

Florida Institute of Technology

Scholarship Repository @ Florida Tech

Theses and Dissertations

5-2004

Web-based Trend Analysis of Rocket Data Using XML

Mark Randall Gibson

Follow this and additional works at: <https://repository.fit.edu/etd>



Part of the [Computer Sciences Commons](#)

Web-based Trend Analysis of Rocket Data Using XML

by

Mark Randall Gibson

A thesis submitted to Florida Institute of Technology in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

Melbourne, Florida
May, 2004

©Copyright 2004 Mark Randall Gibson
All Rights Reserved

The author grants permission to make single copies _____

We the undersigned committee hereby recommend
the attached document be accepted as fulfilling in
part the requirements for the degree of
Master of Science in Computer Science

Web-based Trend Analysis of Rocket Data Using XML

by

Mark Randall Gibson

Rhoda Baggs Koss, Ph.D., Major Advisor
Assistant Professor of Computer Sciences
Spaceport Graduate Center
School of Extended Graduate Studies

David Clay
Assistant Professor of Computer Sciences
Spaceport Graduate Center
School of Extended Graduate Studies

John Barranti, Ph.D.
Director Administrative Services
Spaceport Graduate Center
School of Extended Graduate Studies

William D. Shoaff, Ph.D.
Assistant Professor and Department Head
Computer Science

Abstract

Web-based Trend Analysis of Rocket Data Using XML

Author

Mark Randall Gibson

Principal Advisor

Rhoda Baggs Koss, Ph.D.

This paper will discuss and prove the feasibility of using XML to transfer, compare, and view data created during automated testing of black boxes; stored both in simple text format and relational database format. This system arises from the need for the reliability department to review the test data from the flight hardware at all testing locations. For the purposes of this paper we will refer to this system as the “Test Data Retrieval System” (TDRS). This data review is being done to determine if there are any negative trends in the hardware that may be surfacing. This data resides in two different filing systems and must be displayed together at the same time to be of much value.

In this study a “system” will be presented consisting of a web-server and several web pages containing “PHP” scripting to interface with a "MYSQL" database and text files to extract test data. The text files contain the test data of the hardware (black box's) acceptance testing done at the vendor and at the final assembly plant. This web server based system uses Apache as the web server and

contains the add-ins for XML and PHP. The XML formatted data is being used to be portable to other platforms other than Windows. The PHP scripting language is employed to aid in the use of transmitting the XML formatted data and makes dynamic web page generation relatively easy, and is also portable to other platforms other than Windows.

One of the main benefits of the proposed system (with the implementation of PHP) is that it provides a seamless integration of popular “open-source” software applications to achieve the desired results. Another benefit is that “basic” code is readily available from many different sources and requires little modification to adapt to successfully meet the requirements of this endeavor.

Table of Contents

TABLE OF CONTENTS	v
LIST OF FIGURES	viii
LIST OF ABBREVIATIONS	ix
LIST OF ABBREVIATIONS (CONTINUED)	x
ACKNOWLEDGMENT	xi
DEDICATION	xii
1.0 INTRODUCTION	1
1.1 TERMINOLOGY	1
1.2 SCOPE.....	1
1.3 METHODOLOGY	2
1.4 BACKGROUND	4
1.4.1 Overall Purpose of the System.....	4
1.5 TEST SYSTEM SETUP USED TO GENERATE THE DATA	6
1.6 LITERATURE SURVEY	6
1.6.1 Harvard University Study.....	7
1.6.2 The affect of XML on the Software documentation process.....	8
1.6.3 Conclusions drawn from this research	9
1.7 PROBLEM DEFINITION	10
1.8 SYSTEM REQUIREMENTS.....	10
1.9 PROPOSED SYSTEM IN GENERAL TERMS.....	12
2.0 SYSTEM CONFIGURATION	14
2.1 COMPUTER CONFIGURATION.....	14
2.2 SOFTWARE COMPONENTS USED ON THIS PROJECT.....	14
2.2.1 Apache: The Web Server.....	14
2.2.2 PHP: The scripting language	15
2.2.3 XML the data transfer language.....	15
2.2.4 MySQL: The Database.....	15
2.2.5 Internet Explorer: The Web Browser.....	15
2.2.6 The web pages.....	16
3.0 DISCUSSION OF THE DATA AND THE USE OF XML	17
3.1 NATURE OF THE DATA	17
3.2 THE FLAT FILE (ASCII TEXT)	18
3.3 THE VENDOR DATA - DATABASE SPECIFICATIONS.....	19
3.4 DATA INTEGRITY	20
3.5 WHAT IS XML?.....	20
3.6 BASIC INTRODUCTION TO XML.....	21

3.7 WHY USE XML?.....	24
3.8 PRINCIPLES OF XML DESIGN	26
3.8.1 <i>Design Methodologies</i>	26
3.8.2 <i>Chief Engineer Paradigm</i>	28
3.8.3 <i>Facilitated Team Paradigm</i>	28
3.8.4 <i>Informed Partnership Paradigm</i>	29
3.9 DISCUSSION OF THE DTD USED FOR THIS PROJECT	30
3.9.1 <i>DTD building blocks</i>	30
3.9.2 <i>The Rocket Data system DTD</i>	32
4.0 WEB SERVER DISCUSSION	34
4.1 WEB SERVERS	34
4.2 THE APACHE WEB SERVER: INTRODUCTION	34
4.3 THE APACHE WEB SERVER: A BRIEF HISTORY	35
4.4 THE LOGICAL CHOICE: THE APACHE WEB SERVER.....	36
4.5 WEB SERVER LOG FILE	37
4.6 SHORTCOMINGS OF USING THE APACHE WEB SERVER	38
5.0 PHP SCRIPTING LANGUAGE.....	39
5.1 JOB CONTROL, INTERPRETED, AND SCRIPTING LANGUAGES	39
5.2 PHP (A BRIEF HISTORY).....	42
5.3 PHP ANOTHER LOGICAL CHOICE.....	43
5.3.1 <i>PHP: Functionally Speaking</i>	44
5.3.2 <i>PHP: Desirable Characteristics</i>	45
5.3.1 <i>PHP: Sample Code</i>	48
5.4 SHORTCOMINGS OF USING PHP	49
6.0 THE RELATIONAL DATABASE SYSTEM PROGRAM MYSQL	50
6.1 WHAT IS MYSQL?	50
6.2 WHY USE MYSQL?	50
6.3 HOW PHP INTERFACES WITH DATABASES.....	51
7.0 SYSTEM ARCHITECTURE.....	52
7.1 MORE DETAILS ON THE IMPLEMENTATION	52
7.1.1 <i>How XML appears in Internet Explorer</i>	52
7.1.2 <i>Displaying the data back to the user, Style sheet or Parser?</i>	53
7.1.3 <i>The XML parser</i>	53
7.2 DISPLAYING THE DATA/USER INTERFACE CONCEPTS.....	58
7.2.1 <i>The end user</i>	58
7.2.2 <i>Web pages needed to do the job</i>	58
7.2.3 <i>Brief description of the web pages:</i>	60
8.0 RESULTS.....	62
8.1 RESULTS	62
8.1.1 <i>Final Output of the data request</i>	63
8.1.2 <i>Validating the XML data file</i>	63
9.0 INTERPRETATION OF FINDINGS / FUTURE WORK	66
9.1 INTERPRETATION OF FINDINGS	66

9.2 THE PHP SOLUTION.....	67
9.3 XML THE DATA STANDARD.....	67
9.4 FUTURE WORK / SUGGESTIONS FOR FURTHER STUDY.....	68
LIST OF REFERENCES.....	70
APPENDIX ‘A’ THE FLAT FILE OUTPUT FORMAT	74
APPENDIX ‘B’ DATABASE FIELDS.....	76
APPENDIX ‘C’ XML DEVELOPMENT PROCESS STAGES	77
APPENDIX ‘D’ EXAMPLE ERROR FILE OUTPUT.....	78
APPENDIX ‘E’ SAMPLE WEB PAGES WITH PHP SCRIPTS.....	79
APPENDIX ‘F’ IMPLEMENTATION HURDLES.....	84
IMPLEMENTATION HURDLES.....	84
APPENDIX ‘G’ THE PERL XML VALIDATION SCRIPT ERROR MESSAGES.....	86
APPENDIX ‘H’ APPLICATION INSTRUCTIONS	107
H.1 INSTALLING THE APPLICATIONS	107
<i>H.1.1 Apache Web Server.....</i>	<i>107</i>
<i>H.1.2 Installing the PHP interpreter</i>	<i>109</i>
<i>H.1.3 Installing MySQL.....</i>	<i>110</i>
H.2 TESTING THE INSTALLATIONS	111
<i>H.2.1 Apache: The Web Server.....</i>	<i>111</i>
<i>H.2.2 The PHP Interpreter</i>	<i>112</i>
<i>H.2.3 The MySQL database.....</i>	<i>113</i>
H.3 PHP FOR THE WINDOWS PLATFORM PC	113
<i>H.3.1 Configuration issues with PHP installations on Windows PC’s.....</i>	<i>113</i>
<i>H.3.2 Open Source or Binaries.....</i>	<i>114</i>
<i>H.3.3 Installation situation overall.....</i>	<i>114</i>
H.4 PHP/MYSQL SUPPORT VALIDATED.....	116

List of Figures

1. System mock up.....	3
2. Test System Setup (at the end user).....	6
3. XML displayed in Internet Explorer.....	54
4. Web pages required	59
5. XML converted to HTML	63
6. XML validated message	65
7. Web server error file	78
8. PHP CGI Script call screen.....	109
9. Apache Web Server setup correctly screen.....	111
10. PHP information page.....	112
11. MySQL installed properly.....	116

List of Abbreviations

ANSI	American National Standards Institute
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASP	Active Server Page
awk	Alfred Aho, Peter Weinberger, and Brian Kernighan
CGI	Common Gateway Interface
CLI	Command Line Interface
COBOL	Common Business Oriented Language
CSS	Cascading Style Sheets
DLL	Dynamic Link Library
DOS	Disk Operating System
DTD	Document Type Definition
faq	frequently asked questions
FTP	File Transfer Protocol
GIF	Graphics Interchange Format
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IE	Internet Explorer
IIS	Internet Information Sever
IMAP	Internet Message Access Protocol
JCL	Job Control Language
JPEG	Joint Photographic Experts Group
MFC	Microsoft Foundation Class
NCSA	National Center for Supercomputing Applications
ODBC	Open Database Connectivity
OOP	Object Oriented Programming
OS	Operating System
PC	Personal Computer
Perl	Practical Extraction and Reporting Language
PHP	Hypertext Pre-Processor
PHP/FI	Personal Home Page / Forms Interpreter
PHP-GTK	PHP Graphical Tool Kit
PING	"to get the attention of" or "to check for the presence of" another party online.
POP3	Post Office Protocol 3
RAM	Random Access Memory

List of Abbreviations (continued)

SAPI	Server API
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TDRS	Test Data Retrieval System
TLS	Transport Layer Security
UUT	Unit Under Test
VAX	Virtual Address eXtension
WML	Wireless Markup Language
W3C	World Wide Web Consortium
XHTML	eXtensible Hypertext Markup Language
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformations

Acknowledgment

I want to acknowledge my wife and son for their continued support to this project. I also want to acknowledge my thesis advisor Dr. Koss and Mr. Clay without their invaluable assistance I would not have gotten this project off the ground. I also want to thank some coworkers (they know who they are) that wanted to stay anonymous.

Dedication

I want to dedicate this to my parents and my wife's parents as I know they wanted me to continue my education always.

1.0 Introduction

1.1 Terminology

The terms used in this paper should be well known to the reader as general knowledge. If the term is not known to the reader, they can refer to the page number xi, the Abbreviations page of this document for the definition. The reader may also refer to one of several web pages that describe these terms in more detail [15].

1.2 Scope

The scope of this paper will be to explain a system designed to extract data from a flat file from the user's site, and data base information/data from the vendor site. The system will also contain utilities to create/add data base information from an eXtensible Markup Language (XML) data file. The information will be gathered, manipulated, and displayed on the user's monitor in a window (web browser). The system will employ the use of different off the shelf, mostly open source software programs, and a data exchange specification (XML).

The major components of this system are:

- **XML -- a World Wide Web Consortium (W3C) specification on data exchange**
- **Hypertext Pre-Processor (PHP) -- a web scripting language**
- **MySQL -- a database program**
- **Internet Explorer -- The web browser**

1.3 Methodology

Refer to figure 1, which is the system mockup.

The author has designed the system, built to retrieve that data from either and/or both locations based on what the user inputs.

TD: A MySQL DB

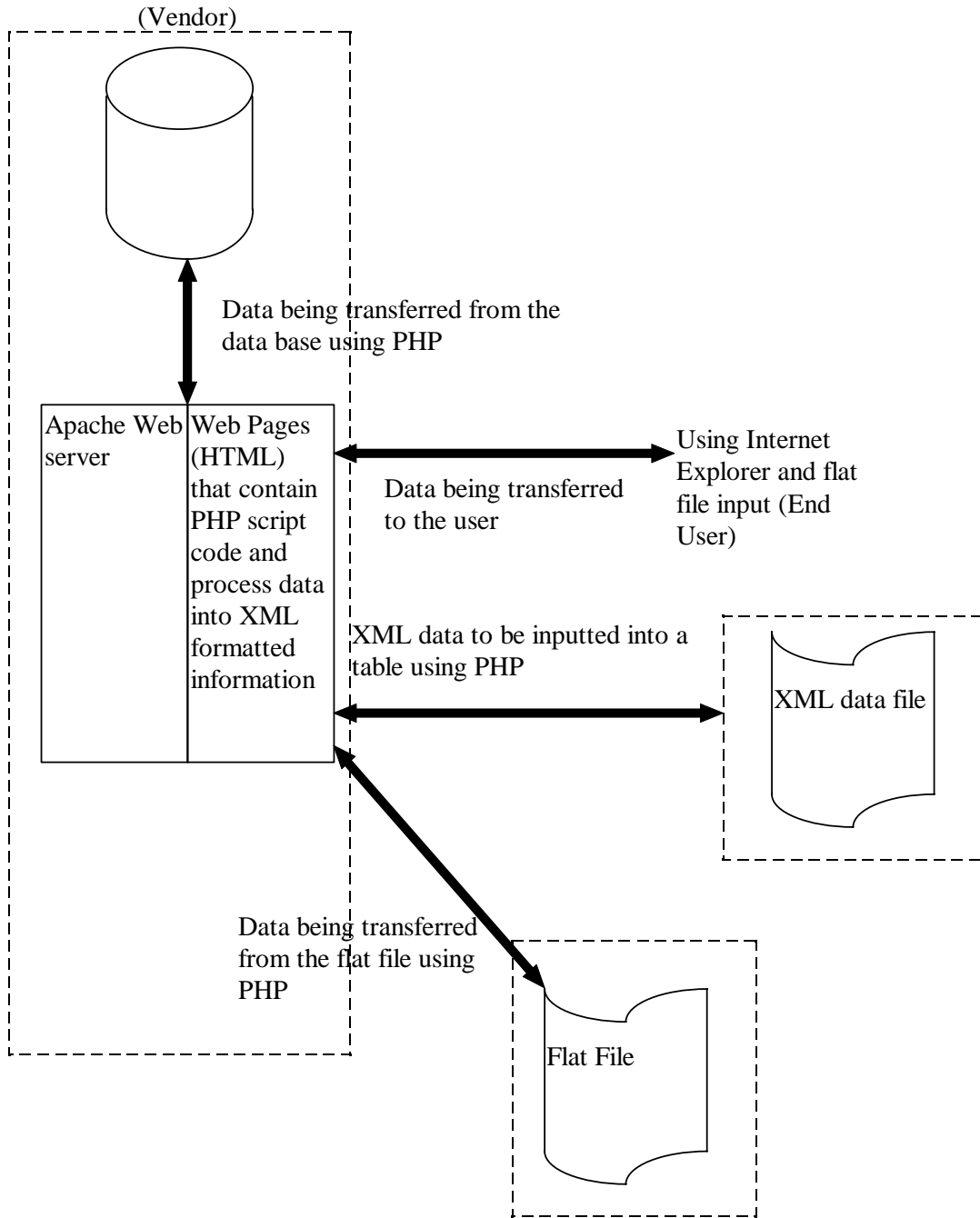


Figure 1 System mock up

1.4 Background

1.4.1 Overall Purpose of the System

The overall purpose of this system is to review past test history data from the flight hardware used on the company's booster rockets. This hardware will also be referred to as "black boxes" throughout this document.

This system allows Test Engineers to determine if there are any negative trends surfacing in the hardware. The data generated when the vendor runs the same tests on black boxes is analogous to the test data generated on black boxes at the end user's facility. The end user stores their data in a text file and the vendor uses a database. This data (at both locations) is compiled from acceptance testing to ensure that the black box is functioning properly prior to shipping it to the end user or for the installation into the rocket. The black boxes are quite old and will not be replaced in the near future due to monetary decisions beyond the scope of this paper. There is a request from the quality department to review the past test history and to review that data to see if there are any negative trends developing. Are the voltages decreasing towards a lower limit? Are the timing measurements increasing in duration?

The data generated by testing the black box at the vendor is extracted from the test computer which is presently a very old mini-computer. The vendor captures the data from the black box testing and transfers it into a database in order to review the data in a more timely manner.

The testing system used at the end user's site is also an old mini-computer (uVAX) and the file of testing data generated is stored in a flat file. The data is then transferred to a floppy disk.

Due to monetary constraints placed on the vendor and end user, a system needs to be built to meet the needs of the reliability department and the system requirements defined in section 1.8 of this document.

1.5 Test System Setup used to generate the data

The test system used to generate this data at the end user is shown below in figure 2. The test set up at the vendor is much the same as in use by the end user, only the equipment in use is much older than the system in use at the end user.

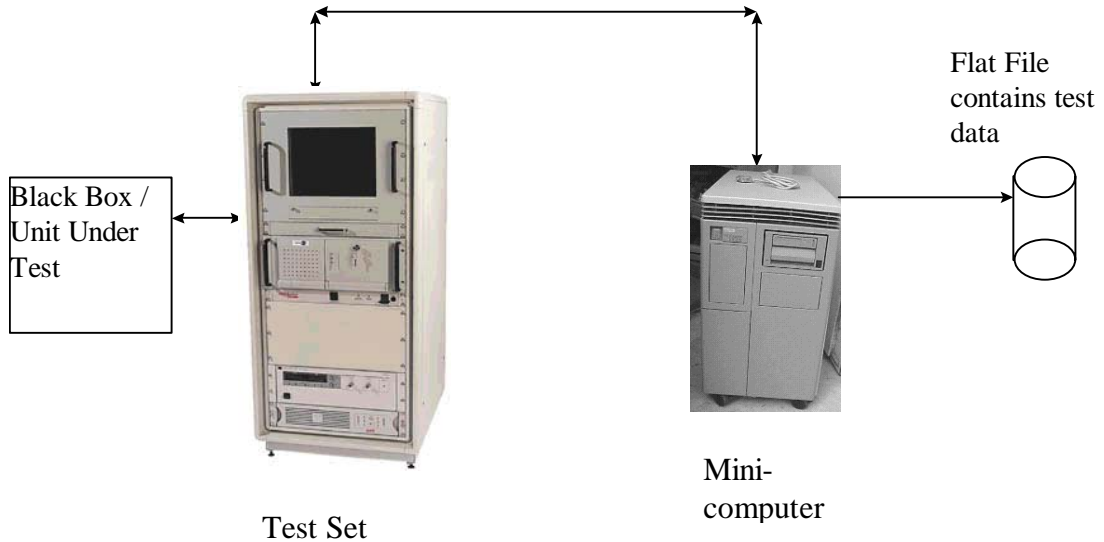


Figure 2 Test System Setup (at the end user)

Note: Not the actual equipment in use at the test sites.

1.6 Literature Survey

The research effort documented herein is the result of an extensive literature search and study of similar types of systems. Therefore the main focus of this paper is not only the primary components of the system but also the glue used to hold the system

together. This in turn will illustrate the nature of software interconnectivity using XML in a web-based Data Server.

1.6.1 Harvard University Study

There is a Harvard University Library Office for Information Systems study that looks at the feasibility of sending magazine articles all over the world. Of primary concern for research libraries and publishers of journals is long term storage and retrieval of information published in said journals. At the time of the writing of the article (2001) many of the journal publishers created their own Standard Generalized Markup Language (SGML) archives using their own proprietary Data Type Definition (DTDs). This meant that each individual publisher's needs were met, however not the technical community as a whole. The article compared SGML and XML and in the conclusions of this paper it was stated that the task/process should use XML and not SGML. Of primary consideration for this conclusion was the fact that there are more tools available for use with XML. It went on further to state that Hypertext Transfer Protocol (HTML) was too simple to perform the need for archival storage [11].

1.6.2 The affect of XML on the Software documentation process

Another related research paper of great interest was “Extensible Markup Language: How might it alter the Software documentation process and the role of the technical communicator?” [13]. This article proposed that there are nine (9) components of the XML Development Process grouped into three (3) stages see below:

XML Development Process

Stage 1 Structuring content with:

- **Document Type Definitions**
- **XML Schema**
- **XML Content Files**

Stage 2 Formatting content with:

- **XSL Transformations (XSLT)**
- **Cascading Style Sheets (CSS)**
- **Extensible Stylesheet language (XSL)**

Stage 3 Linking and manipulating content with:

- **XML Link Language (XLink)**
- **XML Pointer Language (XPoint)**
- **XML Path Language (XPath)**

This paper explained the differences between DTDs and Schemas. DTDs and Schemas are both specifications that are used to define how the markup (embedded codes) are to be processed. Schemas are much more powerful and flexible than are DTDs. They possess three features not found in DTDs: First, Schemas are written entirely in a syntax compatible with XML, whereas DTDs are based on the 20 year

old SGML standards as a basis. Secondly, DTDs only have two data types which greatly reduces their flexibility in transferring data between computers. In contrast Schemas have at least 45 data types and the ability to develop/create complex data types on their own. Lastly DTD's are not able to deal effectively with namespaces, and this is very inhibitive. Namespaces are used in XML in an effort to take out the ambiguity of common names. The XML namespace is a collection of elements and their attributes uniquely defined in that namespace. Therefore an XML document can contain elements and attributes with two part names, one for the namespace and the other the local name.

1.6.3 Conclusions drawn from this research

All this research suggests that XML is here to stay yet it is not the end all in the world of data transfer. There are many issues that still need to be resolved, some are managerial and some are technical. One major issue is semantics, for example how does one define "price"? Dollars, or pounds, etc. [21]?

Another major issue that needs to be resolved is the number of standards guiding the continued use of XML. There are several that govern the world of Finance alone [21]. Even with these major issues essentially unresolved, XML continues to gain acceptance in the world of data transfer. The author believes that these issues can and will be resolved, and the data transmission paradigm will gain widespread use.

1.7 Problem Definition

There is measurement (test) data being created during the automated testing for final acceptance of black boxes being supplied to us by our vendors; and these tests are being performed at several testing locations. The measurement data generated is being stored in simple text format and in relational database format.

The reliability department must now review the test data from the flight hardware of the black boxes at all testing locations. This data review is being done to determine if there are any negative trends in the hardware that may be surfacing. The hardware is getting very near the end of its designed life but not the mission life. The data must be displayed together at the same time for the sake of completeness of the trending analysis.

In the following chapters the author will describe the system and its components.

1.8 System Requirements

The “system” defined herein and henceforth referred to as the “Test Data Retrieval System” (TDRS) must meet the following requirements:

- ◇ Must initially be Windows based but maintain the capability to be ported to other operating systems later if the need arises. The Platform Migration project is getting a lot of attention at all levels of management

and this project will be held to any decisions/recommendations/findings made therein. (Whole project)

- ◇ Must be a web based solution. The company direction is away from the Integrated Development Environment (IDE) systems from Microsoft, due to the enforcement of the Microsoft Foundation Class (MFC) principle. The design/development time should be greatly reduced as the fundamental concepts for the transfer of data is already done for us.(Apache Server)
- ◇ The maintainability, portability, and readily available “components” concept is very enticing to upper management and they are eager to travel down this road. (Whole project)
- ◇ We have a whole cadre of Software Engineers that are well seasoned ‘C’ programmers that have a lot of corporate knowledge that we want to keep employed and busy. (PHP)
- ◇ Must be able to access and process the data stored at the vendor. The data is in a relational database format. The vendor has had their system in place for many years and it is not cost effective for them to change now. Also at this time there is already several years worth of past test data that requires review. (PHP)
- ◇ Must be able to access and process the data stored at the user’s testing location. The end user has had their system in place for many years and it is not cost effective for them to change now. (PHP)
- ◇ Within the organization there are a number of data exchange projects being worked in concert with many of our vendors. Therefore the data and format of the data needs to be well defined, and it fits to use state of the art data transmission protocols, methods, and theologies.

- ◇ The information processed must be transmitted and stored (if desired) in a format such that the exchange of the data will be able to cross systems and platforms. (XML)

1.9 Proposed System in general terms

In general terms the “system” (TDRS) will be made up of mostly “off the shelf”, open source products which are readily available to anyone who has an Internet connection and knows where to look. The core of the system is the web server which is from the Apache Organization. The module that runs along with the web server will be a scripting interpreter. The scripting language is PHP and the interpreter is from The PHP Group. The data stored to simulate the vendor data storage is in the database software known as MySQL from the MySQL AB company. This software is free for the usage described herein as the author obtained it for student usage. The data transferred will be in the format/protocol known as XML. The web pages needed to perform the tasks will be a combination of HTML code and PHP scripts.

The components used herein in the system are designed (used) to meet the requirements of being portable and maintainable. It is also web based and uses the data formatting language of XML that is an emerging technology/theology.

In the discussions in the following chapters we will explore each element/component of the system design/concept in as outlined below:

- ◇ Chapter 3 XML the data formatting language
- ◇ Chapter 4 The Apache web server
- ◇ Chapter 5 PHP the scripting interpreter
- ◇ Chapter 6 MySQL the database application

2.0 System configuration

2.1 Computer configuration

The computer system used to do this study was an Intel Pentium II processor 266 MHz, with 320 Megabytes of Random Access Memory (RAM) running Windows Me.

2.2 Software Components used on this project

2.2.1 Apache: The Web Server

The Apache version level for this project is 1.3.27, and was found at the web site “<http://httpd.apache.org/download.cgi>”. The documentation at the Apache web site said that Apache web server version 2.0 (and beyond), is better suited for Windows NT or Windows 2003 [27].

In the time since the author started this project the particular version used on this project has been moved. It has been moved to the older versions web page: “<http://archive.apache.org/dist/httpd/old/>”.

2.2.2 PHP: The scripting language

The version of PHP used on this project is 4.3.2, found at the following web site: “<http://www.php.net/downloads.php>”. The version of PHP used on this project suffers from the same problem that the author had with the web server. This particular version of PHP is no longer supported.

2.2.3 XML the data transfer language

XML, the extensible Markup language is by design conducive to creating common data formats that share both the data and the format for distribution on the World Wide Web. The files and data generated for this project conform to XML version 1.0.

2.2.4 MySQL: The Database

The version of MySQL used on this project is 4.0.15 and was downloaded from “<http://www.mysql.com/get/Downloads/MySQL-4.0/mysql-4.0.15-win.zip/from/pick#mirrors>”.

2.2.5 Internet Explorer: The Web Browser

The web browser is provided by Microsoft©, Internet Explorer version 6.0.2800.1106IS, the information was found on the help/about screen. This is a customized version of Internet Explorer.

2.2.6 The web pages

The web pages were generated mostly using the Notepad application. They could have just as easily been generated using Word or Wordpad. As it was stated earlier the web pages are a combination of HTML code and PHP scripts.

3.0 Discussion of the data and the use of XML

3.1 Nature of the data

The data contained in both the ASCII (American Standard Code for Information Interchange) text file (flat file) and the data base is simulated test data taken from the acceptance tests run on the black box our company uses during manufacturing operations. The test data is from actual measurements made while the black box is connected to the test set, and exercised by the control computer. The test station provides various stimuli to the black box and takes specific readings based on the stimuli being applied at that time. These readings are then compared to the limits in the software and recorded in the test data file. The readings taken may be voltage, current, or resistance (including but not limited to continuity and isolation) in nature. There are some instances where timing measurements are made. This test data is referenced by a specific test number that uniquely defines the stimuli present at the time of the measurement. The files also contain the test date and the black box serial number.

The tests being performed check to ensure that the black box is ready to support flight operations (or to be shipped to the end user). At the present time there is no method (or system) to compare measurement values made “today” to/against what the value of that reading was the last time the test was performed. This is what the Quality/Reliability office has been tasked to perform, again looking for negative trends that would indicate that there may be a problem with a specific black box. This system when complete will aid the Reliability (or Test) Engineer in identifying negative trends in certain measurements or types of measurements now matter when

or where or in what format the data was recorded. The test number will be inputted by the operator and all the information stored in both locations will be searched and displayed back to the operator.

3.2 The Flat file (ASCII text)

The flat file starts out by listing the test date, then some test station measurements are recorded, then the name of the test, the “version number” of the black box, and then the serial number. This is followed by more text, which is generally the special number of the “floor paper” used to test the box. The “floor paper” is a work document used by the technicians authorizing them to perform the test. Then the header for the first test is listed in the form of the time at which the test starts, name of the test, and the temperature the test was performed at: ambient, high, or low. Next is the tabular data header and this is followed directly by the actual tabular formed data values taken by the test set.

The data that is needed to be reviewed and is operated on by the web pages has 6 fields from the tabular data, and has the added field of the date for trending purposes and is listed below:

Test Num.	Test Nome.	Conn. points	HIGH Val.	LO Val.	Meas. Val.
3.000010	Test is #10	TP1 to TP2	22.00	0.00	2.35 OHM
3.000020	Test is #20	TP1 to TP3	22.00	0.00	2.06 OHM

Refer to Appendix A The Flat file output format for a more detailed pictorial layout.

After that particular test is over, there is one line that lists the actual test run time of that particular portion of testing, and then the overall test run time. The listing continues by going onto the next test as described above.

When all tests are completed, there is an end of data storage message and the end of program message. See below:

```
10-APR-2003 08:06:16 ELAPSED 00:01:38 TOTAL 00:01:38
```

```
Data Storage Disabled on Station Tester
```

```
END OF PROGRAM "Out-the-door test" DATE: 10-APR-2003 TIME: 10:15:14.67
```

3.3 The Vendor Data - Database Specifications

The author had intended to use Microsoft's ACCESS program to build and maintain the database but decided to use a freeware version of a database application instead. The author found some that are readily available: MySQL and FIREBIRD to name two. Preliminary research was to investigate if either program would do XML right off the bat so the author would not have to do it with the C++ program. The site for MySQL said that there was a working version that would output the answer to the query directly into XML format. However after further investigation it appears that this is for a Java implementation. The author finally chose MySQL as it comes bundled with the version of PHP used in this project.

The table used by the vendor that has the fields as depicted below (and in Appendix B Database Fields):

BB	Test	Test	Test	Conn.	High	LO	Actual.
S/N	Date.	Numb.	Nom.	Points	Val.	Val.	Val.
001	01/01/2001	3.000010	Test is #10	TP1 to TP2	22.00	0.00	2.35 OHM
001	01/01/2001	3.000020	Test is #20	TP1 to TP3	22.00	0.00	2.06 OHM

3.4 Data Integrity

The data being transmitted for this project is not secret in nature. The data is being sent over the Internet and uses the Transmission Control Protocol/Internet Protocol (TCP/IP) and guarantees some level of assurance that the data will reach the end user.

There are a few things that the author could do in the future to make the data transmissions more secure. Late in the development of this project the author learned that there is a “sister” project of the Apache organization. It entails the generation of an Apache-SSL web server. This is based on the Apache web server and SSLeay/OpenSSL. SSLeay is the free implementation of Netscape’s Secure Sockets Layer. Secure Sockets Layer (SSL) is a commonly used protocol for managing the security of transmissions on the Internet [29]. OpenSSL is a project (organization) to develop a full featured, robust, commercial grade “Open Source” toolkit implementing the Secure Sockets Layer and Transport Layer Security (TLS) [30]. This is one aspect the author could employ to make this system more secure in the transmission of the data.

3.5 What is XML?

XML is a formal recommendation from the World Wide Web Consortium (W3C) and is similar in format to many of today's web pages HyperText Markup Language (HTML). XML and (HTML) both make use of markup symbols to describe the data/file. Most of the symbols used in HTML are for formatting and the end user cannot create new ones. However, with XML a basic premise is the ability to create tags that reflect the data stored in between them. XML is a way to share both data and format across the web.

XML is a subset of Standard Generalized Markup Language (SGML) which looks a lot like HTML but of course is much more than that. In layman's terms a "markup language" is basically a way to distinguish what is to be printed from instructions on how it is to be printed. The markup is the instruction portion of the equation. Some examples of markup languages include HTML, eXtensible Hypertext Markup Language (XHTML), and XML [26], to name a few.

SGML is itself a standard on how to specify a document markup language. This specification is in itself a Document Type Definition (DTD); however it is not a language. It is a description of how to describe one, it is also metadata. HTML is also a formal recommendation from the W3C. It is a collection of markup symbols and codes used in conjunction with the data in a file to be displayed on a web page by today's browsers.

3.6 Basic Introduction to XML

One important thing to remember is that by itself XML does not do anything, it is a language for defining containers of information. It was designed to carry, store, and exchange data across systems and platforms if necessary. XML is very versatile and in fact can be used to create other languages, which makes it a meta-language. A case in point is Wireless Markup Language (WML), which is written in XML [12].

XML is much more structured than HTML and also more flexible, yet HTML is a little more forgiving in the syntax area. HTML will allow the user to forget end tags, and tags are not case sensitive, whereas XML will not insert end tags and is case sensitive.

Shown below are some examples of XML:

(Author's XML data file, from this project)

```
<rocket_data1>
  <rocket_data>
    <bb_sn>001</bb_sn>
    <test_date>10-APR-2003</test_date>
    <test_num>3.000430</test_num>
    <test_nomenclature>Test is #430</test_nomenclature>
    <test_connection>TP40 to TP48</test_connection>
    <test_high_value>22.00</test_high_value>
    <test_low_value> 0.00</test_low_value>
    <test_actual_value> 2.4</test_actual_value>
  </rocket_data>
</rocket_data1>
```

Another XML example is a file found on the author's computer from the application named "MusicMatch". This file is used for the configuration of MusicMatch i.e. for settings and plug-in information. The XML file is named "MediaServerDevDesc.xml".


```

        <SCPDURL>/ConnectionManager.xml</SCPDURL>
    </service>
    <service>
        <serviceType>urn:schemas-upnp-
org:service:ContentDirectory:1</serviceType>
        <serviceld>urn:schemas-upnp-
org:serviceId:ContentDirectoryServiceID</serviceld>
        <controlURL>/MediaServer/ContentDirectory/Control</co
ntrolURL>
        <eventSubURL>/MediaServer/ContentDirectory/Event</ev
entSubURL>
        <SCPDURL>/ContentDirectory.xml</SCPDURL>
    </service>
</serviceList>
</deviceList>
</device>
</root>

```

3.7 Why use XML?

XML is easy to use, and is posturing itself to become the de facto standard of the World Wide Web as far as information transfers are concerned. It is platform independent, and a formal recommendation. The basic characteristics of standard XML make it ideal for the transmission of data. For instance, one characteristic is the fact that anyone can define a tag set. Another is that tags that are used in XML documents automatically specify the content [13].

Listed below are some XML features [23]:

- **Ease of use**

Based on the simple fact that the tags are so descriptive, XML documents are inherently easy to read. They are also very easy to create, even for those who may not be very computer literate.

- **Formal Structure**

There are, however formal structure requirements that must be strictly adhered to. Tags have to be named and nested properly; opening tags must have a corresponding closing tag. Imposing these rules ensure that all XML documents meet at least some minimum level of structure and syntax. By meeting these requirements, the document will be what is called “well formed”.

- **Internet friendly**

Designed to be used on the Internet, XML plays two very important roles:

1. It provides a tool kit whereby users can represent a wide variety of data, in large or small amounts flowing across the Internet. This in turn makes for better organization and classification of the information on the web.
2. XML is a standard method for the exchange of information, encoding the data in a format easily transmittable from computer to computer while using the standard Internet protocols already in use.

- **Wide application support**

Because XML is so easy to use and create, it takes very little effort to develop an application that uses XML. There are a number of available parsers available online. There are also XML editors, validators, and other similar tools available to the developer. Most of the popular web browsers support XML too.

- **SGML-compatible**

The fact that XML is a subset of SGML implies that XML documents will comply with the rules and constraints of the SGML markup. However, since the scope of XML is much narrower than SGML, it is far easier to use.

In summary, XML provides a standard data interface to a wide variety of already available products.

3.8 Principles of XML Design

3.8.1 Design Methodologies

When designing the tag set one of the most important decisions to be made is the design methodology to be employed to create the tag set. The choice to be made is heavily influenced by the creation environment because this is where the tags emanate from.

The three main sources are as follows:

- ◇ Software written to extract data from a database.
- ◇ Editors that update preexisting tagged XML data sources.
- ◇ Designers that create their material and tag that data in XML.

If software is used to tag the data, one can design a highly complex tag set and be used with little impact on the staff and schedules because once the program is written it will take care of the tedious task of tagging the data. If one is updating and augmenting the data the number of tags and their corresponding ease of use becomes important. The number of tags and their ease of use really becomes important if one is to “tag” the data as it is being created. If the latter is the case then the usability of the tags is of the utmost importance and the design approach selected will be influenced by the XML source that is the basis of the project.

Now depending on the creative environment in the reader's particular situation one can select from the following XML design paradigms:

- ◇ Chief Engineer -- One person is in charge and responsible for the creation of the design.
- ◇ Facilitated Team -- Group of individuals formed as a project team and assisted by an XML expert to create the design.
- ◇ Informed partnership -- Almost the same as a facilitated team except the XML expert is assisted by the project team.

We will explore the pros and cons on each methodology in the paragraphs below.

3.8.2 Chief Engineer Paradigm

Following this design approach means that one will be most likely employ the services of an XML expert to create the DTD or Schema. The chief engineer could also be from the internal staff. In this approach the chief engineer more or less does it all, gathers the requirements, the data samples, and then develops the tag design to meet those requirements. Then depending on the situation the chief engineer may present the finished product to an authoring staff for review and comments. The chief engineer may also hand off the project to an internal user, technology group, or integration group to use in their system design.

There are a couple of advantages to this method. First if this truly was an expert there should be no beginner type mistakes. Secondly there should be very little impact on the day to day operations of the company resources in the organization.

There are also a couple of disadvantages. First, there is little or no participation by the eventual end users, which would have made the product better. Secondly, there is no guaranteed “buy-in” by the end users which in the long run could be more costly than if the company had spared those resources during the development time [1].

3.8.3 Facilitated Team Paradigm

The Facilitated Team Paradigm is almost the exact opposite approach when compared to the chief engineer approach. A project team is formed and the XML expert serves more as a facilitator to the design process. The team usually consists of technology staff, editorial staff, production staff, trainers, and management

representatives. The facilitator guides the team through the design principles and phases. The facilitator must constantly balance the decisions made by the team with solid XML design practices.

Unlike the chief engineer approach “buy-in” by the end users should not be an issue since they would have been involved every step of the way. This is a distinct advantage to this approach. Another plus is the XML experience gained by all participants that can be later employed on other projects that would involve XML design work.

One can clearly see where this process could be a lengthy one and tie up much of the company’s resources for that period of time. This investment in time may not be worth the effort especially if automated XML tagging is going to be employed in the project[1].

3.8.4 Informed Partnership Paradigm

Following this approach to the design means that the project team is involved throughout the whole process but the team defines the project in general terms. It is up to the XML designer/expert to finish these data objects at a later time. After the initial design meetings the designer completes the document model and creates and validates the XML DTD or Schema. The designer will at the end of the project get the team to approve the design most likely by having a review of the DTD and have the team “sign off”.

Once again like the facilitated team approach “buy-in” for the project should not be an issue. This method balances the cost of staff involvement against the cost of the XML expert, and still maintains the end user involvement during the design phases.

Another trait the informed partnership has in common with the facilitated team is that there are more company resources invested than with the chief engineer approach. This again would be a management decision as to whether the benefit outweighs the costs of employing this method[1].

The paradigm chosen for this project was chief engineer.

3.9 Discussion of the DTD used for this project

For this particular project the author employed the Chief Engineer approach to the XML design of the DTD. The DTD/Schema designed for this project is shown below. This is the DTD that the XML to HTML web page will use when it “validates” the data about to be displayed on a web page, from the PHP script that generated the XML document.

3.9.1 DTD building blocks

In this section we will discuss the basic building blocks of a DTD and point out their usage in the DTD used for this project in the next section.

There are two basic parts of a DTD, first part is the declarations of the elements, entities and so forth. This is referred to as the “document type declaration”. Secondly there consists the documentation and conventions needed to explain how to process your data.

Listed below are the most important declarations used in creating a document type definition with a short description of each:

- **Element declarations**

Statements that define the type of information that will be contained by this element, but not anything about the type. For example you can say that the element is text, but you can not say anything further about what the text will contain. If you need to control the input further this requires an XML schema.

- **General entity declarations**

These are used to define the general boilerplate text information, it can refer to a external resource.

- **Comments**

Notes for human readers of the DTD and do not affect the XML documents that use the DTD.

- **Condition sections**

Used when portions of the DTD are to be used under certain circumstances.

- **Parameter entities**

Used in conjunction with conditional sections to customize the DTD. These can also be code snippets that are reused.

In the next section we will examine the DTD for this project in further detail.

3.9.2 The Rocket Data system DTD

Shown below is the DTD for this project.

```
<?xml version="1.0" standalone="yes"?>
<!-- internal DTD -->
<!DOCTYPE rocket_data1 [
<!ELEMENT rocket_data1 (rocket_data)*>
<!ELEMENT rocket_data (bb_sn, test_date, test_number,
test_nomenclature, test_connection, test_high_value,
test_low_value, test_actual_value)>
    <!ELEMENT bb_sn (#PCDATA)>
    <!ELEMENT test_date (#PCDATA)>
    <!ELEMENT test_number (#PCDATA)>
    <!ELEMENT test_nomenclature (#PCDATA)>
    <!ELEMENT test_connection (#PCDATA)>
    <!ELEMENT test_high_value (#PCDATA)>
    <!ELEMENT test_low_value (#PCDATA)>
    <!ELEMENT test_actual_value (#PCDATA)>
]>
```

We will examine each line of the DTD and discuss its purpose further:

```
<?xml version="1.0" standalone="yes"?>
```

This starts the XML declaration and lists the version of 1.0 which is the only version, standalone refers to the fact that there are no external markup declarations.

```
<!-- Main DTD defining the structure of a presentation -->
```

This is a comment line to explain the purpose of this file.

```
<!DOCTYPE rocket_data1 [
```

!DOCTYPE starts the document type declaration. The '[' starts the internal subset.

<!ELEMENT rocket_data1 (rocket_data)*>

This statement tells the reader that there will be more than one instance of rocket_data.

**<!ELEMENT rocket_data (bb_sn,test_date, test_number,
test_nomenclature, test_connection, test_high_value,
test_low_value, test_actual_value)>**

This statement lists all the elements contained in the root element in the DTD. Each element type include a name, content and also attributes if they apply.

<!ELEMENT test_number (#PCDATA)>

This is the definition of test_number as an element. The '#PCDATA' means that text or character data is allowed. The DTD then defines the rest of the elements that make up the root element.

]>

The markup '[' ends the internal subset. The '>' ends the DTD.

This concludes the discussion on the DTD used herein.

4.0 Web Server Discussion

4.1 Web Servers

A web server is a program that employs the client/server and the World Wide Web's Hypertext Transfer Protocol (HTTP) paradigm to support web page usage. The ones most commonly used are Apache (largest installation base) and Microsoft's Internet Information Server (IIS). The Apache organization states that the server itself was originally designed for the UNIX system and then ported to the Windows environment. Apache should never be used on consumer based operating systems such as Windows 95, 98, or Me (Millennium Edition). There are others not so widely used such as Novell's Web Server and IBM's family of Lotus Domino servers [11]. New versions of the server are coming out all the time.

4.2 The Apache Web Server: Introduction

The "Apache HTTP Server Project" as it is referred to by the Apache organization web site, is an effort by a software development team to develop the best web server possible. Their main goals are to provide a freely available, commercial-grade, robust, full of features, source code implementation of an HTTP (Web) server. Located around the world and communicating over the Internet and the web, a group of volunteers jointly manage this project. These volunteers are referred to as the "Apache Group". There are also a lot of other users that make contributions (ideas, code, and documentation) to make the product better overall.

Apache web server is a flexible, powerful, Hypertext Transfer Protocol (HTTP)/1.1 compliant web server [19]. Implementing the latest protocols, it is highly extensible and compatible with third-party software. Apache can be freely obtained either as source code or compiled into binaries both under an unrestricted license. Its open source nature means that it is constantly being updated and encourages input from users to better the software or to fix it.

4.3 The Apache Web Server: A Brief History

In 1995 there was a public domain HTTP daemon developed by Rob McCool. Rob at the time was employed at the National Center for Supercomputing Applications (NCSA), University of Illinois, Urbana-Champaign. The development stalled when Rob left that job in mid-1994. Webmasters took it upon themselves to develop extensions and fixes (patches) and many of these webmasters thought that these fixes should be shared with the rest of the world. Contacted by private e-mail, a small group of them gathered for the purpose of coordinating those changes. Two gentlemen by the names of Brian Behlendorf and Cliff Skolnick put together a mailing list and obtained accounts for a core of developers in the California Bay Area. The bandwidth for this endeavor was donated by the HotWired Company. A short while later a group of eight core contributors became the “Apache Group”. Their names are:

Brian Behlendorf, Roy T. Fielding, Rob Hartill, David Robinson, Cliff Skolnick, Randy Terbush, Robert S. Thau, and Andrew Wilson. These developers used the NCSA httpd 1.4 as their base platform, and they added all the published bug fixes and all the enhancements they could get their hands on. They made their first official public release of version 0.6.2 of the Apache Server in April 1995. After a

few iterations in between Apache 1.0 was released on December 1st, 1995. In less than one year after the group was formed the Apache server was now more popular than the NCSA's HTTPD server. It was now the #1 server on the Internet. According to a survey by Netcraft it remains so even today [20]. Another interesting factoid is how the name was derived, it comes from "A PAtCHy" Server and is a cute name that just stuck.

4.4 The Logical Choice: The Apache Web Server

The author chose to implement this system on a web server for a few reasons. First many applications and techniques are migrating to web based processing. Second web servers make ideal data servers as is being illustrated *no matter what the format*. Still another reason is that almost everyone is comfortable using a browser.

The Apache web server was chosen because it is open source and more or less free to obtain in contrast to Microsoft's IIS. Apache is also the most widely installed web server on the Internet. As of February 2000 there were over 6 million Internet servers running Apache software. All versions are thoroughly tested by users and developers alike. The Apache Group maintains very rigorous standards about releasing new versions to the server software.

According to the Netcraft (www.netcraft.com) Web server survey in February, 2001, 60% of all Web sites on the Internet are using Apache (62% including Apache derivatives), making Apache more widely used than all other Web servers combined. Another survey conducted in July, 2003, found that now 63% of the web servers on the net are Apache [20].

4.5 Web Server Log File

Within the system on the vendor side the errors generated will be displayed in the web server error log file. This will be monitored by the system operator and the error will contain the offending page by name. Some errors that could be encountered will be displayed on the web page itself.

Examples of errors caught by the web pages are mostly data entry type errors and when displayed in real time are:

- if the file can not be opened or created; (will be displayed to the user right then and there).
- if the end date is before the start date
- if the test end number is before the test start number

An example of the error log entries can be seen in Appendix D Sample Error File Output.

4.6 Shortcomings of using the Apache Web Server

There is a notice on the Apache web site that the version chosen for this project is not recommended for the production environment. The next released version that was recommended for production usage was not chosen for this project because it was recommended for the Windows 2000 platform.

Additionally, since Apache was not purchased, there is no support group obligated to help with any problems encountered during the process. There are however a plethora of user groups (Usenet) web sites, books, and colleagues for help and consolation if the need arises. Add to this an electronic weekly publication called the “*Apache Weekly*” and the few companies that offer paid support for this software [12], and one begins to appreciate the nature of the Open Source Movement.

On the Apache web site there is a forum for the user to enter a new bug report if desired. A quick check of open bugs yielded a very short list indeed.

5.0 PHP Scripting Language

5.1 Job Control, Interpreted, and Scripting Languages

Scripting languages have “evolved” (maybe even mutated) over the years. In some aspects it appears to the author that the proliferation of the web has brought about a resurgence in this activity.

Conventional wisdom says in order to make the best use of the computer you must make the most of its time (or your time on the computer). Early on this meant writing tight, extremely efficient code in machine code or assembly, which in turn meant the programmer had to direct the computer’s actions in a very detailed manner. However this was highly target specific and not very portable to other platforms, not to mention very tedious.

Then came the advent of “compiled” languages, such as Common Business Oriented Language (COBOL) and C. Compilers translate source code into machine language for a specific computer. This freed up the programmer to concentrate their efforts more on the algorithms and such, rather than the tedious machine dependent details of computer direction. These compilers were designed and modified to approach the same level of speed obtained by the hand generated assembly code. Because these compilers have done their job so well they have become the tool of choice of programmers no matter what the task is.

In order to function properly applications written in assembly language or “compiled” languages need to be “introduced” to the operating system. Job control

languages such as IBM's Operating System (OS) Job Control Language (JCL) and Disk Operating System (DOS) in a batch processing environment perform this function. This "introduction" however was more or less the use of variables, invocation of programs, and the making I/O connections.

Command languages like the ones found on early micro computers were simpler to use, but basically had looping capabilities in addition to their primitive flow control features. Contrast this with UNIX shells which had programming features that were well developed, these included function, arithmetic operations, and simple data structures.

The UNIX community developed hundreds of user commands to augment the ever increasingly powerful shells. Some of these are reasonably complete programming languages in and of themselves. "Shell scripts" are a very powerful combination of these user commands and the shells, and can accomplish a lot with just a little code.

Interpreted languages are performing more and more of the programming tasks of late. The source code is "interpreted" on the fly (more or less) rather than compiled. There is some loss in speed of execution when dealing with interpreted code. However with the speed of today's computers most programmers are willing to take the speed penalty in the trade for greater flexibility. The cost in speed may anywhere from 3 to 10 times, and in most cases this is probably an acceptable trade off.

There are other benefits to be gained also, in the areas of maintenance and development, as the code needed to do the job can be substantially less than compiled code. A lot of this gain in the area of development, comes from gluing preexisting code (this author sometimes refers to as code snippets) together.

On the Apple platforms, an AppleScript is an interpreted piece of code that commands the Mac OS to perform desired functions. The definition of a scripting language is a little less clear on the UNIX system, where languages that are optimized for ease of use and expressive power are included in the definition.

Scripting languages tend to have the following characteristics:

- Processor and OS independence - able to work in any target environment
- Late binding - data types resolved as necessary
- Runtime evaluation of code - import source code at runtime and execute it
- Various built-in, high-level features - features hard to write from scratch
- Less “administrative overhead” - simple assumptions about data types
- Optimization for programmer efficiency - ability to glue snippets together
- Optimization for certain types of applications - data filtering, test manipulation etc.
- Rapid prototyping - up and running in record time! [31]

Although there may be no official definition of what a scripting language is, simply put a scripting language is a language designed to write small but very powerful programs to perform a certain task [23]. Depending on your definition scripting languages might be the highest level of programming language at this time.

5.2 PHP (a brief history)

Perl scripts were written to serve as the early predecessor of PHP. Perl was written in 1986 by Larry Wall, a UNIX programmer who needed a program to do a certain task. He started with awk and quickly determined it would not meet his needs. He had a choice now, write a utility to perform the task and probably very soon he would have to write another utility to perform a different task. So he decided not waste any time on utilities, and just invented a new language. Perl was written in 'C' and retains some of its syntactic conventions [32].

The early predecessor of PHP was Personal Home Page / Forms Interpreter (PHP/FI). PHP/FI was designed/written by Ramsus Lerdorf in 1995. It started out as a set of simple Perl scripts to keep track of accesses to his resume, which was on-line.

Initially he named these scripts as "Personal Home Page Tools". He thought, "Now of course once you have something neat like this you have to keep adding to it." It needed more functionality so he wrote a much larger capability in C including database interfaces. This in turn allowed users to build/design develop simple dynamic web applications. In the early days PHP/FI had many of the capabilities of the PHP we know today including Perl like variables, automatic interpretation of

variables, and an embedded HTML like syntax. This syntax was much like Perl only much more limited, simple, and inconsistent. The second write-up of the C based implementation called PHP/FI 2.0 had a cult following of several thousand users by 1997. Now at this time with only about 50,000 domains claiming to be using it, it accounted for about 1% of domains on the Internet at that time. PHP/FI 2.0 was officially released in November 1997.

PHP development continued and was performed by Andi Gutmans and Zeev Suraski as a complete rewrite known as PHP 3.0. It closely resembles the version we have today. They started on this project after they found PHP/FI 2.0 severely underpowered for eCommerce applications, while working on a University project. Gutmans, Lerdorf, and Suraski decided to collaborate with each other and to formally announce the release of PHP 3.0 as the official successor to PHP/FI 2.0. It was now released under a whole new name (a recursive acronym), ‘PHP’: Hypertext PreProcessor. The version of PHP 3.0 was released in June, 1998 having spent about nine months in the public beta testing arena.

Not long after PHP 3.0 was released, Gutmans and Suraski started on a rewrite of the PHP core. They termed the core the “Zend” engine derived from a combination of their first names, which was first introduced in mid 1999. PHP 4.0 was based on this new engine and it was officially released in May 2000.

5.3 PHP Another Logical Choice

The fact that PHP is “open source” means that no company or business “owns” the software or copyrights and no one company can extort more money from the

customer to use it. Another very useful trait is that one can write PHP enabled web pages using a plain old text editor. One of the most important features is the ability to manipulate database information with several databases including MySQL. Like XML it too can be used cross-platforms [5].

In the web based programming paradigm PHP is a scripting and parsing language that is primarily used on UNIX based machines. PHP was previously referred to as “Personal Home Page” but now means “Hypertext Pre-Processor”. PHP is a simple, powerful but very popular scripting language that looks a lot like C/C++, used for generating HTML content. It is commonly used as the “glue” to perform the background tasks needed when the user clicks on, or fills in blanks on a web page. To state this another way, PHP can be used in the creation of “Dynamic Web pages”. It is a welcome alternative to Microsoft’s Active Server Page technology (ASP). Another reason PHP is a popular scripting language for a web page generation is it started out as a tool to process HTML [2].

5.3.1 PHP: Functionally Speaking

There are three main uses for PHP: Server-side scripting, Command-line scripting, and Client-side GUI applications. These are described below:

- **Server-side scripting**

This is the “dynamic web page” usage and is still the most common usage. In order to generate HTML it will require the PHP parser, and a web server to send the documents. Server-side scripting is generally more full featured and includes database access. Runs on the server as the name implies and generally results in the output of HTML to the client [25].

- **Command-line scripting**

Much like Perl (another scripting language) and awk (a programming utility), PHP can be run by command line scripts. Traditionally this type of usage is primarily reserved for system administration tasks such as log parsing and backups.

- **Client-side GUI applications with PHP-GTK**

With PHP Graphical Tool Kit [PHP-GTK] (found at <http://gtk.php.net>), it is possible to write full up, cross-platform Graphical User Interface (GUI) applications in PHP [3]. PHP-GTK provides an object-oriented interface to the GTK+ classes and functions, which in turn should greatly simplify the writing of client side cross platform applications.

5.3.2 PHP: Desirable Characteristics

PHP is full-featured programming language and is extremely well suited to web based developments for the following reasons [23]:

- **Portability**

There are several platforms that have PHP distributions available, Windows, UNIX, Linux, Macintosh, and OS/2 to name a few. PHP code written on one platform most of time works on another platform just as is.

- **Performance**

Independent evaluations have declared that PHP is faster and more reliable than other scripting languages. Companies all the way from Fortune 500 to “Mom-and-Pop” businesses are running PHP on their web sites.

- **Ease of Use**

For any programmer familiar with ‘C’, then PHP should not be hard at all to pick up. Even if the programmer is not familiar with ‘C’ PHP is still easy to learn and there are several books on the subject and web sites galore to help with many different programming tasks.

- **Open Source**

Being a child of the open source community itself, PHP source code is freely available and other developers are encouraged to make contributions. Some feel that this open source atmosphere has allowed PHP to gain/enjoy world-wide acceptance.

- **Modular Extensions**

New technology support can be very easily added to augment PHP through its modular extensions. This feature allows developers to quickly add new capabilities and features to their PHP builds. Some of today’s extension capabilities include: performance of File Transfer Protocol (FTP), Internet Message Access Protocol (IMAP), and Post Office Protocol 3 (POP3) operations; and the dynamic generation of Graphics Interchange Format (GIF), Joint Photographic Experts Group (JPEG), and PING images.

- **Database support**

PHP contains out of the box support for a wide variety of databases.

MySQL, the extremely popular database, is well covered by PHP. Some of the other databases which have support in PHP are as follows: IBM DB2, DB2, mSQL, Oracle, PostgreSQL, Sybase, dBase, and Open Database Connectivity (ODBC).

- **Object Oriented Programming (OOP) support**

PHP has had the capability of classes and objects since PHP 3.0. Even though PHP support of Object Oriented Programming (OOP) has always been a little on the incomplete side, PHP version 4.0 is now a little closer to becoming a true object-oriented language. Some of the new features are operator overloading, object nesting, and reference counting.

- **Built-in session management**

PHP is one of the few languages with native session management support, and comes this way out of the box. This makes it possible to track individual client sessions on a web site. The developer can also create web applications that respond to the needs of individual user(s).

5.3.1 PHP: Sample Code

Just what does this language look like? Below is a small example of what a PHP script looks like:

```
<?php

$Serial_Number      = $_GET['Serial_Number'];
$test_start_date    = $_GET['test_start_date'];
$test_end_date      = $_GET['test_end_date'];
$test_start_number  = $_GET['test_start_number'];
$test_end_number    = $_GET['test_end_number'];

//print ("serial number is $Serial_Number");
//print("<br />");

?>
```

All PHP scripts start with “<?php” and end with “?>”. When the web server software hits the beginning of the script it hands it off to the PHP interpreter. The next lines assign internal variables from the system global array `$_GET`. This is done because under HTTP there is no mechanism to pass information from one page to another. HTTP is a stateless protocol, meaning the client connects, makes a request, gets the data and then disconnects.

The “//” indicates that these are comment statements just like in ‘C’ or ‘C++’. The script ends with the “?>”.

All the web pages developed for this project that execute on the web server are a combination of HTML and PHP (refer to Chapter 7 for more details).

5.4 Shortcomings of using PHP

Like the Apache web server there won't be a support group eagerly awaiting a phone call from a customer. However again like Apache there is an abundance of web sites (for example www.php.net), books, user groups, and colleagues just waiting for the questions.

PHP is much like Perl (another scripting language), and it too is constantly being updated much like Linux (A UNIX like OS that is open source too.) Again much like hitting a moving target, the "support" for previous versions does not last long.

6.0 The Relational Database System program MySQL

6.1 What is MySQL?

MySQL is an Open Source SQL database program. It is one of several Open Database Connectivity (ODBC) compliant database programs. ODBC is an open standard programming application for access to databases.

Relational databases are a collection of items (data) in a set of well-defined tables. From these tables the data can be accessed and manipulated in many different ways without having to alter the tables [34].

Other ODBC compliant database programs include: PostgreSQL, Oracle, Sybase, and Microsoft's Access. So it would have been possible to use any of the above listed programs in place of MySQL.

6.2 Why use MySQL?

At the time the author started this project, MySQL was bundled with the PHP interpreter version that was selected. Later versions no longer use MySQL as the database of choice. Even though the later versions of PHP don't bundle up MySQL quite as nicely at the version used for this project, one can either edit the

initialization files to include it; or recompile the source code with the support included. MySQL is free for student use, which fits this usage perfectly.

6.3 How PHP interfaces with Databases

For this project we are using PHP version 4.3.2, and this version provided support for over 20 databases, both popular commercial and the open source varieties. With that being said the designers of that PHP version chose to bundle it with MySQL as the database of choice. Versions released after that no longer had MySQL so neatly bundled. It also has become apparent that the names and file extensions used by MySQL were very different than most other relational databases, and if the user decided later to change databases, the user would then be looking at some rather significant updates [3]. PHP communicates with relational databases (like MySQL) using Structured Query Language (SQL).

SQL is an American National Standards Institute (ANSI) and ISO (used to be referred to as the International Standard Organization) [a body of about 100 countries that govern over worldwide specifications] standard programming and interactive language for obtaining data from, and updating databases [28].

For the purpose of this project PHP will be the interface between the information contained in XML and the MySQL database.

7.0 System Architecture

7.1 More details on the implementation

The two main functions of this project are to provide trending data to the Engineer, and to provide the capability to create new tables and add XML data to those tables.

In the following sections we will discuss the major pieces of the system that were generated to solve the problem of displaying the data to the user from two sources. All the components defined herein work in concert to provide the user with the data displayed on the monitor when processing is complete.

We will begin with a discussion of how XML is displayed by default in Internet Explorer.

7.1.1 How XML appears in Internet Explorer

When XML is displayed in Internet Explorer it looks a lot like a tree outline. There is more than one way to display the data in a more reader friendly format. One way is to use a “style sheet”, another is to write a parser. The author chose to write a parser.

7.1.2 Displaying the data back to the user, Style sheet or Parser?

A style sheet defines how to interpret or translate the logical structure of a document. or basically how it will appear. In terms of the following [33]:

- ◆ Hypertext links are displayed in blue.
- ◆ New chapters are started on the left hand side.
- ◆ Figures are number sequentially throughout the document.

Now since PHP is considered an embedded HTML technology, we are going to use an XML parser to turn an XML document into an embedded part of an HTML web page.

7.1.3 The XML parser

Early in the design phase of this project it had been intended to use Internet Explorer (IE) to do the work of the parser. The embedded parser in Internet Explorer (IE) will display the data.

However it is in a tree format, not conducive to easy reading, see figure 3 below:

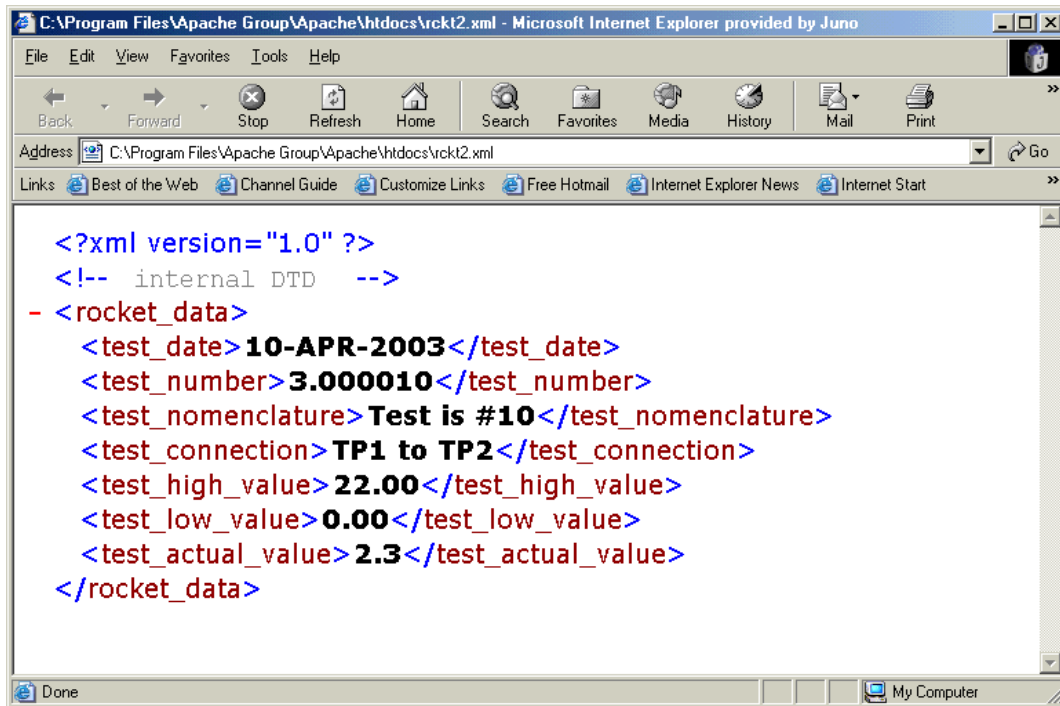


Figure 3 XML tree in Internet Explorer (actual data from the project)

Therefore it was decided to look for other ways to display the data. In the research that was done there was a basic parser in reference [1].

Inside PHP 4 there are several built-in functions for parsing and working with XML documents. The parser itself is based on the Expat parser, which is written by James Clark, and supports XML 1.0.

The Expat parser is an event based parser and will process documents in chunks of data anywhere from a byte to the whole document. When the parser identifies an entity it calls a predefined function to handle the job from there.

There are just a few steps to build a simple parser:

- ◆ The handlers are created.
- ◆ The XML parser is created.
- ◆ The event handlers are registered with the XML parser.
- ◆ Give the XML parser our document to process.
- ◆ Close the parser.

This had to be modified to support the exact elements used in the project. The elements listed in the basic parser [1] were for a specific project and needed to be changed to match the elements used in this project.

The elements used for this project are as follows:

- serial number
- test date
- test number
- test nomenclature
- test connection
- test high value
- test low value
- test actual value

The parser was also changed to display the data in tabular form. This parser was augmented by PHP scripting code that performed the validation of the XML document. In order to be a validated XML document it must be well formed and meet the XML document rules.

To further explain “well formed” it must meet the following criteria:

- a) All elements must be properly closed - means the start tag and end tags must match
- b) Empty elements must be terminated - empty elements are represented in the following manner:

`<empty/>`

- c) Quotes around all attribute values - may use ‘ or “ but must be consistent
- d) All attributes must have values - can not be blank
- e) It is case sensitive - simply stated `<foo></foo>` does not equal `<foo></Foo>`
- f) Proper nesting of tags must be adhered to- means the tags may not be listed out of order,

`<Begin>`

`<step_1>`

`</Begin>`

`</step_1>`,

is of course not allowed and will generate an error. Below is the correct sequence:

`<Begin>`

```
<step_1>...  
</step_1>  
</Begin>
```

The XML document rules are as follows:

- a) Must have a DTD referenced and adhere to it
- b) A single root element is required
- c) Can have an optional XML declaration
- d) Can have an optional *DOCTYPE* declaration

7.2 Displaying the data/user interface concepts

7.2.1 The end user

In most cases the end user will be, either a Test Engineer or a Reliability Engineer that will be looking at a particular measurement looking for trends in the values recorded. There will be some administration issues such as new tables and data to those tables. The addition of tables and adding data to the tables will be done by the Engineer also. The user will select these actions from the web pages displayed on the monitor.

7.2.2 Web pages needed to do the job

A web based design implies a web server and the availability of web pages as the interface to the data housed in the database and the flat files. Figure 4 below shows the pages and how they work together. In essence the end user will have the user log in and request the data and either send that information to a file or to a web page with the information generated from the PHP script embedded in that web page.

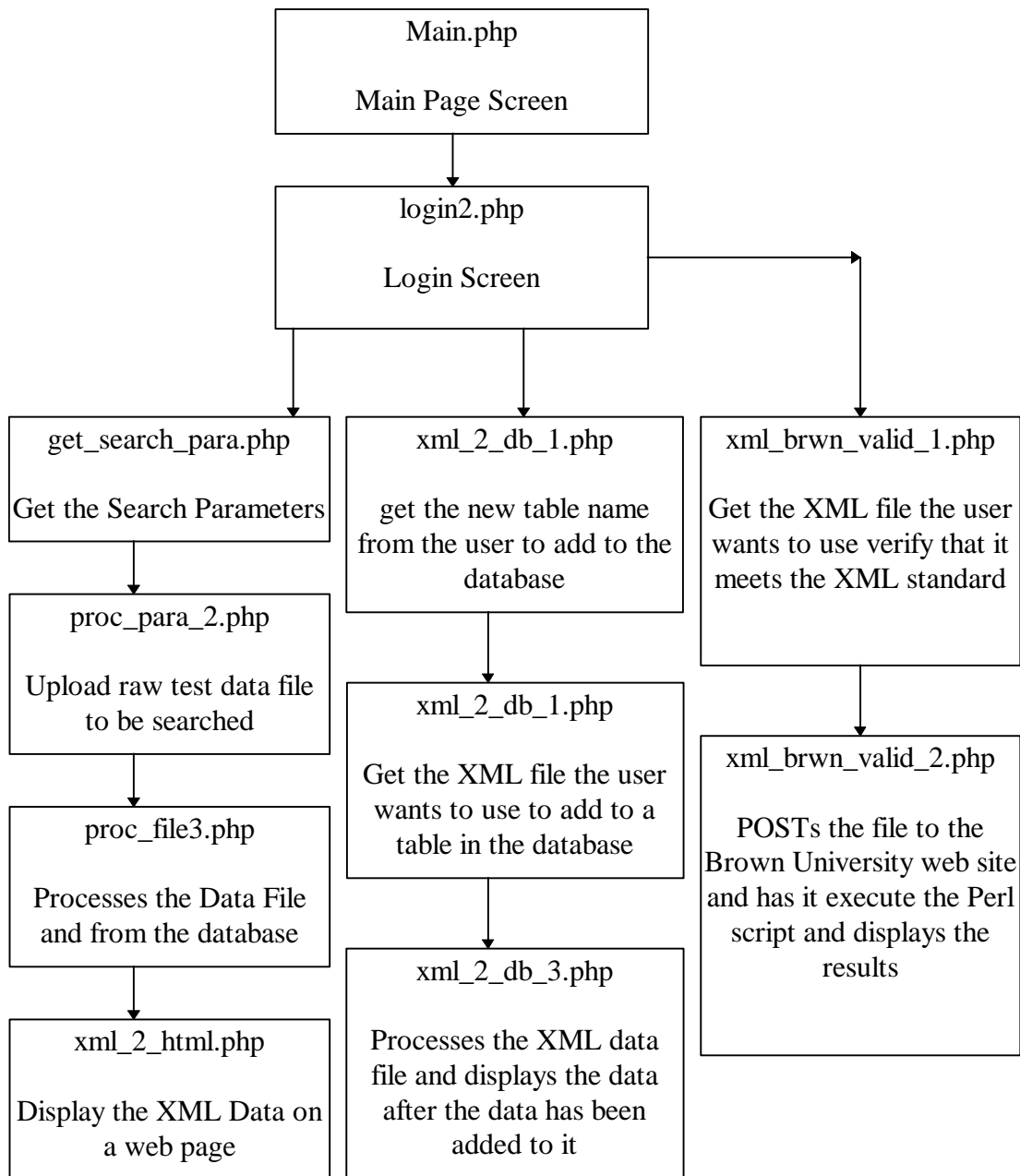


Figure 4 - Web pages required

7.2.3 Brief description of the web pages:

- ◇ ***Main.php*** - this page has the hyperlinks to the two main processes implemented the data retrieval and the database creation
- ◇ ***login2.php*** - this page is the login screen and allows access to the pages after the user has been verified

Trend Analysis functions

- ◇ ***get_search_para.php*** - this is the page that the user types in his/her search parameters for the test data desired
- ◇ ***proc_para_2.php*** - uploads the raw test data file to be searched and sets a cookie to the parameters entered to search with
- ◇ ***proc_file3.php*** - using the search parameters entered previously, this page retrieves the data from the flat file and database and saves it as an XML data file
- ◇ ***xml_2_html.php*** - takes the XML data file and uses the parser found in PHP and displays the data to the user in a tabular form

Add XML to the database (table)

- ◇ ***xml_2_db_1.php*** - gets the table name from the user to put the data into
- ◇ ***xml_2_db_2.php*** - uploads the XML data file from the user
- ◇ ***xml_2_db_3.php*** - processes the uploaded XML data file and displays the contents of the table after it has added the data to it

Validate the XML file created

- ◇ *xml_brwn_valid_1.php* - prompts the user to browse for the XML file they wish to validate at the Brown University web site, and calls the PHP to actually display the results.
- ◇ *xml_brwn_valid_2.php* - Using the PHP function libcurl it connects to the brown University web site and transfers the file uploaded in the previous web page. The Perl script residing at the Brown University site then “validates” the file sent to it. If the validation of the processed file is successful it simply displays “Document validates OK”, otherwise all the errors are listed by line number.

8.0 Results

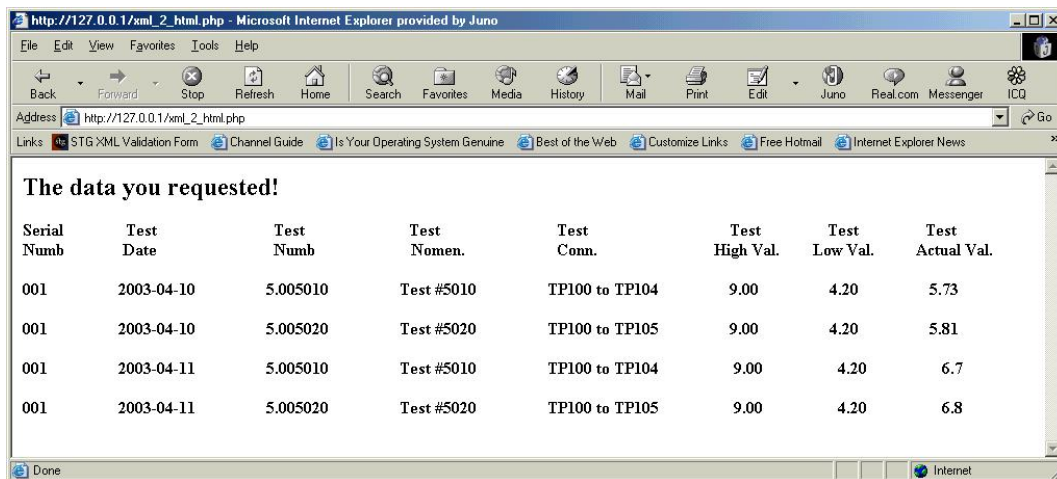
8.1 Results

As in most programming tasks there is more than one way to achieve your goal. In this project PHP scripting and HTML coding were utilized to do various functions that are now module add-ins to the web server. For example, there is now a software web development framework called “Cocoon” that will take XML and display it as HTML on a web page. This could also have been done programming wise using XSL and XSLT (using style sheets).

The fact that the quality and selection of XML tools has improved dramatically since this thesis project was started indicates that XML is a viable and accepted format for data exchange and transmission.

8.1.1 Final Output of the data request

When the user has finished entering the parameters and the software (pages and scripts) has finished processing the data, the data is displayed in a window with the following layout:



The screenshot shows a Microsoft Internet Explorer browser window displaying a table of test results. The browser's address bar shows the URL `http://127.0.0.1/xml_2_html.php`. The table is titled "The data you requested!" and contains the following data:

Serial Numb	Test Date	Test Numb	Test Nomen.	Test Conn.	Test High Val.	Test Low Val.	Test Actual Val.
001	2003-04-10	5.005010	Test #5010	TP100 to TP104	9.00	4.20	5.73
001	2003-04-10	5.005020	Test #5020	TP100 to TP105	9.00	4.20	5.81
001	2003-04-11	5.005010	Test #5010	TP100 to TP104	9.00	4.20	6.7
001	2003-04-11	5.005020	Test #5020	TP100 to TP105	9.00	4.20	6.8

Figure 5 XML converted to HTML

The results of the search are saved as a data file also for later viewing or comparison.

8.1.2 Validating the XML data file

There is a web site (<http://www.stg.brown.edu/service/xmlvalid/>) whereby anyone can submit the XML document in question to see if it does indeed meet the XML 1.0 standards. The author could not find where it clearly stated it would validate to an internal DTD, but this was tested and found to be the case. It does indeed validate the file to the internal DTD. This site was used to ensure that the

document that the author generated using a PHP script did indeed meet that standard.

Reading the install document carefully, it states that 'xmlparse.pl' will validate that the document's internal structure in question meets the XML specification. There are also instructions on how to load this Perl script onto the end user's machine. However since the project was not designed to use Perl we could not pursue this option any further. Also currently it only calls out UNIX as the operating platform.

The error list for when the document does not meet the standard is very extensive and is included for reference in Appendix G. If (or when) the document in question finally meets the XML specification a page very similar to figure 6 shown on next page will then be displayed: Please refer again to chapter 7.2.3 for more details as how this function was accomplished.

A representative example of the actual information that is displayed back to the user delineating the results can be found on the next page, reference figure 6. If the actual file is small (less than 1k in size) the page will display the file contents also. If the file is larger than that the script will not display the file contents back to the user. Figure 6 (shown on next page) depicts a file that has validated "OK". The author experienced several pages that depicted errors found in the file and examples of such have been omitted for clarity.

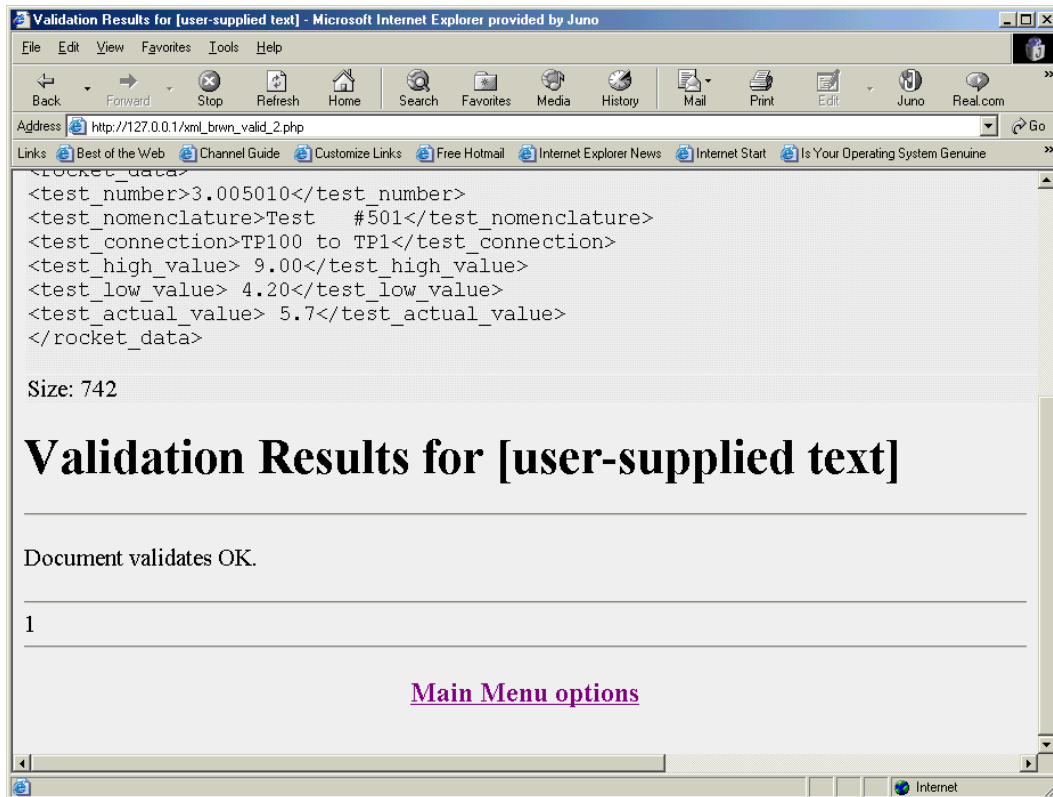


Figure 6 XML validation message

9.0 Interpretation of Findings / Future Work

9.1 Interpretation of Findings

The author feels that the work done on this project and thesis lays the foundation for and clearly shows that the concept (PHP, Apache, and MySQL) will work on a much larger scale/scope. The ideal implementation for this project of course would be when the testing work was pulled back from the vendor in an effort to consolidate the work into just one contract done at just one location. There are issues that must be resolved if the information continues to be housed in the two separate locations. One is the setup of the server at the vendor's site to ensure security of data and system. Also there is the consolidation of the data at the end user's site. The work invested to combine the files using a computer program would be well worth the time, as it would take many hours to manually enter the data and verify that it is correct. There is also the matter of making sense of test numbers which are different at both locations. Researching trends further, this type of programming is now done using graphical techniques and or programming integrated development environments such as Cold Fusion.

This project will allow the test engineer or the quality engineer the ability to search for negative trends in the hardware using the computer versus having to do the data review manually, thereby greatly reducing the chances for human error in the process.

The system as a whole will be more cohesive and easier to use than again having to utilize the hard copy for any searching projects. It is always easier to have the data

in one location versus several and this will at least give the appearance that the data is all in one location.

9.2 The PHP solution

In this author's opinion, using PHP could be teaching some bad programming techniques. There was no need to prototype, declare, or initialize anything. It was very easy just merely to start typing away. Another interesting feature is that if the "view source" is selected from the pull-downs while in the browser, any PHP code will not be displayed. However if the PHP interpreter is not running you can see the code.

The author has personally visited at least one web site that uses PHP to support their web operations, www.niti.com. Also recently the author received a magazine (Network Computing, February 5th 2004) that talked about the use of PHP, MySQL, and Apache. The article [38] was not specifically about the combination of the above three pieces of software into a system, but once again shows that these programs/paradigms are still very much in the news.

9.3 XML the data standard

XML is a data standard and in the author's opinion, probably would be even if there were not any specifications claiming it to be so. It is inherently more informative than plain data. Anyone can view the information and quickly understand what is being conveyed. Also PHP the scripting language lends itself very nicely to the generation of XML data.

By the very nature of the descriptive tags used the information provided is very easy to read and understand even by those that are not computer literate. The author recently loaded a popular program and noticed that the help files had the XML extension for the file names. XML is inherently Internet friendly which is undoubtedly the wave of the future. Designed to be used on the Internet XML performs to important functions. First XML provides a means for a user to huge amounts of data more descriptively. Secondly as a standard mechanism for exchange of information, data is encoded in a format conducive to transmittal from computer to computer [23].

XML is basically text so it can be FTP'd or HTTP'd or sent by any other text capable protocol. Also bear in mind that there is an ever increasing number of XML products being brought to the market to assist programmers and developers.

9.4 Future Work / Suggestions for further study

It is suggested that future work would include the lookup table that cross references the test numbers between the two systems. Presently the test numbers are not the same at the two testing sites. The authors also suggest the possibility of adding features to combine the files here and at the vendor into one large file, either a flat file or into the database/tables.

Another area that can be explored is the addition of a search for the individual components that make up the larger black boxes. There could be a search routine that searches through the database for the installed component based on the date of the testing to find what card is installed in the larger component under test.

The author could not find a standard DTD/Schema at this time for the industry of Electronic (black box) testing, so it might be possible to contact a committee and work towards that end.

Still another area could be the addition of help files and or help screens that utilized the XML format.

There could be a search done at the company level to determine if maybe other testing done somewhere else in the company that lends itself to this upgrade and there may already be DTD's or Schema's that exists for those projects.

From the above it can be clearly seen that there is plenty of enhancements that can be made to the system as a whole. In general the system is a viable working solution to the problem at hand and can be readily and easily improved upon.

List of References

Text books:

- [1] Pat Coleman (editor), "XML Complete" SYBEX Inc., 1151 Marina Village Parkway, Alameda CA 94501, © October 1 2001

- [2] David Siklar and Adam Trachtenberg, "PHP Cookbook" O'Reilly and Associates, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, © 2003

- [3] Rasmus Lerdorf and Kevin Tatroe, "Programming PHP" O'Reilly and Associates, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, © 2002

- [5] Hungry Minds, "PHP: Your visual blueprint for creating open source, server-side content", Hungry Minds, Inc., 5755 Coopers Avenue, Mississauga, Ontario, Canada, maranGraphics, CA 95472, © 2001

- [6] Elizabeth Castro, "HTML 4 for the World Wide Web Forth Edition: Visual QuickStart Guide", Peachpit Press, Inc, 1249 Eight Street, Berkeley, CA 94710, © 2001

- [23] Vikram Vaswani, "XML and PHP", First Edition June 2002, New Riders Publishing, 201 West 103rd Street, Indianapolis, Indiana 46290, © 2002

- [32] David Medinets, "PERL5 by Example", Second Edition 1996, Que Corporation, 201 West 103rd Street, Indianapolis, Indiana 46290

Web sites/pages:

- [11] <http://xml.coverpages.org/ni2002-01-04-a.html> then it led to this one:
<http://www.diglib.org/preserve/hadtdfs.pdf> (current as of 10/12/03)

- [12] http://www.w3schools.com/xml/xml_usedfor.asp (current as of 10/12/03)

- [14] <http://www.phpnoise.com/tutorials/4/1> this is the article for good programming habits (current as of 02/03/04)

- [15] <http://whatis.techtarget.com> (current as of 10/12/03)

- [16] http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci213606,00.html (current as of 02/03/04)

- [17] http://whatis.techtarget.com/definition/0,,sid9_gci211576,00.html (current as of 02/03/04)

- [18] www.php.net (For the scripting language PHP) (current as of 10/12/03)

- [19] www.apache.org (The web server organization page) (current as of 10/12/03)

- [20] www.netcraft.com (The web server survey results) (current as of 10/12/03)

- [23] <http://z.iwethey.org/forums/render/content/show?contentid=6621> (current as of 02/03/04)

- [24] <http://mired.org:8080/home/mwm/scripting/what.html> (current as of 10/11/03)

- [25] http://searchwebservices.techtarget.com/ateQuestionNResponse/0,289625,sid26_cid413696_tax287609,00.html (current as of 10/12/03)

- [26] <http://webdesign.about.com/library/weekly/aa090500a.htm> (current as of 10/12/03)

- [27] <http://httpd.apache.org/docs-2.0/platform/windows.html> (current as of 10/13/03)

- [28] http://searchdatabase.techtarget.com/sDefinition/0,,sid13_gci214230,00.html (current as of 10/14/03)

- [29] http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci343029,00.html (current as of 10/18/03)

- [30] <http://www.openssl.org/> (current as of 10/18/03)

- [33] <http://www.xml.com/pub/a/1999/01/walsh1.html> (current as of 10/22/03)

- [34] http://searchdatabase.techtarget.com/sDefinition/0,,sid13_gci212885,00.html (current as of 10/25/03)

- [36] <http://www.niti.com/docs/home.php?id=1> (current as of 02/23/04)

- [37] <http://www.stg.brown.edu/service/xmlvalid/> (current as of 03/01/04)

Technical Articles/papers:

- [4] Jayavel Shanmugasundaram, Eugene Shekita, Rimon Barr, Michael Carey, Bruce Lindsay, Hamid Pirahesh, Berthold Reinwald , “Efficiently publishing relational data as XML documents”, The VLDB Journal (2001) 10: 133–154 / Digital Object Identifier (DOI) 10.1007/s007780100052, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, USA

- [13] John T. Battalio, ”Extensible Markup Language: How might it alter the Software documentation process and the role of the technical

communicator?”, J. Technical Writing and Communication, Vol. 32(3) 209-244, 2002

- [18] Philip J. Bernhard, Gary Faulkner, Ophit Frieder, “XML Strategies for Legacy Database Search”, IT Pro March/April 2003 1520-9202/03

- [21] Stuart E. Madnick, “The Misguided Silver Bullet: What XML Will And Will Not Do To Help Information Integration”, MIT Sloan School of Management Sloan Working Paper 4185-01 eBusiness@MIT Working Paper 111 October 2001

- [22] Art Rhyno, “XML and relational databases: uses and opportunities for libraries”, OCLC Systems and Services, Volume 18 Number 2, 2002, 97-103 MC UPB Limited ISSN 1065-075X

- [31] Richard Morin and Vicki Brown, “I/Opener, Scripting Languages”, Sun Expert Magazine September 1998

- [35] David Axmark and Michael (Monty) Widenius, MySQL Reference Manual for version 4.0.5., Swedish company MySQL AB, © 2002

- [38] Don MacVittie, “Where does Linux fit?”, Network Computing, February 5th 2004

Appendix 'A' The Flat file output format

START OF PROGRAM Black Box Out-the-door test DATE: 10-APR-2003 TIME:
08:03:53.49

Data Storage Enabled on Station Tester

1.000010 PATCH PANEL Verification	10.00	0.00	0.82 OHM
1.000020 VDC ON MEAS	0.10	-0.10	0.02 V
1.000030 VAC ON MB S/B	1.00	0.00	0.14 V
1.000040 MB ISOLATION	1000.00	2.00	1000.00 MOHM
1.000050 24V STATION PS	24.60	23.40	23.83 V
1.000060 12V STATION PS	12.30	11.70	12.05 V
1.000070 -12V STATION PS	-11.70	-12.30	-12.15 V
1.000080 5V STATION PS	5.15	4.85	5.04 V

BB Product Testing

TYPE NO. XYZ

SERIAL NO. 1

TEST DESCRIPTION Final Acceptance

Floor Paper number: BB000123

CONTINUITY Checks

10-APR-2003 08:04:38

TEMPERATURE: AMBIENT

Next is the tabular data header which is as follows:

Name	Time	“HIGH”	“LO”	“ACTUAL”
BB THRU-CIRC WIRES	08:04:38	HIGH	LOW	ACTUAL

This is followed directly by the actual tabular formed data:

Test Num.	Test Nome.	Conn. points	HIGH Val.	LO Val.	Meas. Val.
3.000010	Test is #10	TP1 to TP2	22.00	0.00	2.35 OHM
3.000020	Test is #20	TP1 to TP3	22.00	0.00	2.06 OHM
3.000030	Test is #30	TP1 to TP4	22.00	0.00	2.52 OHM
3.000040	Test is #40	TP1 to TP5	22.00	0.00	2.41 OHM
3.000050	Test is #50	TP1 to TP6	22.00	0.00	2.73 OHM

•

•

•

10-APR-2003 08:06:16 ELAPSED 00:01:38 TOTAL 00:01:38

Data Storage Disabled on Station Tester

END OF PROGRAM "Out-the-door test" DATE: 10-APR-2003 TIME:

10:15:14.67

Appendix 'B' Database Fields

The fields in this table are as follows:

BB S/N	Test Date	Test Numb.	Test Nom.	Conn. Points	High Val.	LO Val.	Actual. Val.
001	01/01/2001	3.000010	Test is #10	TP1 to TP2	22.00	0.00	2.35 OHM
001	01/01/2001	3.000020	Test is #20	TP1 to TP3	22.00	0.00	2.06 OHM
001	01/01/2001	3.000030	Test is #30	TP1 to TP4	22.00	0.00	2.52 OHM
001	01/01/2001	3.000040	Test is #40	TP1 to TP5	22.00	0.00	2.41 OHM
001	01/01/2001	3.000050	Test is #50	TP1 to TP6	22.00	0.00	2.73 OHM
001	01/01/2001	3.000060	Test is #60	TP1 to TP7	22.00	0.00	3.18 OHM

Appendix 'C' XML Development Process Stages

XML Development Process

Stage 1 Structuring content with

- Document Type Definitions
- XML Schema
- XML Content Files

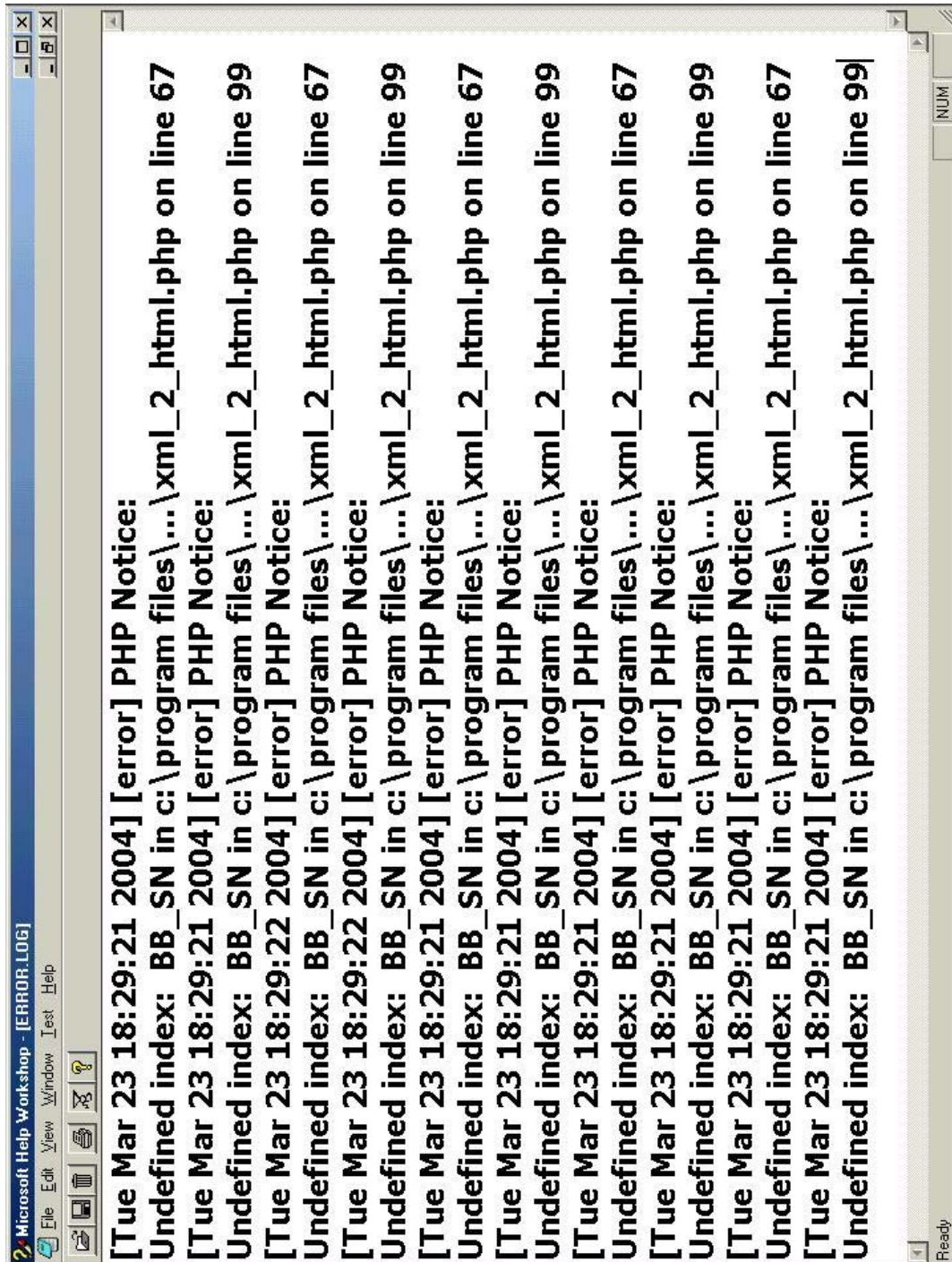
Stage 2 Formatting content with

- XSL Transformations (XSLT)
- Cascading Style Sheets (CSS)
- Extensible Stylesheet language (XSL)

Stage 3 Linking and manipulating content with

- XML Link Language (XLink)
- XML Pointer Language (XPoint)
- XML Path Language (XPath)

Appendix 'D' Example Error file output



```
[Tue Mar 23 18:29:21 2004] [error] PHP Notice:  
Undefined index: BB_SN in c:\program files\...\xml_2_html.php on line 67  
[Tue Mar 23 18:29:21 2004] [error] PHP Notice:  
Undefined index: BB_SN in c:\program files\...\xml_2_html.php on line 99  
[Tue Mar 23 18:29:22 2004] [error] PHP Notice:  
Undefined index: BB_SN in c:\program files\...\xml_2_html.php on line 67  
[Tue Mar 23 18:29:22 2004] [error] PHP Notice:  
Undefined index: BB_SN in c:\program files\...\xml_2_html.php on line 99  
[Tue Mar 23 18:29:21 2004] [error] PHP Notice:  
Undefined index: BB_SN in c:\program files\...\xml_2_html.php on line 67  
[Tue Mar 23 18:29:21 2004] [error] PHP Notice:  
Undefined index: BB_SN in c:\program files\...\xml_2_html.php on line 99  
[Tue Mar 23 18:29:21 2004] [error] PHP Notice:  
Undefined index: BB_SN in c:\program files\...\xml_2_html.php on line 67  
[Tue Mar 23 18:29:21 2004] [error] PHP Notice:  
Undefined index: BB_SN in c:\program files\...\xml_2_html.php on line 99  
[Tue Mar 23 18:29:21 2004] [error] PHP Notice:  
Undefined index: BB_SN in c:\program files\...\xml_2_html.php on line 67  
[Tue Mar 23 18:29:21 2004] [error] PHP Notice:  
Undefined index: BB_SN in c:\program files\...\xml_2_html.php on line 99
```

Figure 7 Web server error file

Appendix 'E' Sample Web pages with PHP scripts

xml_brwn_valid_1.php

```
<html>
<head>
<title>Upload XML data file to send to Brown University website</title>
</head>
<BODY BGCOLOR="#F0F8FF" TEXT="#2F4F4F" LINK="#708090" VLINK
="#4682B4" ALINK="#800000">
<body>

<h1>Upload XML data file to send to the Brown University site </h1>

<!-- Go get the file name from the user -->

<!-- this displays the browse field for the file form -->
<!-- <FORM ENCTYPE="multipart/form-data" method="POST"> -->
<!-- <INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="300000"> -->
<!-- Send this file: <INPUT NAME="userfile" type="file"> -->

<HR>

    <FORM ENCTYPE="multipart/form-data" METHOD="POST"
    ACTION="xml_brwn_valid_2.php">
    <INPUT TYPE="hidden" NAME="language" VALUE="en">
    <TABLE>
    <TR>
    <TD>
    <FONT SIZE="+1">Local file:</FONT>
    </TD>
    </TR>
    <TR>
    <TD COLSPAN=3>
    <INPUT TYPE="file" SIZE=40 NAME="uploaded_file">
    </TD>
    </TR>

</HR>
</P></P><br><br>
```

xml_brwn_valid_1.php - continued

```
<!-- this displays send button -->  
<TD><br><br>  
<input type = "Submit" VALUE = "Upload - Send file">  
</TD>  
</form>  
  
</body>  
</html>
```

xml_brwn_valid_2.php

```
<?php

// File originally written by: Ian Koss           02/05/04
//           Updated by:   Mark Gibson 02/08/04

//      Read the XML file from the global array of files uploaded
//      $filename      = $_FILES['uploaded_file']['tmp_name'];

//      old hard coded location
//      $filename = "rckt.xml";
//      $handle = fopen($filename, "rb");
//      $file_size = filesize($filename);
//      $xml = fread($handle, $file_size);

//      if (is_uploaded_file($_FILES['uploaded_file']['tmp_name']))
//      {
//          echo 'successful upload of the XML file';
//      }

// if file size less than 1k go ahead and display the file contents
// if ($file_size < 1000)
// {

//      Echo XML send the contents of the file to the new page
//      echo '<table>';
//      echo '<tr><td bgcolor="#EEEEEE">XML</td></tr>';
//      echo '<tr><td
// bgcolor="#EEEEEE"><pre>'.htmlentities($xml).'\</pre></td></tr>';
//      echo '<tr><td bgcolor="#EEEEEE">Size: '.$file_size.'\</td></tr>';
//      echo '</table>';

//      } // end if

//      fclose($handle);

//      Set up query
//      $query = array ();
```

xml_brwn_valid_2.php - continued

```
$query['language'] = 'en';
$query['text'] = $xml;

/*      libcurl, a library created by Daniel Stenberg, that allows you to connect
and communicate to many
        different types of servers with many different types of protocols. libcurl
currently supports the http, https,
        ftp, gopher, telnet, dict, file, and ldap protocols. libcurl also supports
HTTPS certificates, HTTP POST,
        HTTP PUT, FTP uploading (this can also be done with PHP's ftp
extension), HTTP form based upload,
        proxies, cookies, and user+password authentication.
*/

//      Initialize cURL handle
$url = "http://www.stg.brown.edu/cgi-bin/xmlvalid/xmlvalid.pl";
$referer = "http://www.stg.brown.edu/service/xmlvalid/";
$ch = curl_init($url);

//      Set cURL options
curl_setopt($ch, CURLOPT_REFERER, $referer);
curl_setopt($ch, CURLOPT_VERBOSE, 1); //Set this to a non-zero value if
you want CURL to report everything that is happening
curl_setopt($ch, CURLOPT_HEADER, 0); //Set this to a non-zero value if
you want the header to be included in the output
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1); //Set this to a non-
zero value to follow any "Location: " header that the server sends as a part of the
HTTP header
curl_setopt($ch, CURLOPT_POST, 1); //Set this option to a non-zero value
if you want PHP to do a regular HTTP POST
curl_setopt($ch, CURLOPT_POSTFIELDS, $query); //Pass a string
containing the full data to post in an HTTP "POST" operation.

//      Execute
$result = curl_exec($ch); // run the whole process
```

xml_brwn_valid_2.php - continued


```
//      Show results on my page
echo '<hr>'.$result.'<hr>';

//      Cleanup the cURL operation
      curl_close($ch);

?>

<h3><p><center>
<a href="http://192.168.1.101/Main.php" target="_blank">Main Menu
options</a><br><br>
<p><center>
<input type="button" value="Back" name="ClickBack" onclick=(history.back())>
</h3>
```

Appendix 'F' Implementation Hurdles

Implementation Hurdles

While debugging the PHP code it first came out looking correct. Since the author was not fully up on the tags for HTML it was not immediately apparent that code was not entirely working as written. Now since we had added PHP as a module to the web server now all the error messages were going to that error log file. During the debug sessions it had been run so many time that it was not able to be opened by WORD. The error log then had to be deleted (actually just renamed) and started over. What was discovered was that there were a few errors being generated but of course not displayed to the user. The code was reviewed and found to be missing a few key items. This was quickly remedied but now it did not work at all. The error file was reviewed again and found to be "timing out". By taking what was being displayed it looked like the parser was having a problem with the end tag routine. Studying that code closely it was discovered that the procedure in the book was incorrect; the decrement instruction only had one minus sign. After that was corrected the code worked properly. There were still a couple of other loose items, the quickest way to get rid of the errors in the error log was to add attributes to the paragraph markers until I could figure out a more elegant method to preclude the errors showing up in the error log file.

While searching the web for information the author found a site that had an article on programming habits when using PHP [9]. The article starts out by saying that due to the explosion of computing power in all aspects of our lives, there are probably and unfortunately more examples of bad code/design than in many other fields. In this author's opinion some of what was said is basic knowledge taught (should be) in a beginning level programming class. One point the article made was to put the variable type as part of the variable name something this author tends to agree with, even though it was not stressed when he went to school. This author having experiences (not just with PHP) trying to understand someone else's code in order to modify or correct it. In some instances it takes a while to decipher code someone has written himself if he has not reviewed it in several months. So it does indeed help to use variable names that actually mean something to someone else who may have to deal with your code after you have gone to greener pastures. The author read an article many years ago that basically stated that you need to be tricky only when necessary, if you have the memory and luxury it is better to do things a little more straight forward than tricky. This may even help yourself in the long run, just in case you don't get to greener pastures soon enough.

Appendix 'G' The Perl XML Validation Script Error messages

```
#####  
#  
#   $RCSfile: messages,v $  
#   $Date: 1999/09/14 17:09:40 $  
#   $Source: /home/richard/Xml/RCS/messages,v $  
#   $Revision: 1.169 $  
#   $Author: richard $  
#  
#####  
#  
# Default message catalog for xmlparse. Used by errabort() and  
# errwarn() (see errabort.c). Used by other stuff too (any time we  
# want to write out a string that users might see, we want to use  
# this catalog).  
#  
# ALTHOUGH XMLPARSE DOES NOT STRICTLY ENFORCE IT, THIS FILE SHOULD  
# BE ENCODED WITH UTF-8 CHARACTERS. ALL ASCII CHARACTERS ARE UTF-8  
# CHARACTERS, SO CAUTION IS ONLY WARRANTED IF YOU ARE USING ISO  
# 8859-x UPPER RANGE CHARACTERS.  
#  
# The format of this file is trivially simple. The file is made up  
# lines. Lines form string key-message pairs and comments. Comments  
# are basically anything after an un-quoted number-sign (#) and may  
# occur either alone on a line, or after a string key-message pair.
```

```

# String key-message pairs consist of an arbitrary string followed by
# whitespace, followed by a quoted message, e.g.:
#
# 1 "this is a quoted message" # this is a comment
#
# Lines may be continued with a backslash, if necessary:
#
# 1 "this is a very long message that we can't comfortably keep \
#   on a single line"
#
# Use getmessage() to access the messages here (see getmessage.c).
# Before calling getmessage() for the first time, be sure to call
# create_message_catalog() (also in getmessage.c). E.g.:
#
# message_catalog *mc;
#
# /* initialize message catalog to use US English */
# mc = create_message_catalog ("messages.en-US");
# ...
# /* oops; syntax error; tell the user how to get help */
# fprintf (stderr, getmessage (mc, "10"));
#
# /* done with catalog (normally we're never done with it) */
# free_message_catalog (mc);
#
# Note also that, when you call errabort(), you are making use of
# this catalog indirectly. E.g., errabort (40, "malloc error in
# %s\n", "main()") actually tries to use message number 40 below in
# place of the supplied default, "malloc error in %s\n".
#
#####
#

```

```

# Help, usage messages
#
1 "usage: %s %s %s\n"
2 "[-c <config filename>] [-C <SGML catalog name>]\n \
  [-d <dirname>] [-E <max errors>] [-f] [-h] [-l <debug level>]\n \
  [-m <message catalog>] [-n] [-p <FPI resolution command>]\n \
  [-s] [-u <URL resolution command>] [-v] <filenames...>\n"
3 "\n\
  Command-line arguments:\n\
\n \
  -c <filename> use <filename> as the configuration file\n \
  -C <filename> use <filename> as SGML catalog file\n \
  -d <dirname> set datafile directory to <dirname>\n \
  -E <max errors> display only <max errors> errors/warnings\n \
  -f force GIs/atts in namespaces to validate OK\n \
  -h emit a brief help message and exit\n \
  -l <level> set debug level to <level> (if enabled)\n \
  -m <filename> use <filename> as message catalog\n \
  -n resolve only http: and urn: system ids\n \
  -p <command> bypass catalog; use <command> to resolve FPIs\n \
  -s report errors/warnings via syslog(3)\n \
  -u <command> use <command> to resolve URIs\n \
  -v emit version number and exit\n \
  <filenames...> (names of XML files or URIs to process)\n\
\n\
  Pecking order:\n\
\n \
  - Configuration-file directives override command-line args\n \
  - Command-line args override compiled-in defaults\n \
  - Compiled-in defaults override nothing";

```

```

# Exit status 4 means that one or more files triggered validation
# errors (messages will have been emitted).
#
4 "there were errors\n"
5 "reached max_errors; too many errors/warnings to print for %s\n"
6 "-l switch ignored; debugging facilities not compiled in\n"
7 "debug_level setting in config file ignored; debugging facilities not compiled in\n"

# Simple command-line syntax errors (xmlparse.c; see also 70-89)
#
10 "command-line syntax error; invoke with -h option for help\n"
11 "no files to process\n"

# Basic system errors
#
12 "path exceeds system size limit: %s\n"
13 "can't unlink file, %s\n"
14 "timed out waiting for lock to release on %s\n"
15 "buffer too small in %s\n"
16 "absolute path/URI lacks leading slash, %s\n"
17 "cannot fork in %s\n"
18 "cannot create pipe in %s\n"
19 "failed write in %s\n"
20 "failed fdopen in %s\n"
21 "too many open files\n"

# Helper app problems
#
30 "system error executing shell command, %s\n"
31 "error executing shell command, %s; status = %d\n"
32 "shell command, %s, returned no text\n"
33 "external command timed out, %s\n"

```

```

34 "external command produced no output, %s\n"
35 "external command, %s, killed; signal = %d\n"

# Memory allocation and re-allocation errors
#
40 "memory allocation error in %s\n"
41 "memory re-allocation error in %s\n"
42 "unexpectedly non-null value in %s\n"
43 "unexpectedly null value in %s\n"
44 "unexpected enumerated type in %s\n"
45 "unexpected return value from %s\n"
46 "unexpected value for variable in %s\n"

# Grievous programming errors
#
50 "wrong type attribute, %s\n"
51 "node, %s, already has children\n"
52 "non-element parsed as STag\n"
53 "tried to merge xml_file structs after document content was reached\n"
54 "tried to dereference ID attribute with xmlparse_env.keep_children set to 'no'"

# Security or URI-resolution problems
60 "can't resolve non-http/urn/ftp URI in no-local-file mode, %s\n"
61 "can't resolve pathname or URI, %s\n"
62 "can't resolve localhost URL in no-local-file mode, %s\n"

# Initialization errors (readcfg.c; see 10-20 above)
#
70 "invalid debug_level (-d %s)\n"
71 "error opening message file, %s\n"
72 "can't cd into %s; using current directory\n" # same as 130 below
73 "can't change into directory, %s\n" # same as 131 below

```



```

74 "can't open configuration file, %s; ignoring\n"
75 "unknown keyword, line %d of configuration file; ignoring\n"
76 "bad %s value, %s, line %d in configuration file; ignoring\n"
77 "ignoring negative integer in configuration file, %d\n"
78 "integer too big, %d; ignoring\n"
80 "can't determine current working directory\n" # same as 136 below

# Signal-handling stuff (sigstuff.c)
#
90 "%s failed\n"

# Message-catalog errors (getmessage.c)
#
130 "can't cd into %s; using current directory\n" # same as 72 above
131 "can't change into directory, %s\n" # same as 73 above
132 "missing message value, line %d\n"
133 "invalid message key, line %d\n"
134 "invalid message value, line %d\n"
135 "duplicate key, message catalog, line %d\n"
136 "can't determine current working directory\n" # same as 80 above

# File processing and I/O errors
#
140 "I/O error reading %s\n"
141 "error opening XML file, %s\n"
142 "error reading XML file, %s\n"
145 "unknown byte order, %s, in XML file, %s\n"
146 "missing byte order mark in XML file, %s\n"
147 "EBCDIC character encoding not supported for XML file, %s\n"
148 "seek error %d for file %s\n"
149 "file, %s, must be seekable\n"
#

```

```

150 "error closing XML file, %s\n"
151 "short read (%d) from file, %s\n"
152 "can't convert %s texts\n"
153 "replacing out-of-range UCS-4 character, %X, with 0x0000FFFD\n"
154 "out-of-sync UTF-8 character at offset %ld in %s; resynchronizing\n"
155 "corrupt UTF-8 character at offset %ld in %s\n"
156 "oops; xml_file structure is already a child\n"
157 "tried to pop element off empty xml_file_stack\n"
158 "can't make an xml_file a child of itself\n"

# DTD, SGML catalog errors
#
160 "SGML catalog maps pubid %s to an unreadable file, %s\n"
161 "external command, %s, failed to produce a readable file, %s\n"
#
170 "can't open SGML catalog file, %s, for reading\n"
172 "error reading SGML catalog file, %s\n"
173 "more than %d catalog files are in your SGML_CATALOG_FILES set; abandoning\n"
175 "public identifier, %s (in %s), already defined; skipping\n"
176 "catalog filename, %s, is too long\n"
179 "too many errors; skipping remainder of %s\n"

# UTF/UCS conversion utility errors
#
180 "corrupt UTF-8 character in string %s\n"
181 "out-of-sync UTF-8 char at offset %ld in string %s; resynchronizing\n"
182 "corrupt UTF-16 character, %X,%X\n"

```

```
#####
```

```
# Errors in parser. These must all have the following format:
```

```

# "error message, %s, line %d: %s\n" where the first %s will end
# up being the filename, the %d will be the line number, and the
# last %s will be the token that triggered the error.

# General parsing, I/O errors:
#
300 "general failure parsing XML document, %s, line %d: %s\n"
301 "bogus document prolog, %s, line %d: %s\n"
302 "superfluous material at end-of-file, %s, line %d: %s\n"
303 "parsing error, %s, line %d: %s\n"
304 "superfluous keyword, %s, line %d: %s\n"
305 "error in external file included via decl ending it, %s, line %d; for: %s\n"
#
320 "I/O error reading %s at line %d: %s\n"
321 "improperly terminated construct; line numbers may be off, %s, line %d: %s\n"
#
330 "out-of-sync UTF-8 character, %s, line %d: %s\n"
331 "corrupt UTF-8 character, %s, line %d: %s\n"
332 "out-of-range character, %s, line %d: %s\n"
333 "unknown Shift-JIS character, %s, line %d: %s\n"

# <?xml errors, bogus processing instructions
#
350 "bogus encoding declarator, %s, line %d; after: %s\n"
351 "bogus processing instruction, %s, line %d: %s\n"
352 "missing encoding declarator in TextDecl, %s, line %d: %s\n"
353 "standalone declaration in external TextDecl, %s, line %d: %s\n"
354 "bogus processing instruction target, %s, line %d: %s\n"
#
358 "Encoding and version decls are reversed, %s, line %d: %s\n"
359 "TextDecl used in place of XMLDecl, %s, line %d: %s\n"
#

```

360 "invalid XML version name/num, %s, line %d: %s\n"

361 "cannot parse XML version specified at %s, line %d: %s\n"

362 "invalid standalone spec, %s, line %d; after: %s\n"

363 "misplaced XML declaration, %s, line %d: %s\n"

364 "improperly terminated TextDecl, %s, line %d: %s\n"

365 "keyword must be lowercased, %s, line %d: %s\n"

366 "unexpected, misplaced, or unknown keyword used in XML declaration, %s, line %d: %s\n"

367 "XML decl not given literally, %s, line %d: %s\n"

368 "keyword must be uppercased, %s, line %d: %s\n"

369 "extra material before TextDecl end delimiter, %s, line %d: %s\n"

370 "invalid TextDecl, %s, line %d: %s\n"

371 "unexpected EOF; improperly terminated XML declaration, %s, line %d; starts at line: %s\n"

372 "unexpected EOF; improperly terminated processing instruction, %s, line %d; starts at line: %s\n"

373 "improperly terminated processing instruction, %s, line %d: %s\n"

#

381 "unknown encoding, %s, line %d: %s\n"

382 "unsupported encoding, %s, line %d: %s\n"

383 "encoding contains a character that is neither alphanumeric nor a dash, %s, line %d: %s\n"

384 "actual file format does not match declared encoding, %s, line %d: %s\n"

386 "spurious whitespace before <?xml decl, %s, line %d: %s\n"

#

390 "processing instruction contains a colon (discouraged until further specs appear), %s, line %d: %s\n"

<!DOCTYPE errors

#

400 "bogus DOCTYPE declaration, %s, line %d: %s\n"

401 "doctype name doesn't match GI for top-level element, %s, line %d: %s\n"

402 "EOF encountered; no doctype declaration found, %s, line %d: %s\n"

405 "markup looks suspiciously like a DOCTYPE decl, %s, line %d: %s\n"

406 "DOCTYPE decl in document declared as standalone references an external DTD, %s, line %d: %s\n"

```

#
410 "DOCTYPE declaration in external XML file, %s, line %d: %s\n"
411 "end-internal-dtd-delimiter found in external entity, %s, line %d: %s\n"
412 "DOCTYPE declaration occurs (recursively) inside DTD, %s, line %d: %s\n"
#
420 "DOCTYPE declaration lacks system identifier (required in XML), %s, line %d: %s\n"

# General DTD errors; PE ref errors in DTD
#
450 "unresolvable PE reference, %s, line %d: %s\n"
451 "unterminated PE reference, %s, line %d: %s\n"
452 "unexpected EOF; improperly terminated DTD section, %s, line %d; starts at line: %s\n"
#
460 "general-entity reference in DTD not in entity or attribute value, %s, line %d: %s\n"
461 "character-entity reference in DTD not in entity or attribute value, %s, line %d: %s\n"
462 "extraneous string in DTD, %s, line %d: %s\n"
463 "markup error in DTD, %s, line %d: %s\n"
464 "regular markup occurs inside DTD, %s, line %d: %s\n"
465 "regular markup occurs in external parameter entity (extSubsetDecl), %s, line %d: %s\n"
466 "apparent character data where markup is expected, %s, line %d: %s\n"

# General declaration errors
#
500 "bad declaration, %s, line %d; token: %s\n"
501 "spurious whitespace in processing instruction, %s, line %d: %s\n"
502 "spurious whitespace in declaration, %s, line %d: %s\n"
503 "unexpected EOF; improperly terminated declaration, %s, line %d; starts at line: %s\n"
504 "character is illegal in its current context, %s, line %d: %s (a cascade of errors may follow)\n"
505 "possible unterminated markup, %s, line %d; before: %s\n"
510 "improperly terminated quoted string, %s, line %d: %s\n"
511 "keyword lacks leading number sign (#), %s, line %d: %s\n"

```

```

# Unresolvable PUBLIC and SYSTEM identifiers
#
550 "unquoted SYSTEM identifier, %s, line %d: %s\n"
551 "unquoted PUBLIC identifier, %s, line %d: %s\n"
#
562 "can't resolve Public ID, %s, line %d: %s\n"
563 "can't resolve System ID, %s, line %d: %s\n"
564 "public ID resolves as empty string, %s, line %d: %s\n"
565 "bogus (in this case, empty) system ID, %s, line %d: %s\n"
566 "external ID resolution command failed, %s, line %d; for: %s\n"

# Syntax errors in PUBLIC and SYSTEM identifiers
#
567 "malformed public identifier string, %s, line %d; after: %s\n"
568 "malformed public/system identifier, %s, line %d: %s\n"
569 "illegal character in public identifier, %s, line %d: %s\n"

# <!ATTLIST errors
#
580 "ID attribute declared for an element that already has one, %s, line %d: %s\n"
581 "discarding duplicate attribute definition, %s, line %d: %s\n"
582 "attribute declared for an (as-yet) undeclared element, %s, line %d: %s\n"
583 "invalid language attribute declaration, %s, line %d: %s\n"
584 "attribute type must be enumerated (default|preserve), %s, line %d: %s\n"
#
585 "empty enumeration in attribute definition, %s, line %d: %s\n"
586 "illegal default for ID attribute, %s, line %d: %s\n"
587 "ID attribute default type not #REQUIRED or #IMPLIED, %s, line %d: %s\n"
588 "external attribute default occurs in document declared as standalone, %s, line %d: %s\n"
589 "error in default attribute value, %s, line %d: %s\n"
#
590 "superfluous quoted string in attribute default, %s, line %d: %s\n"

```

591 "apparent attlist declaration lacks exclamation point, %s, line %d: %s\n"

592 "external DTD subset supplies a default attribute in a document declared standalone, %s, line %d: %s\n"

#

600 "unquoted attribute default, %s, line %d: %s\n"

601 "unsupported attribute default, %s, line %d: %s\n"

602 "unsupported attribute type, %s, line %d: %s\n"

603 "illegal operator in enumerated attribute value list, %s, line %d: %s\n"

#

610 "invalid material trailing attribute definition, %s, line %d: %s\n"

611 "invalid attribute name, %s, line %d: %s\n"

612 "unexpected Name, %s, line %d: %s\n"

#

620 "namespace attribute value normalizes as a blank string, %s, line %d: %s\n"

621 "default entity value has a malformed expansion, %s, line %d: %s\n"

<!ELEMENT errors

#

650 "discarding duplicate element declaration, %s, line %d: %s\n"

651 "invalid element declaration, %s, line %d: %s\n"

652 "element has more than one attlist declaration, %s, line %d: %s\n"

653 "invalid element name, %s, line %d: %s\n"

655 "skipping SGML cruft, %s, line %d: %s\n"

#

658 "RCDATA elements not allowed in XML, %s, line %d: %s\n"

659 "CDATA elements not allowed in XML, %s, line %d: %s\n"

660 "invalid type, %s, line %d; for attribute: %s\n"

661 "duplicate value for ID attribute, %s, line %d: %s\n"

662 "EOF encountered; can't find ID value to match IDREF attribute, %s, line %d: %s\n"

#

667 "malformed mixed content model, %s, line %d; after: %s\n"

668 "mixed content model lacks trailing star, %s, line %d: %s\n"

669 "illegal operator in content model, %s, line %d; for: %s\n"

670 "symbol used twice in name alternative list, %s, line %d: %s\n"

671 "bogus content model, %s, line %d; for: %s\n"

672 "end of DTD; can't find declaration for symbol used in content model, %s, line %d: %s\n"

673 "apparent element declaration lacks exclamation point, %s, line %d: %s\n"

674 "empty content model (use keyword EMPTY instead), %s, line %d; for: %s\n"

675 "use of reserved word in children (\"nested\") content model, %s, line %d: %s\n"

676 "error parsing content model, %s, line %d: %s\n"

677 "extra material after content model, %s, line %d: %s\n"

678 "inclusion and exclusion exceptions are illegal in XML, %s, line %d: %s\n"

#

680 "ambiguous content model, %s, line %d; symbol(s): %s\n"

681 "element named more than once in mixed content model, %s, line %d: %s\n"

682 "XML does not allow name groups, i.e., folding of multiple element or attribute declarations into a single declaration, %s, line %d: %s\n"

683 "empty or malformed NameGroup, %s, line %d: %s\n"

684 "there is no #CDATA keyword; use #PCDATA instead, %s, line %d: %s\n"

685 "XML has no #ALL keyword; you must declare attributes individually, for each element, %s, line %d: %s\n"

#

690 "end of DTD; unused element(s) detected, %s, line %d: %s\n"

691 "end of DTD; attributes declared for an element that is never defined, %s, line %d; element: %s\n"

<!NOTATION errors

#

700 "discarding duplicate notation declaration, %s, line %d: %s\n"

701 "incompatible notation redeclaration, %s, line %d: %s\n"

702 "empty notation name list, %s, line %d; for: %s\n"

703 "apparent notation declaration lacks exclamation point, %s, line %d: %s\n"

<!ENTITY errors


```

#
# Errors in entities, mainly parameter entities:
#
745 "duplicate parameter entity declaration (will be discarded if used), %s, line %d: %s\n"
750 "discarding duplicate parameter entity declaration, %s, line %d: %s\n"
751 "incompatible parameter entity redeclaration, %s, line %d: %s\n"
752 "apparent parameter entity declaration lacks exclamation point, %s, line %d: %s\n"
753 "entity declared in external DTD subset of document declared as standalone, %s, line %d: %s\n"
754 "built-in entity redeclared external, %s, line %d: %s\n"
755 "built-in entity not redeclared according to the spec, %s, line %d: %s\n"
756 "built-in entity redeclared within another entity (may have unintended side-effects), %s, line %d:
%s\n"
757 "percent sign entity declared within another entity (may have unintended side-effects), %s, line
%d: %s\n"
758 "literal percent sign used as entity value in external entity (may have unintended side-effects),
%s, line %d: %s\n"
#
760 "parameter entity replacement text in content model begins with an operator, %s, line %d: %s\n"
761 "parameter entity replacement text in content model ends with an operator, %s, line %d: %s\n"
762 "illegally grouped parameter entity replacement text in content model, %s, line %d; character:
%s\n"
763 "null parameter entity replacement text in content model, %s, line %d: %s\n"
#
770 "external entity's URL points back at a parent file, %s, line %d: %s\n"
771 "malformed external parameter entity, %s, line %d; file: %s\n"
772 "malformed internal parameter entity, %s, line %d: %s\n"
774 "error in external DTD subset included via DOCTYPE decl ending at, %s, line %d; for: %s\n"
775 "unbalanced '<' and '>' characters in parameter entity replacement text, %s, line %d: %s\n"

```

```

#
780 "missing end tag, %s, line %d: %s\n"
781 "invalid entity name, %s, line %d: %s\n"

# Errors in general and unparsed entities:
#
800 "discarding duplicate entity declaration, %s, line %d: %s\n"
801 "incompatible entity redeclaration, %s, line %d: %s\n"
802 "apparent entity declaration lacks exclamation point, %s, line %d: %s\n"
803 "external entity declared in DTD of document declared as standalone, %s, line %d: %s\n"
810 "notation name is not (yet) defined, %s, line %d: %s\n"
811 "end of DTD; missing declaration for notation used in entity decl, %s, line %d; notation: %s\n"
812 "end of DTD; missing declaration for notation used in attlist decl, %s, line %d; notation: %s\n"
#
820 "external entity declaration lacks system identifier (required in XML), %s, line %d: %s\n"
821 "entity has a type that is illegal in XML, %s, line %d: %s\n"
822 "bypassed ampersand in literal text of general entity, %s, line %d: %s\n"
#
830 "default entity declaration is illegal in XML, %s, line %d: %s\n"
840 "malformed entity declaration, %s, line %d; after: %s\n"
841 "extra material in entity declaration, %s, line %d: %s\n"
842 "entity definition in internal DTD subset contains a parameter entity reference, %s, line %d:
%s\n"

# <[ INCLUDE/IGNORE marked sections
#
850 "unterminated conditional, %s, line %d; detected at: %s\n"
851 "improperly nested conditional, %s, line %d: %s\n"
852 "missing [ in conditional section, %s, line %d: %s\n"
#

```

860 "ignoring illegal marked section in document content, %s, line %d: %s\n"

861 "illegal marked section in internal DTD subset, %s, line %d: %s\n"

862 "non-INCLUDE/IGNORE marked section encountered while scanning DTD, %s, line %d: %s\n"

863 "found an illegal second DOCTYPE declaration, %s, line %d: %s\n"

<!-- COMMENT errors

#

900 "discarding apparent old-style SGML comment, %s, line %d: %s\n"

901 "deprecated sequence within comment ending at, %s, line %d: %s\n"

902 "comment ends in more than two dashes, %s, line %d: %s\n"

903 "detected a malformed or unterminated comment, %s, line %d: %s\n"

Entity expansion problems

#

1000 "parameter entity within a markup declaration in the internal DTD subset, %s, line %d: %s\n"

1002 "entity expands recursively, %s, line %d: %s\n"

1003 "entity (or its expansion) is invalid, %s, line %d: %s\n"

1004 "malformed entity reference, %s, line %d: %s\n"

#

1010 "entity's expansion is invalid, %s, line %d; token: %s\n"

1012 "reference to undeclared entity, %s, line %d: %s\n"

1013 "notation attribute doesn't match any declared notation name, %s, line %d: %s\n"

1014 "reference to an unparsed entity, %s, line %d: %s\n"

#

1020 "parameter entity replacement text is malformed, %s, line %d: %s\n"

1021 "improperly nested external entity, %s, line %d; state is: %s\n"

Other entity problems

#

1050 "entity reference outside of document content, %s, line %d: %s\n"

1051 "for implementation reasons the null character entity is not allowed, %s, line %d: %s\n"

1052 "ignoring apparent malformed (#-less) character entity, %s, line %d: %s\n"

1053 "ignoring unterminated character entity, %s, line %d: %s\n"

1054 "character entity specifies value in Unicode surrogate block, %s, line %d: %s\n"

1055 "character entity maps to a byte-swapped byte-order mark, %s, line %d: %s\n"

1056 "character entity specifies an ASCII control character, %s, line %d: %s\n"

1057 "non-Unicode character, %s, line %d: %s\n"

1058 "character entity specifies a Unicode C1 control character, %s, line %d: %s\n"

Element problems in document body

#

1080 "ignoring spurious exclamation point (is this markup in the document body?), %s, line %d: %s\n"

1081 "ignoring spurious percent sign (is this an ASP tag?), %s, line %d: %s\n"

#

1098 "end tag doesn't match its start tag, %s, line %d: %s\n"

1100 "end tag found for element that is not open, %s, line %d: %s\n"

1101 "element declared as empty has content, %s, line %d: %s\n"

1102 "tag uses GI for an undeclared element, %s, line %d: %s\n"

1103 "end tag uses GI for an undeclared element, %s, line %d: %s\n"

1104 "document body starts with CharData, %s, line %d: %s\n"

1105 "end tag with no start tag, %s, line %d: %s\n"

1106 "empty-tag syntax used for element not declared with EMPTY content model, %s, line %d: %s\n"

1107 "element declared with children content model contains a marked section, %s, line %d: %s\n"

1109 "superfluous material in end tag, %s, line %d: %s\n"

1110 "malformed end tag, %s, line %d: %s\n"

1111 "empty end tag, %s, line %d: %s\n"

1112 "there is no #EMPTY keyword; use EMPTY instead, %s, line %d: %s\n"

#

1114 "spurious whitespace, %s, line %d; after: %s\n"

1115 "spurious whitespace in tag, %s, line %d: %s\n"

1116 "remember that XML is case-sensitive, %s, line %d; for: %s\n"

#

1120 "unexpected EOF; improperly terminated markup, %s, line %d; starts at line: %s\n"

1121 "malformed (unterminated?) tag, %s, line %d: %s\n"

1122 "spurious punctuation inside markup, %s, line %d: %s\n"

1123 "GI begins with illegal character sequence, %s, line %d: %s\n"

#

1150 "enclosing tag undefined or lacks content model; can't check child, %s, line %d: %s\n"

1151 "enclosing tag declared as empty; can't check child, %s, line %d: %s\n"

1152 "element violates enclosing tag's content model, %s, line %d: %s\n"

1153 "invalid enclosing element, can't check child, %s, line %d: %s\n"

1154 "content ends prematurely for element, %s, line %d: %s\n"

1155 "non-element violates element-only content model of enclosing tag, %s, line %d: %s\n"

1156 "content-model check terminated before extra element, %s, line %d: %s\n"

1157 "content-model check out of sync with element parse, %s, line %d: %s\n"

#

1165 "externally defined element directly contains whitespace in document declared standalone, %s,
line %d: %s\n"

Attribute problems in document body

#

1200 "attribute value doesn't match fixed default, %s, line %d: %s\n"

1201 "required attribute missing, %s, line %d: %s\n"

1202 "attribute used in an element for which it is not declared, %s, line %d: %s\n"

1203 "attribute can't be checked because element is not defined, %s, line %d: %s\n"

1205 "attribute value doesn't match any declared (enumerated) value, %s, line %d: %s\n"

1206 "attribute value in standalone document changes as a result of normalization, %s, line %d:
%s\n"

1207 "attribute value defaulted in a document declared standalone, %s, line %d; for attribute:
%s\n"

#

1210 "value appears in multiple enumerations for attributes of one element, %s, line %d; attribute: %s\n"

1211 "external entity reference in attribute value, %s, line %d; attribute: %s\n"

1215 "discarding duplicate attribute, %s, line %d: %s\n"

1216 "unquoted attribute value, %s, line %d: %s\n"

1217 "discarding duplicate namespace-resolved attribute, %s, line %d: %s\n"

#

1220 "invalid (nonstandard) language code, %s, line %d: %s\n"

1221 "character in attribute value is illegal according to declaration, %s, line %d: %s\n"

1222 "bogus attribute, %s, line %d; before: %s\n"

#

1223 "argument length is excessive; consider breaking the value out as element content, %s, line %d: %s\n"

#

1230 "reserved character, '<', used in attribute value, %s, line %d: %s\n"

CDATA section errors

#

1300 "CDATA end delimiter with no CDATA start (use > to escape), %s, line %d: %s\n"

1301 "CDATA section not in document body, %s, line %d: %s\n"

1310 "illegal character data type, RCDATA, %s, line %d: %s\n"

1311 "detected a malformed or unterminated marked section, %s, line %d: %s\n"

1312 "spurious whitespace in marked-section start delimiter, %s, line %d: %s\n"

#

1315 "illegal marked section delimiter, use <![CDATA... instead, %s, line %d: %s\n"

General Content errors

#

1350 "illegal character, %s, line %d: %s\n"

1351 "malformed Content, %s, line %d: %s\n"

1352 "reached garbage-character limit, %s, line %d: %s\n"

1353 "illegal digit-initial name at, %s, line %d: %s\n"

1354 "illegal control character, %s, line %d: %s\n"

1355 "name has illegal characters, %s, line %d: %s\n"

Namespace errors

#

1370 "only one namespace permitted; too many colons in end-tag name, %s, line %d: %s\n"

1371 "empty LocalPart; i.e., end-tag name ends in a colon, %s, line %d: %s\n"

1372 "empty prefix; i.e., end-tag name begins with a colon, %s, line %d: %s\n"

#

1373 "only one namespace permitted; too many colons in attribute name, %s, line %d: %s\n"

1374 "empty LocalPart; i.e., attribute name ends in a colon, %s, line %d: %s\n"

1375 "empty prefix; i.e., attribute name begins with a colon, %s, line %d: %s\n"

#

1376 "only one namespace permitted; too many colons in element name, %s, line %d: %s\n"

1377 "empty LocalPart; i.e., element name ends in a colon, %s, line %d: %s\n"

1378 "empty prefix; i.e., element name begins with a colon, %s, line %d: %s\n"

#

1380 "multiple namespaces in attribute name appearing in declaration ending at, %s, line %d;
attribute: %s\n"

1381 "LocalPart missing from attribute name appearing in declaration ending at, %s, line %d;
attribute: %s\n"

1382 "namespace prefix missing from attribute name appearing in declaration ending at, %s, line %d;
attribute: %s\n"

#

1383 "undeclared namespace, %s, line %d: %s\n"

1384 "cannot declare reserved xml- namespace, %s, line %d: %s\n"

1385 "cannot redeclare built-in xml namespace, %s, line %d: %s\n"

#

1390 "namespace-declaring attribute not declared #FIXED or #REQUIRED (this is not good style),
%s, line %d: %s\n"

1391 "namespace-declaring attribute inserted via DTD defaulting mechanisms (this is not good
style), %s, line %d: %s\n"

1392 "default namespace-declaring attribute (xmlns) inserted via DTD defaulting mechanisms (this is not good style), %s, line %d; for element: %s\n"

Appendix ‘H’ Application Instructions

H.1 Installing the applications

H.1.1 Apache Web Server

The first step was to download the installation file (binary application) and then set up the web server. Before the application would function properly however, the support for Transmission Control Protocol/Internet Protocol (TCP/IP) had to be installed on the target machine. The machine chosen for this project already had the TCP/IP support loaded. The author had loaded this support many years earlier prior to the start of this project. The steps taken to setup the web server were fairly straight forward and easy to follow [5].

