Air Force Institute of Technology

AFIT Scholar

Theses and Dissertations | Student Graduate Works

3-2000

Extensible Markup Language as a Weather Tool

Michael J. Calidonna

Follow this and additional works at: https://scholar.afit.edu/etd

 Part of the Meteorology Commons, and the Other Computer Sciences Commons

Recommended Citation

Calidonna, Michael J., "Extensible Markup Language as a Weather Tool" (2000). *Theses and Dissertations*. 4754.
https://scholar.afit.edu/etd/4754

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact AFIT.ENWL.Repository@us.af.mil.

EXTENSIBLE MARKUP LANGUAGE AS A WEATHER TOOL

THESIS

Michael J. Calidonna, Captain, USAF

AFIT/GM/ENP/00M-02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

20001113 030

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC QUALITY INSPECTED 4

The views expressed in this thesis are those of the author and do not reflect the official policy or the position of the Department of Defense or the U.S. Government.

AFIT/GM/ENP/00M-02

EXTENSIBLE MARKUP LANGUAGE AS A WEATHER TOOL

THESIS

Presented to the Faculty

Department of Engineering Physics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Meteorology

Michael J. Calidonna, B.S.
Capt, USAF

March 2000

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT/GM/ENP/00M-02

EXTENSIBLE MARKUP LANGUAGE

AS A WEATHER TOOL

Michael J. Calidonna, B.S.
Captain, USAF

Approved:

Cecilia A. Miner (Chairman)	Date
Michael K. Walters (Member)	Date
Timothy M. Jacobs (Member)	Date

Acknowledgments

I would like to take this time to thank the people that have helped me through this endeavor. From an academic viewpoint I'd like to thank Lt. Col Miner for her guidance, Lt Col Walters for his insight, and Lt Col Jacobs for his patience. Without the support of these individuals I would not have had either the willpower or the knowledge to complete the task at hand. I'd like to thank my friends for keeping me on track, specifically Jimmy Trigg and Tom Renwick for those necessary sanity checks and those needed respites. I'd like to thank my family. My children for their understanding and unconditional love, my brothers and sisters for their kindness, my mother for, well, for being mom, and my father for loving me so much in life that I can still feel it. