Ammunition Center and The University of Oklahoma have developed an application that has the capability to display data in a structured manner. The application initially had several loops within Flash that created a slow web loading time that frustrated the employees of the Defense Ammunition Center. Research has been done to prevent extra loops and coding within Flash. Although officials who saw the application were impressed, The University of
Oklahoma and the Defense Ammunition Center knew the application could be better. The following chapters will explain in detail how the Technology Tree algorithm was improved and how it has the potential when used in web-based applications to increase the efficiency in transferring structured data over the Internet.
Chapter 2: Background

2.1 Overview of the Project

The initial research on demilitarization and knowledge management started at The University of Oklahoma in 2002. The work included developing a web-based application that visually shows how to demilitarize ammunition using a tree structure. The web-based application, known as Technology Trees, used several technologies including Active Server Pages (ASP), Macromedia Flash MX, and MySQL.

The goal of the current research is to find the best method to bring in structured data from a database into Flash. The use of structured data will prevent Flash from having to find the parent/child relationships. In order to fully understand the technologies used in this research the basic background information will be presented for Macromedia Flash, Structured Query Language (SQL), MySQL, SQL Server, XML, and ASP.NET.

2.2 Overview of the Technologies Used

2.2.1 Web-Based Applications

Web-based applications are applications that may be accessed through the Internet by using a web browser (Microsoft Internet Explorer, Mozilla Firefox,
etc...). On the other hand, stand alone applications are distributed to the user and are installed directly on the client’s computer. Depending on the number of applications this could be a time consuming task. If a change is made to the application, an update must be distributed to all computers with the application. However, web-based applications do not have to be installed on the client’s computer. The client simply accesses the application through the Internet. If a change needs to be made, the administrator of the application will simply make the change and then upload the new version of the application to the server.

Web-based applications can typically be broken down into three different parts. The first part of the application is the web browser. The web browser is how the user accesses the web-based application. The second part is a dynamic scripting language (ASP.NET, PHP, CGI, etc…) that is executed on the server. The third part is a database (Oracle, Microsoft SQL Server, IBM DB2, etc…) that contains the data needed in the application. Figure 2.2.1.1 shows the operation steps that typically take place in a web-based application.
2.2.2 Macromedia Flash

Macromedia Flash is an advanced authoring environment for creating interactive websites, digital experiences, and mobile content [8]. In 1996 FutureWave Software was purchased by Macromedia. Macromedia then turned FutureWave’s vector-based animation program, FutureSplash Animator, into Macromedia Flash 1.0 [9]. Since then, Macromedia has developed 8 different versions of the Flash technology. The most recent version, Flash 8, was released on September 13, 2005. One of the best characteristics of Flash is its built-in scripting language called Actionscript. Actionscript 1.0 was implemented in Flash 5 and then Flash 7 introduced Actionscript 2.0. Actionscript can be used to create many different kinds of interactive web based applications, including an
MP3 player, a multi-user drawing application, a 3D walkthrough of a home, an online store, a message board, an HTML editor, and even interactive games [10].

Two of the main file formats used in Macromedia Flash are the .fla and the .swf. The Flash .fla file contains the source material as well as the ActionScript code. The Flash .swf file is the compiled and published file that may be viewed on the web or viewed in the Flash Player. The .swf file can be seen on the Internet by importing it into an html file. The Figure below shows a screen capture of a simple animation .swf file in the Flash Player.

![Macromedia Flash Player](image)

*Figure 2.2.2.1: Flash flower swf*

The development of the Technology Tree application was influenced by a Flash animation of a dynamically drawn flower (shown in Figure 2.2.2.1) [11]. The methods used in the flower animation consisted of creating a movie clip for each petal of the flower and then rotating it based on an equation. This idea of
creating movie clips and moving them to certain locations is known as object
oriented programming. One of the main goals of this research was to develop a
tree structure that is dynamically drawn in Flash. The use of object oriented
programming in Flash was essential for creating this tree structure for the
Technology Tree application.

2.2.3 Relational Databases and Structured Query Language

A database is a collection of data that is organized by related information. The software that is used to store these databases is known as a Database
Management System (DBMS). One of the most common types of DBMSs is the
Relational Database Management System also known as the RDBMS. Some of
the main RDBMSs are Microsoft Access, Microsoft SQL Server, DB2, Sybase
Adaptive Server, Oracle, MySQL, and PostgreSQL. A RDBMS uses a relational
model to represent data in sets of tables. The relational model was first developed
by Edgar F. Codd in 1970 [12]. IBM then developed a Structured English Query
Language (SEQUEL) in the 1970s that was used to manipulate and retrieve data
from one of their databases called System R. SEQUEL eventually became known
as Structured Query Language (SQL) [13].

Databases only store information and the users of the database must know
how to communicate with the database in order to access this information. The
most common way to communicate with databases is to use Structured Query
Language (SQL). “SQL is the language for generating, manipulating, and retrieving data from relational databases [14].” Although there is an SQL standard recognized by the American National Standards Institute (ANSI) and by the International Organization for Standardization (ISO) companies of RDBMSs have their own dialect. “These dialects most commonly have evolved because the user community of the given database vendor required capabilities in the database before the ANSI committee created a standard [15].” Table 2.2.3.1 shows the different types of SQL dialects.

<table>
<thead>
<tr>
<th>Dialect</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/SQL</td>
<td>Found in Oracle. PL/SQL stands for Procedural Language/SQL and contains many similarities to the language Ada.</td>
</tr>
<tr>
<td>Transact-SQL</td>
<td>Used by both Microsoft SQL Server and Sybase Adaptive Server. As Microsoft and Sybase have moved away from the common platform they shared early in the 1990s, their implementation of Transact-SQL have also diverged.</td>
</tr>
<tr>
<td>PL/pgSQL</td>
<td>The name of the SQL dialect and extensions implemented in PostgreSQL. The acronym stands for Procedural Language/postgreSQL.</td>
</tr>
<tr>
<td>SQLPL</td>
<td>The newest dialect is DB2’s SQLPL (SQL Procedural Language), which is based on the standard SQL control statements. Since SQLPL came after the standard, it is in greater compliance.</td>
</tr>
</tbody>
</table>

The tables within Relational Database Management Systems are constructed from rows and columns of data. Databases have the capabilities of having many tables and each table having different columns of data. Figure
2.2.3.2 shows part of a table within a database that has three columns of information.

```
<table>
<thead>
<tr>
<th>id</th>
<th>Node</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>President</td>
<td>NEIL</td>
</tr>
<tr>
<td>2</td>
<td>Vice President</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>CEO</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>CTO</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Group Project Manager</td>
<td>4</td>
</tr>
</tbody>
</table>
```

*Figure 2.2.3.2: Part of a Table within a Database*

### 2.2.4 Database Connections and SQL Statements

Early stages of the Technology Tree application connected to a database through an application programming interface known as the Open Database Connectivity (ODBC). The ODBC may be reached on a Windows XP computer by going to Start/Control Panel/Administrative Tools/Data Sources (ODBC). By clicking on the System DSN tab the window will show all of the data sources and their respective driver that will be used to access the database. Data sources may be added, removed, or configured in this window.
Once the connection with the database is set through the ODBC, there must be a structure query language (SQL) statement to access a table within the database. One of the most important SQL statements is the SELECT statement. The SELECT clause is the first clause in a SELECT statement and it determines which columns to include in the query’s result set [14]. Other possible clauses in the SELECT statement are FROM, WHERE, GROUP BY, HAVING, and ORDER BY. The data that was obtained using the SELECT statement is put into
a string where it is then sent to Macromedia Flash. Macromedia Flash then uses a function that is about 200 lines of code that searches the data for the correct node information. The following ASP.NET example code accesses the ODBC and then connects to the database called “tt_missile” and a string is set equal to an SQL statement to read the data from a table called “aa” within the tt_missile database.

```
Dim conNodes As OdbcConnection = New OdbcConnection("DSN=tt_missile")
Dim SQLinput As String = "SELECT * FROM aa"
```

Example 2.2.4.1: ODBC Connection and SQL Statement

Other Microsoft APIs are ActiveX Data Object (ADO), Object Linking and Embedding for Databases (OLEDB), and ADO.NET. An ADO connection in ASP is shown in Example 2.2.4.2.

```
Dim Connect
Set Connect = Server.CreateObject("ADODB.Connection")
```

Example 2.2.4.2: ActiveX Data Object Connection

Another way of accessing a database is through a direct connection. A direct connection does not go through the ODBC. All of the information needed by the database including the server name, user id, password, and the database
name are all included in the ASP.NET connection string as shown in Example 2.2.4.3.

```vbs
Dim MyConString As String = "Server=JAMES\SQLEXPRESS; uid=root; pwd=trial; database=Northwind"
```

Example 2.2.4.3: Direct Connection String

One of the advantages for the direct connection string is that it does not have to go through an API to connect to the database. It may directly connect to the database. The direct connection method was implemented into the Technology Tree application because it was easier to use and a secure method of connecting to a database.

**2.2.5 MySQL**

Databases are commonly used for businesses, education, research, and web-based applications. Organizations understand the importance of accessing information resources from one centralized location. This characteristic was utilized with the creation of databases, but small organizations were unable to purchase and support databases because of their large cost. This has changed with the creation of inexpensive operating systems such as BSD UNIX derivatives and various forms of Linux. Large efforts have been made to push for software that is available for anyone free of cost and this has led to open source software. There
have been several successful open source software products that have been important in the movement towards using open source software. The main ones include the Apache web-server, the PHP scripting language, and the PostgreSQL database management system. Another open source database management system is MySQL [16].

MySQL is a relational database management system (RDBMS) and it is free under most circumstances. MySQL is known for is its speed and is fast for simple and complex SELECT statements. The Technology Tree application must be fast and MySQL initially met the requirements the DAC needed. The use of MySQL is user friendly, especially with the use of the visual administration tool called EMS MySQL Manager. The visual administration tool allowed several databases to be managed and maintained easily.

Versions of MySQL prior to the release of MySQL 5.0 were criticized for not having stored procedures, triggers, or views. The release of MySQL 5.0 on October 24, 2005, reduced the criticism. The use of stored procedures has improved the capabilities within MySQL. InnoDB and Berkeley DB, were two database engines used by MySQL that were acquired by Oracle Corporation, who develops database management systems. InnoDB was acquired in October 2005 and Berkeley was acquired in January 2006. Oracle could implement these engines in a new database management system similar to MySQL.
2.2.6 SQL Server

Another database that is implemented into the Technology Tree application is SQL Server. SQL Server is a RDBMS made for Microsoft Windows operating system. Several new versions of SQL Server 2005 were released on November 7, 2005. One of these versions was SQL Server 2005 Express Edition which is a free, lightweight version of SQL Server 2005 [17]. The Express Edition is able to support only one Central Processing Unit and is only able to address 1 gigabyte of RAM. The maximum database size is 4 gigabytes. These limits are acceptable for testing the Technology Tree application. The other versions of SQL Server 2005 are the Enterprise Editions, Standard Editions, and the Workgroup Editions. The retail price for these versions vary from $739 to $24,999. The Express Edition may be downloaded from http://www.microsoft.com/downloads/. The Express Edition was capable of handling the tables that were used for testing.

There is a Microsoft graphical administration tool available for SQL Server 2005. The administration tool is called SQL Server Management Studio. The tool may be used to complete most of the administrative tasks [18]. Another SQL Server graphical administrative tool is EMS SQL Manager 2005 for SQL Server 2005. Both of these administrative tools may be used to manage servers, databases, security, and more.
Security is an issue that the Defense Ammunition Center takes seriously. Any type of software that must be installed on one of the Defense Ammunition Servers must be proven to be a secure. The process of getting software approved is a tedious task and a slow process. This is one of the main reasons why the DAC wants to avoid open source software like MySQL, which was used with the initial versions of the Technology Tree application. Since MySQL will cause an extended period of time to be approved, a new DBMS was tested and implemented into the Technology Tree application.

In December 2005, the DAC databases were tested in SQL Server 2005 and SQL Server proved to be a faster DBMS for the application. Another benefit of implementing SQL Server 2005 is that it has capability of using stored procedures, unlike MySQL 4, which allow for more complicated SQL statements. Also, SQL Server 2005 was integrated with Visual Studio 2005 to create web-based applications that work directly with databases. The integration of SQL Server 2005 and Visual Studio 2005 has made them one of the most capable in the industry.

2.2.7 Extensible Markup Language (XML)

Previously, the Technology Tree application used one long text string to move data from the database to Flash, but this was a slow method. New methods were researched to find faster ways of sending data through the Internet. One of
the methods identified was XML. XML is a descendant of the Standard Generalized Markup Language (SGML). “The language that would eventually become SGML was invented by Charles F. Goldfarb, Ed Mosher, and Ray Lorie at IBM in the 1970s and developed by several hundred people around the world until its eventual adoption as ISO standard 8879 in 1986 [19].”

XML is a good way to send data through the web because it is readable by humans as well as computers. It also has the ability to represent almost any kind of data structure. XML uses tags to structure the data. The tags are used to describe the data. Unlike XML, HTML has fixed tags. Example 2.2.5.1 shows a basic HTML code that will print “Hello World!” to the screen. The tags shown in the HTML code include HTML, HEAD, TITLE, and BODY. These tags are on every HTML page and these tags tell the web browser what to do.

```
<html>
<head>
<title>Hello World!</title>
</head>
<body>
Hello World!
</body>
</html>
```

Example 2.2.5.1: Basic HTML File

Similar to HTML, XML uses tags to describe and structure the data. Two of the main parts of an XML document are the elements and the attributes. A
defined piece in an XML document that is denoted by a start and end tag is called an element. An attribute defines a specific setting or provides additional information to the element [20]. Both elements and attributes are shown in Example 2.2.5.2. In Example 2.2.5.2 the elements are labeled “element” and the attributes are labeled “attribute”. The first element in the example is also called the root element because it is the top level element that contains all other elements. Within each element in Example 2.2.5.2 there is an attribute. Data is stored in the attributes and in the elements.

Example 2.2.5.2: Basic XML File

<?xml version="1.0" encoding="UTF-8"?><element attribute="data 1"><element attribute="data 2">data 3</element></element>

XML files can have two different types of formats. They can either be flat or hierarchical. Flat refers to the fact that the XML does not have a structure or a hierarchy. A flat XML file is shown in Example 2.2.5.3. It is flat because there is only one level of data in the root element.
Another kind of XML is called hierarchical XML. Hierarchical data has sub-data or children as shown in Example 2.2.5.4. Notice how there is more than one level of data inside the root element.

Hierarchical XML can be thought of as a tree structure. If the data from Example 2.2.5.4 was put into a tree structure it would look like Figure 2.2.5.1.
The structure of a hierarchical XML file is similar to the Technology Tree data. Thus, it would be ideal if this XML structure could be imported directly into Flash. If hierarchical XML can be created, the Technology Tree application would not need to use several loops in the Flash ActionScript to structure the data. The data would already be structured in a tree format. If the loops do not need to be executed in Flash to find the tree structure, then valuable time will be saved in the application.

2.2.8 ASP.NET

The Technology Tree application currently uses ASP, which does not have many XML capabilities. ASP.NET is the next version of Microsoft’s server-side scripting language and has many XML capabilities that will be beneficial to the Technology Tree application.
ASP.NET is a server-side scripting language that was developed by Microsoft to create dynamic HTML pages. ASP.NET was first released in January 2002. The previous Microsoft release of a server-side scripting language was in November 2000 when ASP 3.0 was released. An ASP.NET file has a .aspx file extension. Example 2.2.6.1 shows a basic ASP.NET file that simply outputs “Hello World!” to the browser.

```
<%@ Page Language="VB" %>
<html>
<head>
<title>ASP.NET Page</title>
<script runat="server">
    Sub Page_Load()
        Message.Text = "Hello World!"
    End Sub
</script>
</head>
<body>
<asp:Label ID="Message" runat="server"/>
</body>
</html>
```

Example 2.2.6.1: Hello World ASP.NET

The key to understanding how ASP.NET Web Applications work is to understand that the code in a `<script runat="server">` block (or a `<% %>` render block) is executed on the server [21]. When an ASP.NET page is called in a browser the page is sent to the browser where the ASP.NET code will be stripped out and only HTML will remain. The remaining HTML may be seen in the page.
source. The Hello World ASP.NET page will have the source code shown in Figure 2.2.6.2.

![Hello World Source Code](image)

*Figure 2.2.6.2: Hello World Source Code*

### 2.2.8.1 Installation

Before an ASP.NET script can be run on the server the appropriate software must be installed. Currently the operating systems Windows 2000, Window Server 2003, or Windows XP can be used with ASP.NET. The web server, Internet Information Services (IIS), must be activated from the add/remove Windows components. Finally the .NET Framework needs to be installed. Before installing the .NET Framework, install the latest updates for the operating system and also for Internet Explorer. The .NET Framework may be downloaded for free from Microsoft. Once the .NET Framework is installed an ASP.NET page will run correctly. A problem will occur if two versions of the .NET Framework are
installed on the same machine. This problem occurred while using IIS 5.1 and .NET Framework 1.1. To solve the problem the “aspnet_regiis –i” script must be executed from the .NET Framework folder. This same problem occurred when the .NET Framework 2.0 was installed, and again the “aspnet_regiis –i” script must be executed to fix the problem. The “aspnet_regiis –i” script was executed in the command prompt. Figure 2.2.6.1.1 shows what will happen when the script is executed.

![Command Prompt for Installing the .NET Framework](image)

**Figure 2.2.6.1.1: Command Prompt for Installing the .NET Framework**

### 2.2.8.2 .NET Framework

The .NET Framework provides a highly productive environment for creating next-generation applications and services, as well as the agility to solve the challenges of deployment and operation of enterprise-scale applications [22]. The .NET Framework provides these capabilities by utilizing the Common Language Runtime (CLR) and the Framework Class Library (FCL). “The CLR manages the execution of .NET code, including memory allocation and garbage
collection (which helps avoid memory leaks), security (including applying differing trust levels to code from different sources), thread management enforcing type-safety, and many other tasks [21].” The FCL contains all of the classes that make up ASP.NET as well as support for Windows and web applications, data access, web services, and more. The .Net Framework has the ability to compile five different languages (C#, Visual Basic, Managed C++, J#, and Jscript) [23].
Chapter 3: Loading Time Improvement

3.1 Server and Browser Test

The main problem with the Technology Tree application was the speed of the web-based application. The process that takes place during the application was not a simple process. In order to understand how to improve the speed of the application the process must be completely understood. In Table 3.1.1 the current process for the Technology Tree application is explained and it is shown in Figure 3.1.1. Each step in the application process was researched to identify potential improvements.

<table>
<thead>
<tr>
<th>Table 3.1.1: Application Process I</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A HTML page is called within the browser</td>
</tr>
<tr>
<td>2. A Flash swf file is accessed in the HTML file, on the browser</td>
</tr>
<tr>
<td>3. Inside the Flash swf a table is called</td>
</tr>
<tr>
<td>4. The Flash ActionScript sends the table name to ASP</td>
</tr>
<tr>
<td>5. The ASP uses the table name in an SQL statement to obtain the data from MySQL</td>
</tr>
<tr>
<td>6. The data is placed in a string in the ASP and then sent back to Flash</td>
</tr>
<tr>
<td>7. Flash loops through the data to find the parent/child relationships</td>
</tr>
<tr>
<td>8. The movie clips for the parent/child relationships are created</td>
</tr>
<tr>
<td>9. Flash displays the parent/child relationships in the form of a tree structure</td>
</tr>
</tbody>
</table>
The first step in the application process starts when an HTML page is called in a browser. Various tests were performed to see if the browsers or the server affected the loading time. Table 3.1.2 shows the test measurements in seconds.
Table 3.1.2: Browser and server speed comparisons (seconds)

<table>
<thead>
<tr>
<th>Trial</th>
<th>Server: IIS 5.1</th>
<th></th>
<th>Server: Windows Server 2003 (IIS 6.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Firefox</td>
<td>IE</td>
<td>Firefox</td>
</tr>
<tr>
<td>1</td>
<td>9.5</td>
<td>9.2</td>
<td>9.6</td>
</tr>
<tr>
<td>2</td>
<td>9.4</td>
<td>9</td>
<td>9.2</td>
</tr>
<tr>
<td>3</td>
<td>9.4</td>
<td>8.8</td>
<td>9.2</td>
</tr>
<tr>
<td>4</td>
<td>9.4</td>
<td>8.7</td>
<td>9.2</td>
</tr>
<tr>
<td>5</td>
<td>9.3</td>
<td>8.7</td>
<td>9.8</td>
</tr>
<tr>
<td>6</td>
<td>9.7</td>
<td>8.8</td>
<td>9.4</td>
</tr>
<tr>
<td>7</td>
<td>9.7</td>
<td>9</td>
<td>9.1</td>
</tr>
<tr>
<td>8</td>
<td>9.5</td>
<td>8.8</td>
<td>9.2</td>
</tr>
<tr>
<td>9</td>
<td>9.4</td>
<td>9</td>
<td>9.2</td>
</tr>
<tr>
<td>10</td>
<td>9.4</td>
<td>8.8</td>
<td>9.2</td>
</tr>
<tr>
<td>Average</td>
<td>9.47</td>
<td>8.88</td>
<td>9.3</td>
</tr>
</tbody>
</table>

The test was run on a Flash application that receives data from the database. The application tested was essentially the first six steps from Table 3.1.1. The same table was accessed for each trial. The database table had 158 rows of data. The first five trials were done within about five minutes. Then a couple hours later the next five trials were recorded. The testing was split into two parts because the network often fluctuates and the testing needed to be verified. The tests show that Internet Explorer (IE) is slightly faster than Mozilla Firefox and that IIS 6.0 is slightly faster than IIS 5.1. IE was on average 0.775 seconds faster than Mozilla Firefox. IIS 6.0 was on average 0.34 seconds faster than IIS 5.1. These results were minor when looking at the big picture, but the
results were good to know when the Defense Ammunition Center installed their server.

### 3.2 Developmental Server

Although the DAC planned to use their own server to run the Technology Tree application, a development server was set up at The University of Oklahoma. All of the appropriate software and updates were loaded on the OU server to duplicate the future DAC Server. The developmental server was key in understanding problems and it is assumed that testing times and comparisons of different methods will be similar between servers.

Security is a major issue when a government branch hosts a web-based application. There are specialized information technology professionals at the DAC that setup web servers and only applications that have been tested for security will be installed. One of the Technology Tree administrators was involved with the developmental server at The University of Oklahoma so that when the IT people were setting up the server, this administrator was able to guide them through the problems that were experienced at The University of Oklahoma.

However, there were a couple of issues that arose due to the difference in the server installations. First, occasionally new data would be uploaded to the server and the employees would not be able to see this new data for a day or two.
This means that the data was being cached somewhere between the
developmental server and the clients at the DAC. It was soon discovered that the
DAC has a Internet Cache Machine that is caching the data for the Technology
Tree application and is affecting their employees. After the problem was reported
to the DAC IT professionals, the problem was no longer encountered.

Another issue with the DAC server was the application would give an
error, “Error HTTP/1.1 100.” Initially this error did not make any sense, because the error could not be duplicated at The University of Oklahoma. The error was only seen at the DAC. The application showed that data was being sent, but then it reported the failure and stopped sending the data. It often required several refreshes to get the error. The problem was that the data gets scrubbed before it gets to the clients. This process was for security reasons, but it was causing the data to become corrupt when the clients received it. After the problem was reported to the DAC IT professionals, it was no longer encountered.

3.3 ASP and XML

The application must send data from the database to Macromedia Flash in an efficient way. XML is well known for its ability to transfer from one system to another and still be understood. In ASP, data may be read from a database and then using the “response.write” function, tags may be created to form an XML document. Since Flash has the capabilities of reading XML, the XML may be
used. The current ASP file creates an ADO connection with one of the DAC databases and then places all of the database data into a string that is read in Flash. The current ASP file is shown in Example 3.3.1. The method of placing the database data in a string is a slow method and must be improved.

Example 3.3.1: ASP that places the database data into a string and returned to browser
The use of XML has the potential to improve the performance because of its ability to create almost any kind of data structure. The current ASP file was modified to create elements for each field in the database. The modified page will access the database and then convert the data into XML. This was done by creating a loop that will create tags for every filed by using, `response.write "<" & x.name & ">" & x.value & "</" & x.name & ">". Also a new element was created that finds all of the children for each node. The children are found by using a SELECT statement that finds the id of the nodes whose parent is a certain id. The children were found so that parent/child relationships will be easier to find in Flash. The modified ASP file that creates the flat XML and also finds the children of each node is shown in Appendix A.

After the flat XML is created, Flash must be able to read the XML data. Since the XML is flat and does not have a tree structure the number of nodes is identified by counting the number of child nodes the first child has using ActionScript. Then, an ActionScript loop is executed based on the amount of child nodes the first child has. The ActionScript loop will continue to search through all of the nodes to find each nodes value. The Flash ActionScript that reads the XML is shown in Example 3.3.2. This example does not find the parent/child relationships within the data, it only reads the data in each element.
myLoadXML.onLoad = function (success)
{
if (success)
{
    performance_txt.text += "XML Data Loaded...
\n";
    temp_txt.text = "";
    for (var i = 0; i < this.firstChild.childNodes.length; i++)
    {
        for (var j = 0; j < this.firstChild.childNodes[i].childNodes.length; j++)
        {
            temp_txt.text += this.firstChild.childNodes[i].childNodes[j].firstChild.nodeValue + " ";
        }
        temp_txt.text += "\n";
    }
} else
{
    temp_txt.text = "file did not load successfully!";
}
}

Example 3.3.2: ActionScript function to process XML from ASP code

A Flash application was developed to test the speed of the modified ASP that creates the XML. Not only was the testing application used to test the modified ASP, but it was used to test any changes that were made to the application to see if time improvements were made. The testing application does everything the Technology Tree application does except find the data structure or visually display the data in a tree structure. Figure 3.3.1 shows a screenshot of the Flash application used for testing.
First a table is picked from the pull down menu in the upper right hand corner of the testing application, then the “Get Data” button must be pressed to start the application (Figure 3.3.1). Once the “Get Data” button is pressed the table name will be sent to the ASP file. The ASP file will use the table name in the SELECT statement to retrieve the data from the MySQL database. The ASP file will then create flat XML and sent it back to Flash. Flash will then read the data from the XML and show it in a text box on the right side of the testing
application. The time it took to obtain and display the data will be displayed in the text box on the left side of the testing application.

There are four different tables in the pull down menu in the Flash testing application. The four tables are all different in size and each of their characteristics are shown in Table 3.3.1. The tables variation in size creates an ideal testing environment to see how the testing application will handle the small tables as well as the large tables.

<table>
<thead>
<tr>
<th>Tables</th>
<th>Rows</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aa</td>
<td>7</td>
<td>23</td>
</tr>
<tr>
<td>HMLN</td>
<td>158</td>
<td>86</td>
</tr>
<tr>
<td>HMAT</td>
<td>295</td>
<td>102</td>
</tr>
<tr>
<td>HMSW</td>
<td>1256</td>
<td>234</td>
</tr>
</tbody>
</table>

The modified ASP that creates flat XML was tested in the Flash testing application for the four different tables. The time measurements were recorded and were slow. There was no improvement over the ASP that placed the data into a string. The time measurements for the modified ASP are shown in Table 3.3.2. Table HMSW took so long that Macromedia Flash had an error message pop up more than ten times while loading the data. The error says that the script is running slowly and then asks if you want to abort the script. This error message only appeared for the largest table, HMSW. The error message was answered
quickly to continue running the script to get the most accurate time measurement. This error message pops up when the same script in Macromedia Flash is being executed for a long period of time. The error message is shown in Figure 3.3.2.

<table>
<thead>
<tr>
<th>Trials</th>
<th>aa</th>
<th>HMLN</th>
<th>HMAT</th>
<th>HMSW</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.255</td>
<td>11.9</td>
<td>26.23</td>
<td>269.82</td>
</tr>
<tr>
<td>2</td>
<td>0.201</td>
<td>12.6</td>
<td>25.68</td>
<td>218.80</td>
</tr>
<tr>
<td>3</td>
<td>0.161</td>
<td>11.5</td>
<td>26.6</td>
<td>254.81</td>
</tr>
<tr>
<td>4</td>
<td>0.179</td>
<td>11.3</td>
<td>26.1</td>
<td>NA</td>
</tr>
<tr>
<td>5</td>
<td>0.183</td>
<td>12.0</td>
<td>28.6</td>
<td>NA</td>
</tr>
<tr>
<td>Average</td>
<td>0.195</td>
<td>11.9</td>
<td>26.6</td>
<td>247.8</td>
</tr>
</tbody>
</table>

Figure 3.3.2: Macromedia Flash error message

The Flash testing application was executed on The University of Oklahoma’s network, which is fast. If the testing application were executed on a slower network, or a slower computer, the time measurements would be larger, creating a slower application. Networks often do not have a constant internet bandwidth. The network’s bandwidth often fluctuates throughout the day. This
fluctuation of the network often leads to differences in loading times. Several measurements were taken during the testing of the Technology Tree application. The errors of the time measurements, due to the fluctuations of the network, are reduced by taking the average of the time measurements.

The test shows that this method is slow. No improvements were made from the original ASP that placed the database data in a string. The method tested is unacceptable and needed to be improved. The problem is the data takes too long to load. The largest table is only 234 KB and should be sent through the network and loaded in seconds instead of minutes. This means there is a problem with one of the methods used to have such a long loading time. The loading times started to grow exponentially based on the size of the table. Figure 3.2.3 shows a graph of the loading times for the Flash testing application. The graph represents the average loading time, based on ten time measurements for each table.
3.4 Scripting Language Upgrade

The Technology Tree application previously implemented Microsoft’s server-side scripting language, ASP. The use of ASP has decreased since the release of newer ASP.NET in January 2002. New books are no longer being published on ASP and even online forums are merging to ASP.NET forums. These circumstances have led to research in new types of server-side scripting.
languages that will increase the speed of the web-based application. Some of the available options are ASP.NET, Coldfusion, JSP, SSI, Perl, Python, and PHP.

The use of ASP within the Technology Tree application as described in Table 3.1.1 functions correctly but, the application is too slow. Two of the most well known and most capable server-side scripting languages are ASP.NET and PHP. PHP stands for Hypertext Preprocessor. PHP is an open source scripting language. The DAC information technology personnel have voiced concern for open-source technologies and would rather avoid them. Also MySQL is being used for development which is also open-source. A replacement for MySQL was also needed. The timing for searching for a new DBMS came right after the release of Microsoft’s SQL Server 2005. Along with the release was an Express Edition that was free to the public. SQL Server 2005 Express Edition was downloaded and was useful in researching methods for improving speed. The methods researched will be explained in section 3.5.3.

ASP.NET is much more powerful than classic ASP; it adds new features such as rich server controls, a more powerful programming model, and built-in support for XML web services. ASP.NET also allows users to access the full richness of the .NET Framework Class Library, which provides classes for everything from sending mail via SMTP to performing multithreaded operations [21]. Although ASP.NET is a more capable scripting language than classic ASP, that alone did not mean it would be implemented. There also had to be a speed
increase to implement ASP.NET technology into the Technology Tree application. ASP.NET has XML capabilities that will allow XML to be created and modified. The ASP used in the Technology Tree application has limited capabilities with XML. An improvement must be made because current methods used have unacceptable loading times.

Merging from ASP to ASP.NET is not a simple task because they both have different syntaxes. If an ASP file needs to be converted to ASP.NET most likely there will have to be changes made to the file and this can be confusing for an experienced ASP programmer. Creating an ASP.NET file that created XML took a long time to develop. It took even longer to develop an ASP.NET file that used some of the XML capabilities.

After a few weeks of learning ASP.NET, two different ASP.NET pages were created to use in the Flash testing application. One of the ASP.NET files creates its own tags by using the OdbcDataReader, part of the System.Data.Odbc namespace. The OdbcDataReader provides a way of reading a forward-only stream of data, obtained by the SELECT statement, from a data source [24]. The OdbcDataReader was used with MySQL because MySQL is identified and connected through the ODBC. Example 3.4.1 shows how the tags are created using the OdbcDataReader in the ASP.NET file. If the database in not connected through the ODBC, SqlDataReader will replace OdbcDataReader in Example 3.4.1.
MyCommand.CommandText = "SELECT * FROM " & tableName
Dim MyDataReader As OdbcDataReader = MyCommand.ExecuteReader
Dim XMLdata As String = "<?xml version="1.0" encoding="UTF-8"?>"
XMLdata &= "<datapacket>"
While MyDataReader.Read()
    XMLdata &= "<data>"
    XMLdata &= "<id>" & CStr(MyDataReader("id")) & "</id>"
    XMLdata &= "<node>" & MyDataReader("node") & "</node>"
    XMLdata &= "<parent>" & CStr(MyDataReader("parent")) & "</parent>"
    XMLdata &= "<type>" & MyDataReader("type") & "</type>"
    XMLdata &= "<status>" & MyDataReader("status") & "</status>"
    XMLdata &= "<terminal>" & CStr(MyDataReader("terminal")) & "</terminal>"
    XMLdata &= "<notes>" & MyDataReader("notes") & "</notes>"
    XMLdata &= "</data>"
Wend
XMLdata &= "</datapacket>"
Example 3.4.1: ASP.NET code that creates XML

The other ASP.NET script uses the dataset class, part of the System.Data namespace, in ASP.NET to retrieve the data. The dataset method is one of the XML capabilities within the ASP.NET syntax. The dataset automatically puts the data retrieved from a SELECT statement into XML. The XML created using the dataset method is flat. Example 3.4.2 shows the ASP.NET file that uses the dataset method.
Example 3.4.2: ASP.NET that creates XML using the dataset method

Both of the ASP.NET files created are capable of writing the XML to the page or to output the XML to an XML file. Flash has the ability to access either the XML file or the XML on the ASP.NET.

The loading times for the two ASP.NET files created must be compared to the loading times for the ASP file that creates XML and to the flat XML files for each table. The flat XML files were created by the ASP.NET dataset method and are stored on the server to be accessed by Flash. The Flash testing application was modified to handle the different testing cases. A pull down menu was added to allow the user to choose the ASP, ASP.NET that creates the XML, ASP.NET that uses the dataset, or the XML file. Also a scroll bar was added so that large tables may be viewed in the text box on the right side of the application. The text
box on the left side now tells the number of XML records that are read into Flash.

Figure 3.4.1 shows a screenshot of the modified flash testing application.

![Figure 3.4.1: Modified Flash testing application screenshot](image)

The four different cases were tested and the application showed a slight improvement in loading time when the ASP.NET files were implemented.

Appendix B shows the time measurements taken for the comparisons between ASP, ASP.NET, and the XML file. The dataset had the fastest loading times for
the larger tables, but these time measurements are still not typical for the amount of data that is being sent. Table 3.4.1 shows the average time for each method from the measurements in Appendix B. The errors due to the fluctuations in the network are reduced by taking the average of the time measurements.

<table>
<thead>
<tr>
<th>Input Method</th>
<th>.20</th>
<th>11.90</th>
<th>26.67</th>
<th>247.81</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP SQL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASP.NET SQL</td>
<td>.20</td>
<td>9.24</td>
<td>24.37</td>
<td>258.86</td>
</tr>
<tr>
<td>XML file</td>
<td>.03</td>
<td>8.75</td>
<td>26.50</td>
<td>227.09</td>
</tr>
<tr>
<td>ASP.NET dataset</td>
<td>.17</td>
<td>7.84</td>
<td>21.86</td>
<td>207.16</td>
</tr>
</tbody>
</table>

The time measurements are still exponentially increasing with the size of the table. The loading time for the smaller three tables is substantially smaller than the loading time for the largest table. In theory the XML file should have the shortest loading time because the XML is already created and Flash must only read the XML, where the other ASP and ASP.NET files must first create the XML. This proved that there is a problem with how Flash reads the XML. The average time measurements to obtain and read the data in Flash are shown in Figure 3.4.2. Figure 3.4.2 visually displays the measurements from Table 3.4.1.
In the Flash testing application a timer was created to find the time it takes to display each record on the screen. Creating the timer allowed a major problem to be identified. The first few records of table HMLN were taking about one millisecond each to display on the screen, but the records at the end of table HMLN were taking one hundred and fifty milliseconds. Each record at the beginning and at the end should be taking the same amount of time. Table 3.4.3 shows screenshots of the Flash testing application with the use of the new timer. The time per record is the integer below each row of data. This problem is
causing the larger tables, with many rows of data, to have a long loading time.

The smaller tables are not affected as much as the larger tables because the loading time increases after more records have been read and displayed.

Table 3.4.3: Times after each record 1

<table>
<thead>
<tr>
<th>XML Output</th>
<th>XML Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 HMLN 0 Part A 0 primary 2</td>
<td>The MSO is one of several technologies being developed for waste streams. MSO can supplement or reduce reliance on incineration. Other demilitarization processes may be used in the proper configuration for demilitarization in the future.</td>
</tr>
<tr>
<td>2 Unpacking 1 Existing A 0 The item is unpacked.</td>
<td>124 WS 120 Part A 0 Waste Stream 126,125</td>
</tr>
<tr>
<td>3 Disassembly 2 Existing A 0 The item is disassembled</td>
<td>145</td>
</tr>
<tr>
<td>4 Guidance and Control 3 Part A 0 N/A 5,14</td>
<td>123 Recycle 122 Existing A 1 N/A undefined</td>
</tr>
<tr>
<td>5 Recycle 9 Existing A 1 N/A undefined</td>
<td>140</td>
</tr>
<tr>
<td>6 Recycle 7 Existing A 1 N/A undefined</td>
<td>122 Clean 121 Part A 0 The item is free of explosives.</td>
</tr>
<tr>
<td>7 Recycle 8 Existing A 1 N/A undefined</td>
<td>148</td>
</tr>
<tr>
<td>8 Clean Metal Parts 3 Part A 0 N/A 9</td>
<td>121 PT 120 Part A 0 Inert Parts 122</td>
</tr>
<tr>
<td>9 Recycle 6 Existing A 1 N/A undefined</td>
<td>148</td>
</tr>
<tr>
<td>10 Recycle 5 Existing A 1 N/A undefined</td>
<td>120 Incinerate 119 Existing A 0 Item is incinerated 12</td>
</tr>
<tr>
<td>11 Recycle 4 Existing A 1 N/A undefined</td>
<td>143</td>
</tr>
</tbody>
</table>

3.4.1 Flash ActionScript Modification

The problem with large tables taking so long to load was identified in the Flash ActionScript that reads the XML. The Flash ActionScript searched through each child and then added the data from each element to a dynamic text variable called temp_txt.text (Example 3.2.2). The temp_txt.text variable is displayed in the text box on the right side of the Flash testing application. The reason the application was taking so long was because Flash was recursively adding data to
the text variable. This method was improved by changing the special text variable to a base variable and then setting that variable equal to a text variable at the end, after the loops. This prevented Flash from recursively adding the data to the text variable. The ActionScript modifications are highlighted in Example 3.4.1.1 and it shows the new implementation of the variable.

Example 3.4.1.1: Flash ActionScript with improved text variable

```
myLoadXML.onLoad = function (success)
{
    if (success)
    {
        performance_txt.text += "XML Data Loaded...
        performance_txt.text += "number of XML records = " + this.firstChild.childNodes.length + "\n";
        atemp ="
        for (var i = 0; i < this.firstChild.childNodes.length; i++)
        {
            D3 = getTimer();
            for (var j = 0; j < this.firstChild.childNodes[i].childNodes.length; j++)
            {
                atemp += this.firstChild.childNodes[i].childNodes[j].firstChild.nodeValue + " \n";
            }
            D4 = getTimer();
            Time_per_record = D4 - D3;
            atemp += "\n" + Time_per_record + "\n";
        }
        temp_txt.text = atemp;
        D2 = getTimer();
        Elapsed_Time = D2 - D1;
        performance_txt.text += "Elapsed Time = " + Elapsed_Time + " \n" + "***************\n";
        // function(mm, var_name, x1, y1, w, h, mc1r, bclr, depth_mc, startVal, endVal, numDecPlace)
        createSlider("text_line_mc", "text_line", 684, 224, 5, 320, 0xcccccc, 0x666666, 1000, 1, temp_txt.maxscroll, 0);
    }
    else
    {
        temp_txt.text = "file did not load successfully!";
    }
}
```

The Flash testing application was executed with the new Flash ActionScript that implements the new variable, for table HMLN. The records at the end of the table were taking the same amount of time as the records at the
beginning of the table. Table 3.4.1.1 shows how the records were taking the same amount of time at the beginning of the table and at the end of the table. The implementation of the variable in the ActionScript code prevents Flash from recursively adding data to the text variable. The integer after each record in Table 3.4.1.1 is the time it takes to read each record that was pulled from a database using a server-side scripting page.

<table>
<thead>
<tr>
<th>XML Output</th>
<th>Beginning of HMLN</th>
<th>End of HMLN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 HMLN 0 Part A 0 primary 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Unpadding 1 Existing A 0 The item is unpacked. Responses. 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Disassembly 2 Existing A 0 The item is disassembled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Guidance and Control 0 Part A 0 N/A 6, 14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 Recycle 0 Existing A 1 N/A undefined</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 Recycle 7 Existing A 1 N/A undefined</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 Recycle 8 Existing A 1 N/A undefined</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 Clean Metal Parts 5 Part A 0 N/A 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 Electronic 6 Part A 0 N/A 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time measurements were taken for all four tables using the new method to read the XML in Flash. Appendix C shows the time measurements taken from the Flash testing application for the ASP and ASP.NET files. Table 3.4.1.2 shows the average times for each input method from the measurements in Appendix C.
The table shows that the text file is the fastest. The text file method does not have to get data from the database and it does not have to create XML. The XML file is already created. The XML method was used for testing purposes only. It cannot be implemented into the Technology Tree application because the data in the database is constantly changing.

<table>
<thead>
<tr>
<th>Input Method</th>
<th>aa</th>
<th>HMLN</th>
<th>HMAT</th>
<th>HMSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP SQL</td>
<td>0.263</td>
<td>3.13</td>
<td>5.51</td>
<td>20.1</td>
</tr>
<tr>
<td>ASP.NET SQL</td>
<td>0.191</td>
<td>0.757</td>
<td>1.80</td>
<td>15.6</td>
</tr>
<tr>
<td>XML file</td>
<td>0.023</td>
<td>0.099</td>
<td>0.142</td>
<td>0.549</td>
</tr>
<tr>
<td>ASP.NET dataset</td>
<td>0.173</td>
<td>0.343</td>
<td>0.526</td>
<td>1.00</td>
</tr>
</tbody>
</table>

All of the methods made major improvements, but the dataset method had the largest improvement. The dataset method was able to access the largest table in the database, HMSC, and convert the data to XML, and then the XML data was read and displayed in Flash in one second. This test showed that the new XML capabilities in ASP.NET created a major improvement over the methods used in ASP. The fastest method tested in ASP.NET is more than 20 times faster for the largest table, than the ASP method. Figure 3.4.1.1 visually displays the measurements from Table 3.4.1.2.
Figure 3.4.1.1: Time measurements with the improved variable in flash

The dataset was the fastest for both the old Flash ActionScript method, that added text to the text variable, and the new Flash ActionScript method, that added text to a variable and then set a text variable equal the variable after the loop. Table 3.4.1.1 shows the comparisons between old and new methods of reading the data in Flash using the dataset input method. The larger the table the faster the newer method is when compared to the old method. The smallest table aa was almost the same speed as the previous version; this is due to the small amount of data being sent to Flash. The new dataset method that implements the
updated Flash Actionscript variable is 257.8 times faster than the ASP.NET method that created its own XML tags that implemented the old Flash ActionScript text variable.

<table>
<thead>
<tr>
<th>Tables</th>
<th>Old Time (sec)</th>
<th>New Time (sec)</th>
<th>Ratio</th>
<th>Times Faster</th>
</tr>
</thead>
<tbody>
<tr>
<td>aa</td>
<td>.168</td>
<td>0.17</td>
<td>1.03</td>
<td>0.970</td>
</tr>
<tr>
<td>HMLN</td>
<td>7.84</td>
<td>0.34</td>
<td>0.0438</td>
<td>22.8</td>
</tr>
<tr>
<td>HMAT</td>
<td>21.9</td>
<td>0.53</td>
<td>0.0241</td>
<td>41.5</td>
</tr>
<tr>
<td>HMSW</td>
<td>207</td>
<td>1.00</td>
<td>0.00480</td>
<td>206</td>
</tr>
</tbody>
</table>

These breakthroughs in the loading time made the application usable for the DAC employees, but only one of the XML capabilities have been implemented. Now it is time to look at some of the other XML capabilities within ASP.NET and SQL Server 2005.

3.5 Hierarchical Extensible Markup Language

3.5.1 Recursive Functions

The use of ASP.NET and XML has greatly improved the speed of the web-based application, but XML has the capability to be structured in a way so that Flash does not have to do several loops to find the parent/child relationships. Several methods were researched using some of the XML capabilities in ASP.NET and SQL Server.
One of the first methods researched was to create the XML structure by creating the beginning and end tags. It was recognized early in the research that this was not going to be a simple task. There are two different ways this could be accomplished. One way would be to use many loops with if statements, but this would only complicate the code and could potentially cause an increase in loading time instead of a decrease. Another way to achieve the XML structure would be to use a recursive function. A recursive function is a function that is defined by itself and calls itself in the function and continues to do so until a condition is met. One of the most basic recursive function examples is a function that will create a factorial for any number as shown in Example 3.5.1. Let $N = 5$; the function will execute once and then call factorial($4$), and this happens when $\text{factorial} = 5 \times \text{factorial}(4)$. Then factorial three will be called, creating $\text{factorial} = 5 \times 4 \times \text{factorial}(3)$. The function will not end until $N = 0$. When $N$ is equal to 0 the condition is supplied in the code, $\text{factorial} = 1$. These types of recursive functions will work well to find the parent/child relationships for the Technology Tree application.

```
Function factorial(N)
    If N = 0 Then
        factorial = 1
        Exit Function
    End If
    factorial = N * factorial(N - 1)
End Function
```

Example 3.5.1: Factorial recursive function [25]
One of the downsides of recursive functions is that they have the potential to use large amounts of memory and this could be detrimental to the Technology Tree application. The ASP.NET that created the XML tags was already one of the slower methods, and by adding a recursive function, this could have the potential to cause the loading time to increase. A new method must be implemented that uses some of the newer capabilities for creating XML and find the parent/child relationships in a short amount of time.

### 3.5.2 Methods for Creating Hierarchical XML

XML is being used more everyday and as more software companies use XML it will be necessary to transform one XML file format to another by using an Extensible Style Language Transformation (XSLT). “A transformation functions similarly to a stylesheet, except that instead of simply applying formatting rules to elements, it can alter the structure of a document to produce a document with a new structure [20].” The process of transforming an XML document is recursive because each node, as well as the nodes children, need to processed. This method has potential, but the XML must be created first and then the XSLT would be applied. This would add another step to the process and could potentially add more time to the loading process.
Another method to create structured XML is to use the XMLTextWriter object in ASP.NET. Elements and attributes can be created using the XMLTextWriter. First, the document must be started by using WriteStartDocument. Then the elements and attributes may be created. Example 3.5.2 shows an ASP.NET that implements the XMLTextWriter and creates hierarchical XML. The example shows how elements are created using the WriteStartElement. Four of the elements created in example 3.5.2 were not closed using WriteEndElement(). This was not an error, because the WriteEndDocument() closes any open tags.
Example 3.5.2: ASP.NET code using the XmlTextWriter object

The use of XMLTextWriter is a simple way to create hierarchical XML. The output to example 3.5.2 is shown in example 3.5.3. The use of a recursive function must be used to create hierarchical XML using the XmlTextWriter for such large tables as in the Technology Tree application. Appendix D shows how
the XmlTextWriter will work with data from a database and put the top level parent into XML.

<?xml version="1.0" encoding="utf-8" ?>
<root>
  <data>
    <id>1</id>
    <parent>0</parent>
    <data>
      <id>2</id>
      <parent>1</parent>
      <data>
        <id>3</id>
        <parent>2</parent>
        <data>
          <id>4</id>
          <parent>3</parent>
        </data>
      </data>
    </data>
  </data>
  <data>
    <id>5</id>
    <parent>1</parent>
    <data>
      <id>6</id>
      <parent>5</parent>
    </data>
  </data>
</root>

Example 3.5.3: Output XML file from the ASP.NET using the XmlTextWriter

The first two methods for creating hierarchical XML need a recursive function to actually produce hierarchical XML. However, instead of using a recursive function to find the parent/child relationships after the data is retrieved from the database, it is possible to use a recursive function in an SQL statement to
retrieve hierarchical XML from the database. This would allow the data to already be structured when it is called in ASP.NET.

3.5.3 SQL Stored Procedures

A stored procedure is made up of one or more SQL statements that are stored in a DBMS. Stored procedures are usually stored in a compiled form, thus preventing the DBMS from having to do more work, which improves performance [26].

MySQL 4 did not utilize stored procedures, but SQL Server 2005 did. Before SQL Server 2005 could be used the databases from MySQL needed to be converted to SQL Server. A tool was identified and used to convert all of the databases from MySQL to SQL Server [27]. Once the databases were converted stored procedures could be created and executed for the Technology Tree application.

The structure of an organizational chart could also be represented as a hierarchical XML file. An organizational chart may be obtained by using a recursive stored procedure to search through the employees in a database table to find the relationships between the employee and their bosses [28]. The Technology Tree application may use the same kind of recursive stored procedure to construct hierarchical XML.
Stored Procedures are used to store SQL statements in a database. Stored procedures may be used to create databases, create tables, give database permissions, add data to tables, etc… Creating hierarchical XML is not easy to do in SQL Server. SQL Server uses an SQL dialect called Transact-SQL that has several XML capabilities. These XML capabilities were researched for possible implementation into the Technology Tree application. Before these XML capabilities were fully investigated a recursive stored procedure was created that created hierarchical XML. The created stored procedure has an input parameter called @Root. @Root must be given to the stored procedure when the stored procedure is executed. The stored procedure can be executed by using the SQL command (Exec PrintHierarchy 0). The 0 is the input parameter that is used to find the parent in the stored procedure. Variables are declared and are given data types (DECLARE @id int). These variables are then set equal to SELECT statements (SET @Node = (SELECT Node FROM dbo.aa WHERE id = @Root)). Then another variable is used to set equal to a combination of text and variables that will create starting tags and ending tags for each element (SET @Result = ('<data><id>'+CAST(@Root AS varchar)+'</id>...'). Next, @id is set equal to another child of the parent by using a SELECT statement (SET @id = (SELECT MIN(id) FROM dbo.aa WHERE parent = @Root)). If there are children left, the stored procedure will be executed again using the other child’s id (WHILE @id IS NOT NULL BEGIN EXEC dbo.PrintHierarchy @id). This recursive process will continue
until there are no more children for that element and then an end tag is created. A new element was also created in the recursive stored procedure. The new element tells the level at which the data is on. The root node is at level 1 and its children are at level 2, etc… This level will be beneficial when Flash reads the hierarchical XML. Example 3.5.3.1 shows the recursive stored procedure created in SQL Server that will create structured XML.

```sql
set ANSI_NULLS ON
set QUOTED_IDENTIFIER ON
go
CREATE PROC [dbo].[PrintHierarchy]
(  
    @Root int
)
AS
BEGIN
  SET NOCOUNT ON
  DECLARE @id int, @Node varchar(30), @Terminal int, @Status varchar(10), @Parent int,
  @Notes varchar, @Type varchar(50), @Result varchar(255), @Result2 varchar(255)
  SET @Node = (SELECT Node FROM dbo.aa WHERE id = @Root)
  SET @Terminal = (SELECT Terminal FROM dbo.aa WHERE id = @Root)
  SET @Status = (SELECT Status FROM dbo.aa WHERE id = @Root)
  SET @Parent = (SELECT Parent FROM dbo.aa WHERE id = @Root)
  SET @Type = (SELECT Type FROM dbo.aa WHERE id = @Root)
  SET @Result = ('<data><id>'+CAST(@Root AS varchar)+'</id><node>'+@Node+'</node><parent>'+CAST(@Parent AS varchar)+'</parent><type>'+ @Type+'</type><status>'+@Status+'</status><terminal>' +CAST(@Terminal AS varchar)+'</terminal><level>'+CAST(@@NESTLEVEL-2 AS varchar) +'</level>')
  SELECT @Result
  SET @id = (SELECT MIN(id) FROM dbo.aa WHERE parent = @Root)
  WHILE @id IS NOT NULL
  BEGIN
    EXEC dbo.PrintHierarchy @id
    SET @Result2 = ('</data>')
    SELECT @Result2
    SET @id = (SELECT MIN(id) FROM dbo.aa WHERE parent = @Root AND id > @id)
  END
END
```

Example 3.5.3.1: SQL stored procedure to create XML
When the stored procedure is executed a table is created after each time a
SELECT @Result or SELECT @Resutl2 statement is executed, thus making the
output confusing. This problem also makes reading the data from ASP.NET
difficult. The solution to reading the structured XML data in ASP.NET is to
create a temporary table in the stored procedure and then add data to the
temporary table instead of using SELECT. A temporary table is denoted by a #
character before the table name. The use of the temporary table is highlighted in
Example 3.5.3.2. The SQL execution statement is different for this stored
procedure, because the temporary table will be created and then dropped
(deleted). Example 3.5.3.3 shows the execution statement.
Example 3.5.3.2: SQL stored procedure that uses a temporary table

| Create Table #hier (data varchar(max)) |
| Exec ShowHierarchy 0 |
| Select data From #hier |
| Drop Table #hier |

Example 3.5.3.3: SQL execution statement with temporary table
The stored procedure produces correct hierarchical XML data, but how fast is it? This stored procedure was tested in SQL Server Management Studio Express for the four different trial tables. Stored procedures that accessed tables aa, HMLN, and HMAT all were executed in under one second, but the largest table HMSW was executed in about 13 seconds. This time is still too slow for the Technology Tree application. Although the time was small for the smaller tables, the time for the larger table is unacceptable. The stored procedure was modified to create attributes instead of elements. The use of attributes reduces the amount of data that is needed to be printed to the screen. Elements need end tags, where attributes do not. The attributes were created by modifying the @Result variable. The @Result variable shows that the attributes were reduced to either a single letter name or two letter name (Set @Result = '<data i="' + CAST(@Root AS varchar) + '" n="' + @Node + '" p="' + CAST(@Parent AS varchar) + '" ty="' + @Type + '" s="' + @Status + '" te="' + CAST(@Terminal AS varchar) + '" l="' + CAST(@@NESTLEVEL-2 AS varchar) + '"'>). Example 3.5.3.4 shows the modified stored procedure that implements attributes.
Example 3.5.3.4: SQL stored procedure that uses attributes

The new stored procedure that uses attributes greatly reduced the amount of data, but when it was tested in SQL Server Express Edition there was little change in the loading time. The stored procedure still took more than 13 seconds to execute. This stored procedure will only cause the application to be slower.

Table 3.5.3.1 shows the times and the output sizes of the stored procedures that implement elements and attributes.
Transact-SQL, the SQL dialect used in SQL Server, has several XML capabilities that may be researched for use in the Technology Tree application. One of the XML capabilities is FOR XML. FOR XML has four different types of modes; RAW, AUTO, EXPLICIT, and PATH. Both RAW and AUTO create XML where the columns are made as attributes for each row. EXPLICIT allows the developer to control how the XML is generated. The easiest method to produce complex XML document is to use FOR XML PATH. In order to use these capabilities with the data from the DAC’s database a recursive function must be developed.

User-defined functions may be created and stored in SQL Server. A recursive scalar valued user-defined function was developed using FOR XML PATH. The recursive function uses the Transact-SQL CASE…WHEN statement.

<table>
<thead>
<tr>
<th>Table</th>
<th>aa</th>
<th>HMLN</th>
<th>HMAT</th>
<th>HMSW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Time with Elements (sects)</td>
<td>.019</td>
<td>.18</td>
<td>.93</td>
<td>13.25</td>
</tr>
<tr>
<td>Average Time with Attributes (sects)</td>
<td>.017</td>
<td>.17</td>
<td>.92</td>
<td>13.22</td>
</tr>
<tr>
<td>Output Size with Elements (KB)</td>
<td>1.3</td>
<td>27.7</td>
<td>53.3</td>
<td>229.5</td>
</tr>
<tr>
<td>Output Size with Attributes(KB)</td>
<td>0.8</td>
<td>16.9</td>
<td>33.2</td>
<td>143.8</td>
</tr>
</tbody>
</table>
The CASE…WHEN is similar to an IF…THEN but one major difference is that the CASE…WHEN may be in the SELECT statement. In the function, WHEN parent=@@id THEN dbo.structure(id), is where the function is recursively called. When the function is executed again it will find the children data. The recursive function created is shown in example 3.5.3.5. In order to execute this recursive, user-defined function, it must be called. A stored procedure was created to call the recursive function. The stored procedure implements FOR XML PATH as well. The stored procedure is shown in example 3.5.3.6.

```
set ANSI_NULLS ON
set QUOTED_IDENTIFIER ON
go
CREATE FUNCTION [dbo].[structure](@id int)
RETURNS XML
WITH RETURNS NULL ON NULL INPUT
BEGIN RETURN
(SELECT id as '@id', node as '@node', parent as '@parent', type as '@type', status as '@status', terminal as '@terminal', notes as '@notes',
CASE WHEN parent=@@id
THEN dbo.structure(id)
END
FROM aa WHERE parent=@@id
FOR XML PATH, TYPE)
END
```

Example 3.5.3.5: SQL Scalar valued, user-defined function
Example 3.5.3.6: SQL stored procedure that calls the scalar valued, user-defined function

The new stored procedure that utilizes the recursive user-defined function was tested in SQL Server Management Studio Express for the time it takes the stored procedure to execute. Table 3.5.3.2 shows the times it took to execute the new stored procedure that calls the recursive, user-defined function. Note, all times are in milliseconds since they are small.

<table>
<thead>
<tr>
<th>Table 3.5.3.2: Stored procedure execution time using a recursive function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>Average Time (seconds)</td>
</tr>
<tr>
<td>Output Size (KB)</td>
</tr>
</tbody>
</table>
Two stored procedures were tested earlier in this chapter and their times are shown in Table 3.5.3.1. Their execution times were significantly higher than the times shown in Table 3.5.3.2. Also the stored procedures discussed earlier in the chapter did not include the notes field, which is the largest field. The new method includes the notes field and is still up to 5.5 times faster. This is a major improvement in the production of XML hierarchies. This method is fast and also implements the notes field into the XML.

The new stored procedure that accesses the recursive function must now be accessed from an ASP.NET page and then sent to Flash. Database driven applications are sometimes difficult because everything must work dynamically. This means that when a certain table is chosen in Flash the ASP.NET knows what table to input into the SQL statement. This information is sent between Flash and ASP.NET. Implementing the table name into the SQL statement is not difficult for basic SQL SELECT statements, but for complex statements, like the one developed, it is difficult. This was accomplished dynamically by having two input parameters for the stored procedure; one being an integer “0” and the other, the table name. Then the recursive, user-defined function that is called has the table name in the function name allowing for only one stored procedure, but there must be a function for every table. The dynamic stored procedure is shown in example 3.5.3.7.
Example 3.5.3.7: Dynamic Stored Procedure

3.6 Macromedia Flash Upgrade

3.6.1 Movie Clip Creation

The Technology Tree application accesses movie clips from a library. All of the node movie clips are inside the library in Flash. These nodes are what the application uses in the tree structure, depending on the data of the node. There are different movie clips for certain nodes based on whether it is a part, an existing node, an emerging node, a terminal node, or several other kinds of characteristics. These movie clips are created by hand and placed in the library, and then throughout the ActionScript code these movie clips are called and used in the tree design. This method seems to work well, but there are several other methods that should be investigated for time improvements.
Three different methods were researched and tested in Flash for loading time improvements. The first method researched was a method that draws a circle using ActionScript and then it was created 200 times by using a loop (Method 1). The next method researched used a circle that was drawn on the screen using the Flash drawing tools, and then it is duplicated 2000 times using the following ActionScript code: `c.duplicateMovieClip("c"+i,i,(_x:my_x,_y:my_y))`, where `c` is the name of the movie clip (Method 2). The last method researched created a circle using ActionScript code and then it was duplicated 2000 times (Method 3). The Technology Tree application was initially developed in Flash MX (Flash Version 6), but now there are two more versions that are available to the public. These new versions will be tested and compared with Flash 6 using the three different methods for accessing movie clips. Table 3.6.1.1 shows the time measurements for the three different versions of Flash. Each version of Flash was timed on how fast the movie clips were drawn and how fast the movie clips were moved. Each method had a considerable difference in loading time, but the biggest surprise was the major difference between the Flash version 6 and Flash 7. The fastest method in every Flash version was the method that created a circle using ActionScript code and then it was duplicated 2000 times. This method was only slightly faster than Method 2, in which a circle was drawn on the screen using the Flash drawing tools and then duplicated 2000 times.
Table 3.6.1.1: Movie Clip Creation in Different Flash Versions (sec)

<table>
<thead>
<tr>
<th>Movie Clip Method</th>
<th>Flash 6</th>
<th></th>
<th>Flash 7</th>
<th></th>
<th>Flash 8</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Drawing</td>
<td>Moving</td>
<td>Drawing</td>
<td>Moving</td>
<td>Drawing</td>
<td>Moving</td>
</tr>
<tr>
<td>Method 1</td>
<td>8.33</td>
<td>0.327</td>
<td>2.57</td>
<td>0.110</td>
<td>2.57</td>
<td>0.107</td>
</tr>
<tr>
<td></td>
<td>7.98</td>
<td>0.325</td>
<td>2.61</td>
<td>0.111</td>
<td>2.56</td>
<td>0.105</td>
</tr>
<tr>
<td></td>
<td>7.95</td>
<td>0.327</td>
<td>3.00</td>
<td>0.130</td>
<td>2.53</td>
<td>0.104</td>
</tr>
<tr>
<td></td>
<td>7.63</td>
<td>0.315</td>
<td>2.59</td>
<td>0.108</td>
<td>2.54</td>
<td>0.104</td>
</tr>
<tr>
<td></td>
<td>8.07</td>
<td>0.323</td>
<td>2.75</td>
<td>0.111</td>
<td>2.61</td>
<td>0.107</td>
</tr>
<tr>
<td>Average</td>
<td>7.99</td>
<td>0.323</td>
<td>2.70</td>
<td>0.114</td>
<td>2.56</td>
<td>0.105</td>
</tr>
<tr>
<td>Total</td>
<td>8.32</td>
<td>2.82</td>
<td>2.670</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Method 2</td>
<td>0.702</td>
<td>0.316</td>
<td>0.355</td>
<td>0.112</td>
<td>0.393</td>
<td>0.117</td>
</tr>
<tr>
<td></td>
<td>0.739</td>
<td>0.364</td>
<td>0.373</td>
<td>0.107</td>
<td>0.369</td>
<td>0.111</td>
</tr>
<tr>
<td></td>
<td>0.757</td>
<td>0.378</td>
<td>0.391</td>
<td>0.112</td>
<td>0.381</td>
<td>0.110</td>
</tr>
<tr>
<td></td>
<td>0.777</td>
<td>0.359</td>
<td>0.395</td>
<td>0.123</td>
<td>0.378</td>
<td>0.115</td>
</tr>
<tr>
<td></td>
<td>0.739</td>
<td>0.342</td>
<td>0.379</td>
<td>0.118</td>
<td>0.370</td>
<td>0.112</td>
</tr>
<tr>
<td>Average</td>
<td>0.743</td>
<td>0.352</td>
<td>0.379</td>
<td>0.114</td>
<td>0.378</td>
<td>0.113</td>
</tr>
<tr>
<td>Total</td>
<td>1.09</td>
<td>0.493</td>
<td>0.491</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Method 3</td>
<td>0.173</td>
<td>0.289</td>
<td>0.174</td>
<td>0.109</td>
<td>0.174</td>
<td>0.116</td>
</tr>
<tr>
<td></td>
<td>0.200</td>
<td>0.315</td>
<td>0.196</td>
<td>0.117</td>
<td>0.174</td>
<td>0.104</td>
</tr>
<tr>
<td></td>
<td>0.180</td>
<td>0.292</td>
<td>0.181</td>
<td>0.113</td>
<td>0.169</td>
<td>0.109</td>
</tr>
<tr>
<td></td>
<td>0.186</td>
<td>0.311</td>
<td>0.176</td>
<td>0.108</td>
<td>0.181</td>
<td>0.111</td>
</tr>
<tr>
<td></td>
<td>0.180</td>
<td>0.303</td>
<td>0.197</td>
<td>0.131</td>
<td>0.178</td>
<td>0.118</td>
</tr>
<tr>
<td>Average</td>
<td>0.184</td>
<td>0.302</td>
<td>0.185</td>
<td>0.116</td>
<td>0.175</td>
<td>0.112</td>
</tr>
<tr>
<td>Total</td>
<td>0.486</td>
<td>0.300</td>
<td>0.287</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The test showed that the Technology Tree application must be upgraded to at least Flash 7, also known as Flash MX 2004. Although Flash 8 was slightly faster in all three methods than Flash 7, there is not a large enough difference for the conversion to take place. Figure 3.6.1.1 shows a graph of the speeds of all three methods including the Flash versions.
3.6.2 Flash MX 2004 (Flash 7)

The Flash version test showed that the Technology Tree application needs to be upgraded to Flash 7. Upgrading to Flash 7 sounded much simpler than it actually ended up becoming. One of the differences between Flash 6 and Flash 7 besides the speed differences is that Flash 6 is not case sensitive where Flash 7 is. This caused the application to have errors. This problem typically is easy to solve, but making sure the cases are the same for all variables is a tedious task in an application with so much coding. This problem led to improvements in the ActionScript in Flash during the search for case-sensitive mistakes. The code was
better organized and more comments were added to explain what was being accomplished.
Chapter 4: Database Driven Applications

4.1 Web-Based Applications

Large companies often have problems with work being repeated. The best solution to this problem is to have a database that anyone in the company can access. This database must be structured so that the employees will know where to find what they are looking for. Database driven, web-based applications are a better solution to allow the company’s employees to have instant access whenever they are on the web.

Having access to readily available materials related to an employee’s field, leads to higher production and thus more profit for the employee’s company. The use of database driven applications are key parts to major companies. These applications may include CAD drawings, project requirements, or even engineering procedures to assist the companies employees in their everyday work.

Companies are willing to invest in these applications, because they see the potential in how these could increase productivity. Some companies are using these database-driven applications to store all past files. This allows their employees to have access to all the past documents from one centralized location.

One of the main issues with databases is security. Proprietary information is often stored on databases. It is important to not let this proprietary information
get into the wrong hands, and this is being satisfied by having more security options in databases. The databases being developed today are secure and are capable of becoming even more secure.

### 4.2 Industry Experience

Lockheed Martin Space Systems Company had many databases that their employees had access to that greatly benefited the company’s production. One of the main databases used is a database called DOORS. DOORS was a database that was used to store the project’s requirements. The requirements from the customer are usually in large documents that completely explain in detail each requirement. Links are made to each of the requirements on a side bar that allow for easier navigation through the documents. All of the requirements are then put into an Excel spreadsheet that has more information about the requirement. The Excel spreadsheet included the paragraph number of where the requirement is within the document, the lead engineer for that requirement, and also the topic of that requirement. This was not the only database driven application used at Lockheed Martin Space Systems Company. There was also a database driven applications for Risk Management and for past documents related to the project.

The technologies learned in the past year in this research have given the author a key understanding of database driven, web-based applications. The
author sees the potential of creating database driven, web-based application like the Technology Tree application and now has the ability to do so.
Chapter 5: Conclusion

5.1 Summary

The Technology Tree application has received many improvements over the past year. The development of this application has impressed many U.S. Army officials and the employees at the Defense Ammunition Center. New improvements are seen for the future of the application including adding pictures (jpeg and gif) and pdf files to the database.

The methods used to upgrade the application have implemented some of the newest software technologies available. Using such new software led to new improvements in the speed of the application as well as breakthroughs that are beneficial to future applications.

The Defense Ammunition Center has high expectations for this application in the upcoming years. The research done has led to the application being implemented by the DAC and the U.S Army for the demilitarization of ammunition all over the world. The DAC has released the Technology Tree application to the defense community. Several of the loading improvements made by this research were implemented into the Technology Tree application. The demilitarization industry is large and affects a large number of people. The DAC
also sees the application used in new areas such as item production, maintenance, storage, and development.

5.2 Accomplishments

XML was first created in ASP using a string that created its own tags. The application was tested using this method and was slow. The server-side scripting language was upgraded from ASP to ASP.NET to use some of the new XML capabilities in ASP.NET. Several XML methods were tested in ASP.NET and the results are shown in Figures 3.4.2 and 3.4.1.1. The ActionScript was modified to read the data faster and now the loading time is over 250 times faster than the previous version of the application. The Flash file was upgraded from Flash 6 to Flash 7 and this saved valuable time in the creation of the movie clips as well as the movement of the movie clips. MySQL 4 was replaced by the more capable SQL Server 2005. In SQL Server 2005, a stored procedure and scalar valued, user-defined functions were created. The stored procedure is executed from an ASP.NET page, and then the recursive function is called, where the hierarchical XML is created. The hierarchical XML is created in under 2.4 seconds for the largest table. All of these accomplishments led to improvements in the loading time or reduced the amount of coding for the Technology Tree application.

The creation of a stored procedure that creates structured data allows the Technology Tree application to be fast for the users. The structured data is in a
hierarchical XML format that may be read in Macromedia Flash. Flash uses this hierarchical XML to create movie clips to visually display a tree structure that may be used to demilitarize ammunition.

5.3 Benefits to Future Applications

The hierarchical XML algorithm developed in this research may be used in future applications to create structured data. Almost all data is structured in some format or another and XML is the one of the best ways to structure data. The need for data to be in XML files is becoming more apparent than ever before, especially with Microsoft Office 2007. In Microsoft Office 2007 the default save option is XML and this will cause other software companies to do the same. When software companies start to implement XML, there will be a need to be able to store XML files in databases. New databases are being developed that will store XML files in their native format, XML. These databases are called Native XML Databases.

The increased use of XML will create many uses for the hierarchical XML algorithm developed for the Technology Tree application. This type of algorithm may be used to convert all non-structured data in databases to structured, hierarchical XML files.
5.4 The Project’s Future

The Technology Tree application has a bright future. There are plans to apply the application to new areas at the DAC such as item production, maintenance, storage, and development. There are also plans of adding pdf, jpeg, and gif files to the database so that these file formats may be accessed in the Technology Tree application.

The hierarchical XML algorithm was not implemented into the current production version of the Technology Tree application. The tests completed in the research have shown the benefits of implementing this algorithm. The use of the algorithm will create a new application process. The operation steps are explained in Table 5.4.1 and Figure 5.4.1 shows this process.

<table>
<thead>
<tr>
<th>Table 5.4.1: Application Operation Steps II</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A HTML page is called within the browser</td>
</tr>
<tr>
<td>2. A Flash swf file is accessed in the HTML file, on the browser</td>
</tr>
<tr>
<td>3. Inside the Flash swf a table is called</td>
</tr>
<tr>
<td>4. The Flash ActionScript sends the table name to an ASP.NET</td>
</tr>
<tr>
<td>5. The ASP.NET uses the table name as an input parameter for an SQL stored procedure in SQL Server 2005</td>
</tr>
<tr>
<td>6. The stored procedure calls the scalar valued, user-defined function that recursively finds the parent/child relationships and outputs them in XML</td>
</tr>
<tr>
<td>7. Flash then reads the ASP.NET file with the XML data</td>
</tr>
<tr>
<td>8. Flash reads the structured XML and creates movie clips based on the structure</td>
</tr>
<tr>
<td>9. Flash displays the parent/child relationships in the form of a tree structure</td>
</tr>
</tbody>
</table>
Although the new technologies implemented into the application have made improvements, there will eventually be better software programs to create a faster application. New software programs will be released and must be researched for possible implementation into the Technology Tree application. One of the newest software programs that could be implemented into the application is Macromedia Flex. Flex is an application server that compiles an XML markup language (MXML) and ActionScript to create a Flash application. Flex 2.0 uses ActionScript 3.0, which treats XML as a native data type instead of having to open and read the data from an XML file. Another technology that has potential to be implemented into the Technology Tree application is the use of Asynchronous JavaScript and XML (AJAX). The AJAX method sends XML.
back and forth between the server and the browser and allows JavaScript to use the received data to update a web page without having to refresh [29]. AJAX is implemented in Gmail. The methods in AJAX have been implemented into Flash and have been called Asynchronous Flash and XML (AFLAX). Macromedia Flex has several AFLAX capabilities. Also the use of a Native XML DBMS to store the structured XML instead of having to create the structure each time a table is called in the application are possible improvements for the application. Also the use of an ASP.NET Tree View or a Flash Tree component could allow for easier navigation through the Technology Tree application.

This project has a bright future and will continue to be used for a long time. During its lifetime there will be many other applications developed that are similar to the Technology Tree application.
References


(accessed on 13 April 2006).

(accessed on 16 April 2006).


Appendix A: ASP that Creates XML

```xml
<%@ Language = VBSCRIPT %>
<%
Server.ScriptTimeout = 1000 'time out in 1000 sec (16 min)
response.buffer=false
'data sent by XML without any element tags (cheated)
lngByteSize = Request.TotalBytes
tableName = BinaryToString(Request.BinaryRead(lngByteSize))
'set up connection to database
set adoConn = Server.CreateObject("ADODB.Connection")
adoConn.Open "********"
;set up record set and send SQL command
set adoRS = Server.CreateObject("ADODB.Recordset")
adoRS.ActiveConnection = adoConn
adoRS.Open "SELECT * FROM " & tableName
response.write "<?xml version="1.0" encoding="UTF-8"?>"
response.write "<datapacket>"
echoString = tableName & " 
while not adoRS.EOF
    response.write "<DATA>"
    for each x in adoRS.Fields   'write each field as a new element
        response.write "<" & x.name & ">" & x.value & "</" & x.name & ">"
     next
    'get all kids using 2nd SQL statement
    set adoRS2=Server.CreateObject("ADODB.Recordset")
tempstring=adoRS.Fields("id")  'current node id
query="SELECT id FROM " & tableName & " WHERE parent=" & tempstring
adoRS2.Open query,adoConn
    response.write "<kids>"
    ii = 1
    while not adoRS2.EOF
        If ii = 1 then
            response.write adoRS2.Fields("id")
        Else
            response.write "," & adoRS2.Fields("id")
        End If
        ii = ii + 1
        adoRS2.MoveNext
    wend
    response.write "</kids>"
response.write "</DATA>"
adoRS.MoveNext
wend
response.write "</datapacket>"
'write text file to test
Set filesys = CreateObject("Scripting.FileSystemObject")
Set txtfile = filesys.CreateTextFile("c:\temp\tempfile.txt", True, False)
'overwrite and no unicode
txtfile.WriteLine(echoString)
txtfile.Close
'converts binary data from Flash (XML stuff) to a normal string with XML tags
```
Function BinaryToString(Binary)
    Dim I, S
    For I = 1 To LenB(Binary)
        S = S & Chr(AscB(MidB(Binary, I, 1)))
    Next
    BinaryToString = S
End Function
## Appendix B: Time measurements I

<table>
<thead>
<tr>
<th>Trials</th>
<th>ASP SQL</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>aa</td>
<td>HMLN</td>
<td>HMAT</td>
<td>HMSW</td>
</tr>
<tr>
<td>1</td>
<td>255</td>
<td>11983</td>
<td>26231</td>
<td>269818</td>
</tr>
<tr>
<td>2</td>
<td>201</td>
<td>12598</td>
<td>25678</td>
<td>218799</td>
</tr>
<tr>
<td>3</td>
<td>161</td>
<td>11568</td>
<td>26597</td>
<td>254807</td>
</tr>
<tr>
<td>4</td>
<td>179</td>
<td>11311</td>
<td>26190</td>
<td>NA</td>
</tr>
<tr>
<td>5</td>
<td>183</td>
<td>12032</td>
<td>28671</td>
<td>NA</td>
</tr>
<tr>
<td>Average</td>
<td>195.8</td>
<td>11898.4</td>
<td>26673.4</td>
<td>247808</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trials</th>
<th>ASP.NET SQL</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>aa</td>
<td>HMLN</td>
<td>HMAT</td>
<td>HMSW</td>
</tr>
<tr>
<td>1</td>
<td>214</td>
<td>9752</td>
<td>23255</td>
<td>258864</td>
</tr>
<tr>
<td>2</td>
<td>229</td>
<td>9152</td>
<td>24447</td>
<td>NA</td>
</tr>
<tr>
<td>3</td>
<td>196</td>
<td>9100</td>
<td>22781</td>
<td>NA</td>
</tr>
<tr>
<td>4</td>
<td>179</td>
<td>9179</td>
<td>24853</td>
<td>NA</td>
</tr>
<tr>
<td>5</td>
<td>185</td>
<td>9023</td>
<td>26499</td>
<td>NA</td>
</tr>
<tr>
<td>Average</td>
<td>200.6</td>
<td>9241.2</td>
<td>24367</td>
<td>258864</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trials</th>
<th>ASP.NET text file</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>aa</td>
<td>HMLN</td>
<td>HMAT</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>8667</td>
<td>24829</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>8786</td>
<td>27908</td>
</tr>
<tr>
<td>3</td>
<td>29</td>
<td>8697</td>
<td>27066</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>8879</td>
<td>26190</td>
</tr>
<tr>
<td>5</td>
<td>23</td>
<td>8716</td>
<td>26531</td>
</tr>
<tr>
<td>Average</td>
<td>30.2</td>
<td>8749</td>
<td>26504.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trials</th>
<th>ASP.NET dataset</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>aa</td>
<td>HMLN</td>
<td>HMAT</td>
<td>HMSW</td>
</tr>
<tr>
<td>1</td>
<td>150</td>
<td>8175</td>
<td>19372</td>
<td>207155</td>
</tr>
<tr>
<td>2</td>
<td>191</td>
<td>7804</td>
<td>23513</td>
<td>NA</td>
</tr>
<tr>
<td>3</td>
<td>148</td>
<td>7722</td>
<td>22520</td>
<td>NA</td>
</tr>
<tr>
<td>4</td>
<td>160</td>
<td>7846</td>
<td>22683</td>
<td>NA</td>
</tr>
<tr>
<td>5</td>
<td>192</td>
<td>7643</td>
<td>21194</td>
<td>NA</td>
</tr>
<tr>
<td>Average</td>
<td>168.2</td>
<td>7838</td>
<td>21856.4</td>
<td>207155</td>
</tr>
</tbody>
</table>
# Appendix C: Time measurements II

<table>
<thead>
<tr>
<th>Trials</th>
<th>ASP SQL</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Trials</td>
<td>aa</td>
<td>HMLN</td>
<td>HMAT</td>
</tr>
<tr>
<td>1</td>
<td>255</td>
<td>11983</td>
<td>26231</td>
<td>269818</td>
</tr>
<tr>
<td>2</td>
<td>201</td>
<td>12598</td>
<td>25678</td>
<td>218799</td>
</tr>
<tr>
<td>3</td>
<td>161</td>
<td>11568</td>
<td>26597</td>
<td>254807</td>
</tr>
<tr>
<td>4</td>
<td>179</td>
<td>11311</td>
<td>26190</td>
<td>NA</td>
</tr>
<tr>
<td>5</td>
<td>183</td>
<td>12032</td>
<td>28671</td>
<td>NA</td>
</tr>
<tr>
<td>Average</td>
<td>195.8</td>
<td>11898.4</td>
<td>26673.4</td>
<td>247808</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trials</th>
<th>ASP.NET SQL</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Trials</td>
<td>aa</td>
<td>HMLN</td>
<td>HMAT</td>
</tr>
<tr>
<td>1</td>
<td>214</td>
<td>9752</td>
<td>23255</td>
<td>258864</td>
</tr>
<tr>
<td>2</td>
<td>229</td>
<td>9152</td>
<td>24447</td>
<td>NA</td>
</tr>
<tr>
<td>3</td>
<td>196</td>
<td>9100</td>
<td>22781</td>
<td>NA</td>
</tr>
<tr>
<td>4</td>
<td>179</td>
<td>9179</td>
<td>24853</td>
<td>NA</td>
</tr>
<tr>
<td>5</td>
<td>185</td>
<td>9023</td>
<td>26499</td>
<td>NA</td>
</tr>
<tr>
<td>Average</td>
<td>200.6</td>
<td>9241.2</td>
<td>24367</td>
<td>258864</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trials</th>
<th>ASP.NET text file</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Trials</td>
<td>aa</td>
<td>HMLN</td>
<td>HMAT</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>8667</td>
<td>24829</td>
<td>271715</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>8786</td>
<td>27908</td>
<td>NA</td>
</tr>
<tr>
<td>3</td>
<td>29</td>
<td>8697</td>
<td>27066</td>
<td>NA</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>8879</td>
<td>26190</td>
<td>NA</td>
</tr>
<tr>
<td>5</td>
<td>23</td>
<td>8716</td>
<td>26531</td>
<td>NA</td>
</tr>
<tr>
<td>Average</td>
<td>30.2</td>
<td>8749</td>
<td>26504.8</td>
<td>271715</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trials</th>
<th>ASP.NET dataset</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Trials</td>
<td>aa</td>
<td>HMLN</td>
<td>HMAT</td>
</tr>
<tr>
<td>1</td>
<td>150</td>
<td>8175</td>
<td>19372</td>
<td>207155</td>
</tr>
<tr>
<td>2</td>
<td>191</td>
<td>7804</td>
<td>23513</td>
<td>NA</td>
</tr>
<tr>
<td>3</td>
<td>148</td>
<td>7722</td>
<td>22520</td>
<td>NA</td>
</tr>
<tr>
<td>4</td>
<td>160</td>
<td>7846</td>
<td>22683</td>
<td>NA</td>
</tr>
<tr>
<td>5</td>
<td>192</td>
<td>7643</td>
<td>21194</td>
<td>NA</td>
</tr>
<tr>
<td>Average</td>
<td>168.2</td>
<td>7838</td>
<td>21856.4</td>
<td>207155</td>
</tr>
</tbody>
</table>
Appendix D: ASP.NET that uses XmlTextWriter with data from a database

```vbnet
<%@ Page Language="VB" Debug="true" %>
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Data.Odbc" %>
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="System.Text" %>
<%@ Import Namespace="System.IO" %>
<%
Dim conv As New UTF8Encoding()
Dim tableName As String = "aa"
Dim MyConString As String = "DSN=*******"
Dim MyConn1 As New OdbcConnection(MyConString)
MyConn1.Open()
Dim MyCommand As New OdbcCommand()
MyCommand.Connection = MyConn1
MyCommand.CommandText = "SELECT * FROM " & tableName
Dim MyDataReader As OdbcDataReader = MyCommand.ExecuteReader
While MyDataReader.Read()
    Dim id As String = CStr(MyDataReader("id"))
    Dim node As String = CStr(MyDataReader("node"))
    Dim par As Integer = CStr(MyDataReader("parent"))
    Dim typ As String = CStr(MyDataReader("type"))
    Dim ter As String = CStr(MyDataReader("terminal"))
    Dim sta As String = CStr(MyDataReader("status"))
    Dim note As String = CStr(MyDataReader("notes"))
    'Create instance of XmlTextWriter as a tool for writing our XML information
Dim xml = New XmlTextWriter("c:\temp\" & tableName & ".xml", Encoding.UTF8)
xml.Formatting = Formatting.Indented
xml.Indentation = 4
xml.WriteStartDocument()
xml.WriteStartElement(tableName)
    If par = 0 Then
        xml.WriteStartElement("aa" & id)
    'xml.WriteAttributeString("id", id) -- This may be used for attributes
        xml.WriteStartElement("id", id)
        xml.WriteElementString("node", node)
        xml.WriteElementString("parent", par)
        xml.WriteElementString("type", typ)
        xml.WriteElementString("terminal", ter)
        xml.WriteElementString("notes", note)
    End If
    xml.WriteEndElement()
    xml.WriteEndElement()
End While
MyConn1.Close()
<%>
```