Referring to the Apache web site documentation, it was highly recommended that the end user download the digital signature keys and perform another program to verify that the software the user downloaded was not hacked into in any way. Following the hyperlinks to the other web sites listed on the Apache web site, the author downloaded the windows version of the program that would then verify the digital keys.

After running the program that generated the digital signature and making sure that the digital signature was correct, Apache was then installed following the instructions in “PHP: Your visual blueprint for creating open source, server-side content” [5], without further incidents.

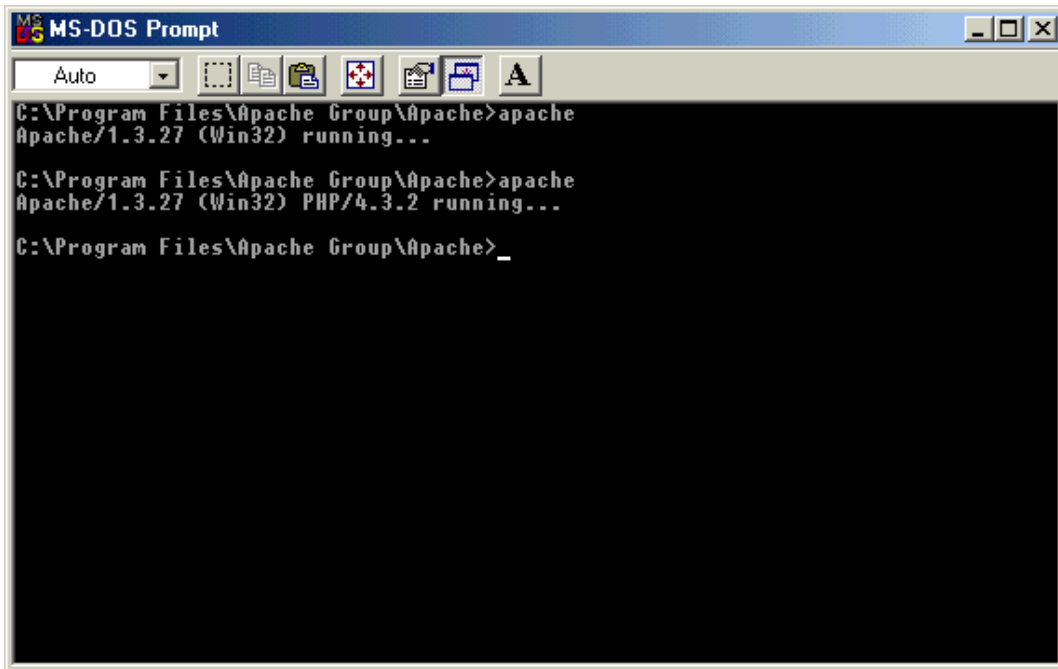
After the web server was installed it had to be configured to allow it to properly interface with PHP documents. The following lines were added to the configuration file "httpd.conf":

```
ScriptAlias /php/ "c:/php/"  
AddType application/x-httpd-php .php .phtml  
Action application/x-httpd-php "/php/php.exe".
```

Early in the implementation of the system, the author figured out he had added PHP as a CGI script instead of as a module and had to go back and add another line to the "httpd.conf" file to have Apache load PHP as a module. The line is shown below:

```
LoadModule php4_module C:/php/sapi/php4apache.dll
```

After that when the Apache application was launched it came up as follows where it shows PHP 4.3.2 is running, the line above it shows how the Apache application came up when PHP was being executed as a CGI call (refer to figure 8):



```
MS-DOS Prompt
Auto
C:\Program Files\Apache Group\Apache>apache
Apache/1.3.27 (Win32) running...
C:\Program Files\Apache Group\Apache>apache
Apache/1.3.27 (Win32) PHP/4.3.2 running...
C:\Program Files\Apache Group\Apache>_
```

Figure 8 PHP CGI Script call screen

H.1.2 Installing the PHP interpreter

After the binary was downloaded and checked for the digital signature, it was loaded much the same way the web server was by following the instructions found in “PHP: Your visual blueprint for creating open source, server-side content” [5].

Next PHP must be configured with/into a Web Server. And once again there were several choices available to the author. As stated earlier PHP can be setup to run as a stand alone CGI script or linked directly into the server via the server’s native Server API (SAPI). This is support for SAPI for the web server that was chosen for use on this project. There is also support for IIS, AOLserver, and Netscape iPlanet. One so inclined can also run it as a Java servlet engine. However [Programming PHP] warned against using the SAPI approach at the time of the writing of the book

as it was felt to be too unstable (especially for production use). One can configure the server either manually or by the use of an “installer”.

H.1.3 Installing MySQL

The first task was to download from the web site the Windows binaries in ‘zip’ form after reading all the licensing agreements. Since the software written here is not for profit, there will be no help whatsoever from the company if there are problems. There is a plethora of information available on the subject, including a reference manual available from the web site. The manual’s ‘step-by-step’ instructions for PC installation are:

1. Review your system to ensure it meets the basic system requirements
2. Use the setup executable
3. Decide which of the 5 binaries included to run
4. Use an option file if one of these binaries is to be used `mysqld.exe`, `mysqld-max.exe`, or `mysqld-max-nt.exe`
5. After installation is complete test it in a DOS window
6. Finally there is a recommendation to read the section on Running MySQL on a Windows box

Further reading into the section on running on a Windows box revealed a important side note: regular task manager could not be used to kill MySQL once started. Instead, the user must type ‘`mysqladmin shutdown`’ in a DOS window. Also filenames and tables are case sensitive.

H.2 Testing the installations

H.2.1 Apache: The Web Server

After the installation was complete there was a test to be made to ensure that the installation was made correctly. Since this was a load onto a Windows machine using Windows Me, it had to be started a little differently than shown in the books. For a Windows 9x derivative it is better to start it as a console operation and leave the windows open during the time needed. This was fairly simple and merely entailed opening a MS-DOS window and starting Apache and then opening an Internet Explorer application and typing in the address 127.0.0.1. If this worked correctly the screen below will be shown:

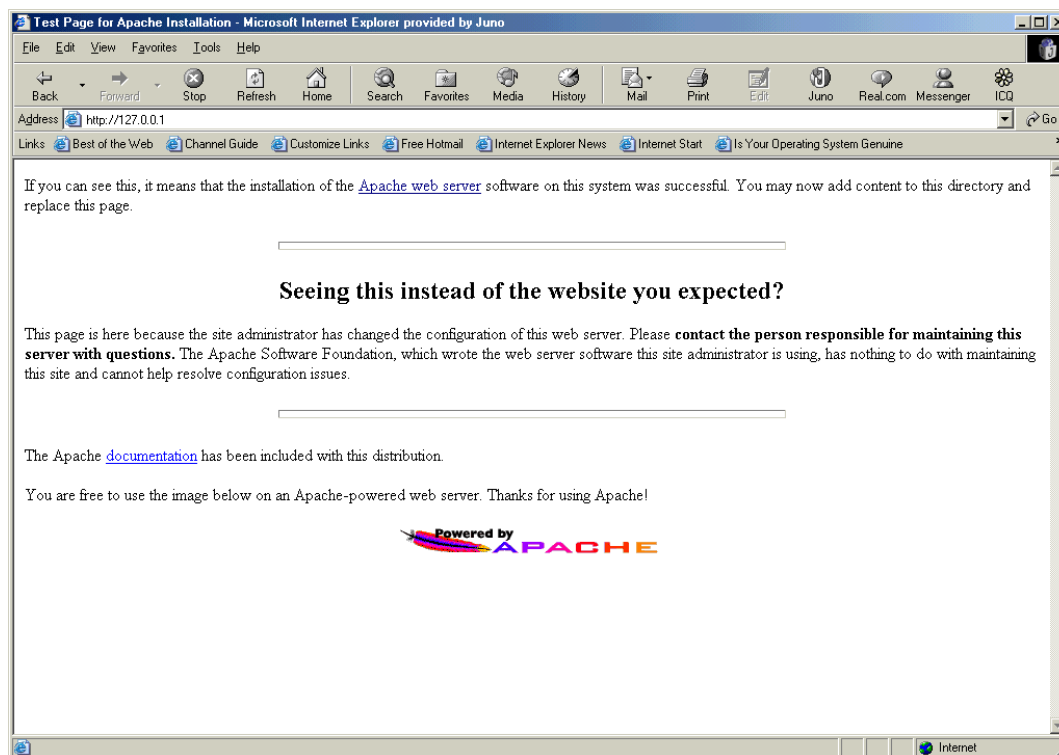
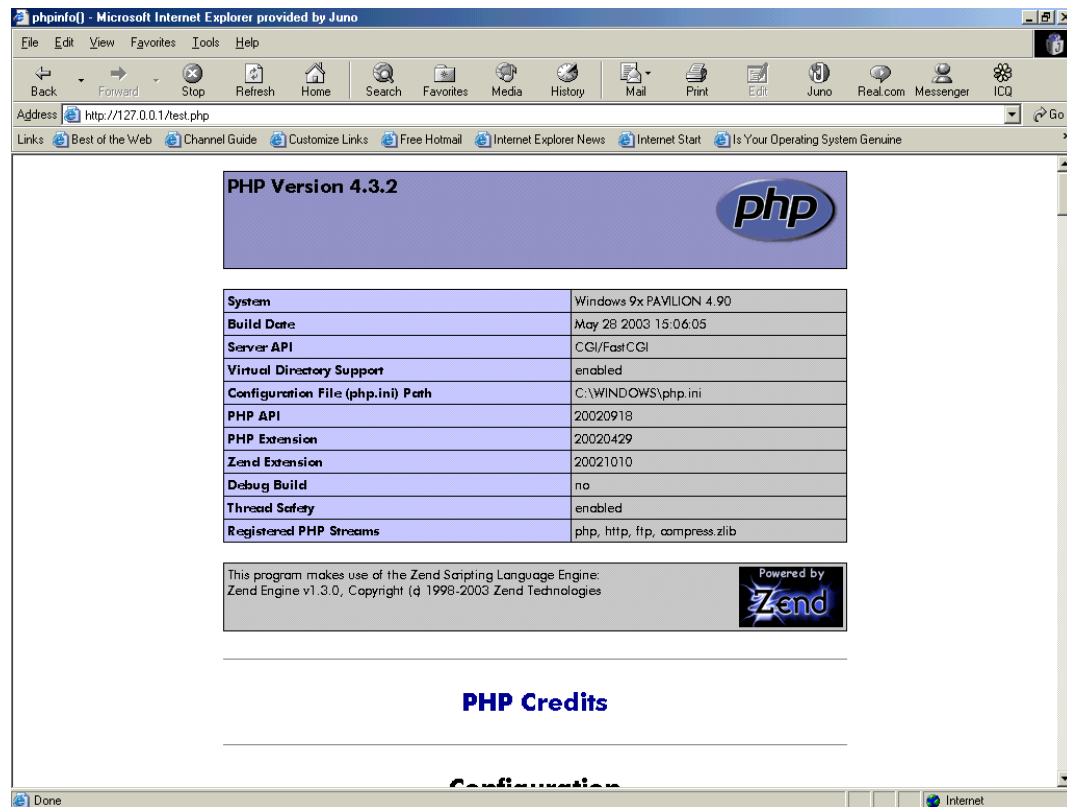


Figure 9 Apache Web Server setup correctly screen

H.2.2 The PHP Interpreter

There is a test to see if the two pieces of the puzzle are working up to this point. To do this one creates a test web page containing PHP scripting, and save it as “test.php”. Then one just calls up the browser and types in the local address of 127.0.0.1/test.. This test page runs a PHP function called phpinfo() which sends back to the browser a plethora of information seen below (note there are several pages worth of information as a result of the performance of this function):



The screenshot shows a Microsoft Internet Explorer browser window titled "phpinfo() - Microsoft Internet Explorer provided by Juno". The address bar displays "http://127.0.0.1/test.php". The main content area features a blue header with "PHP Version 4.3.2" and the PHP logo. Below this is a table with the following data:

System	Windows 9x PAVILION 4.90
Build Date	May 28 2003 15:06:05
Server API	CGI/FastCGI
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS\php.ini
PHP API	20020918
PHP Extension	20020429
Zend Extension	20021010
Debug Build	no
Thread Safety	enabled
Registered PHP Streams	php, http, ftp, compress.zlib

Below the table, there is a note: "This program makes use of the Zend Scripting Language Engine: Zend Engine v1.3.0, Copyright (c) 1998-2003 Zend Technologies" and a "Powered by Zend" logo. At the bottom of the page, the text "PHP Credits" and "Configuration" is visible.

Figure 10 PHP information page

H.2.3 The MySQL database

Referring to the Windows help file it suggests that testing should be done from a DOS window. This is because the server will display status messages in the DOS window. Next, make sure that the next command is executed where MySQL is located:

```
C:\mysql\bin> mysqld-max --standalone
```

You should see the following messages as the server starts up:

```
InnoDB: The first specified datafile c:\ibdata\ibdata1 did not exist:
```

```
InnoDB: a new database to be created!
```

```
InnoDB: Setting file c:\ibdata\ibdata1 size to 209715200
```

```
InnoDB: Database physically writes the file full: wait...
```

```
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
```

```
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
```

```
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
```

```
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
```

```
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
```

```
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
```

```
InnoDB: Doublewrite buffer not found: creating new
```

```
InnoDB: Doublewrite buffer created
```

```
InnoDB: creating foreign key constraint system tables
```

```
InnoDB: foreign key constraint system tables created
```

```
011024 10:58:25 InnoDB: Started
```

H.3 PHP for the Windows platform PC

H.3.1 Configuration issues with PHP installations on Windows PC's

Even after reviewing the reference material the author still found it confusing as to what would be best approach to get the software installed onto the PC. The author

chose what software to load, not just “how to” but “how to configure it after it was loaded”. PHP can be configured as either a Dynamic Link Library (DLL) or as a Common Gateway Interface (CGI) script. The incorrect choice will most likely improperly configure the system, and it will not work properly if at all.

H.3.2 Open Source or Binaries

One choice that is offered is to download the source code since it is open source, but most people don’t have the compiler. So the alternative is to download the binary distributions for Windows systems. Now at the root level of your distribution there will be the file “*php.exe*”, which the user can run from the command line, provided they have copied or created some sort of test file. There were several short examples in the books that were reviewed. The author chose to download the binaries for this application.

H.3.3 Installation situation overall

There is at least one book that the author found, that took the user step by step (within reason) through the installation processes for each system [5]. However since this book was a couple of years old there were a few things changed between the writing of the book and when the author started on this project. One conclusion the author arrived at is, even though you have to be careful when using information found on the web, it may be best to go there and double check the process before using information that may be a couple of years old.

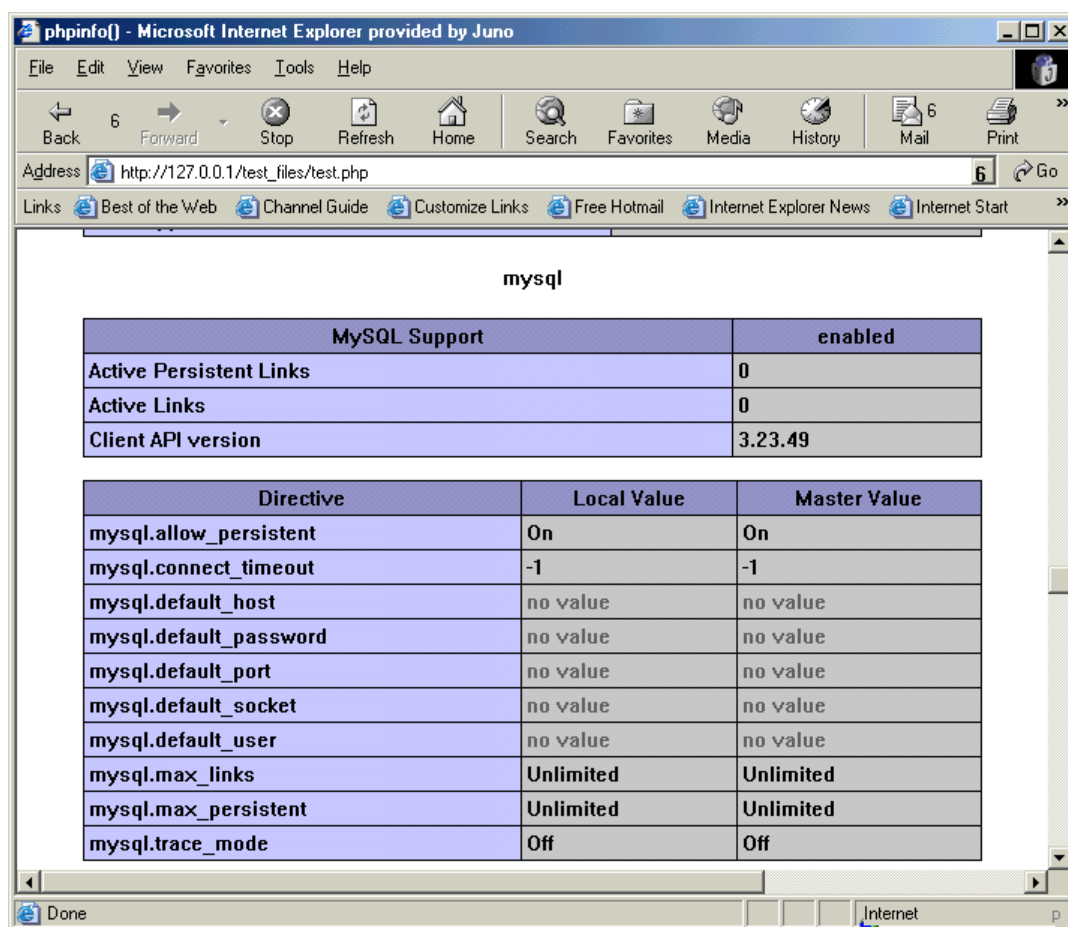
There is thankfully a set of “configuration” steps that is common to all Microsoft installations, regardless of the server chosen. The installation using the installer was

chosen at this time as it configured the system as needed for this project and had it set up for use with MySQL automatically. Now after answering a few questions it was finished.

H.4 PHP/MySQL support validated

One of the first things needed early on, is to ensure that the PHP module/interpreter has MySQL support included. This validates that the correct functionality is in place. This was achieved by reviewing the output to the web page when the text below was entered onto a web page and executed; there will be a section with the details of the MySQL support see below:

```
<? phpinfo(); ?>
```



The screenshot shows a Microsoft Internet Explorer browser window with the title "phpinfo() - Microsoft Internet Explorer provided by Juno". The address bar displays "http://127.0.0.1/test_files/test.php". The main content area shows the output of a PHP script, specifically the "mysql" section. This section contains two tables: one for "MySQL Support" and another for "Directive" values.

MySQL Support	enabled
Active Persistent Links	0
Active Links	0
Client API version	3.23.49

Directive	Local Value	Master Value
mysql.allow_persistent	On	On
mysql.connect_timeout	-1	-1
mysql.default_host	no value	no value
mysql.default_password	no value	no value
mysql.default_port	no value	no value
mysql.default_socket	no value	no value
mysql.default_user	no value	no value
mysql.max_links	Unlimited	Unlimited
mysql.max_persistent	Unlimited	Unlimited
mysql.trace_mode	Off	Off

Figure 11 MySQL installed properly

See the text contained in the screen capture on the previous page (figure 11). One can see that presently there are no connections to the MySQL server at this time.

The text

```
(<? phpinfo(); ?>)
```

on a web page could be downloaded to any web server/service to test to ensure that PHP and its supporting interfaces are installed.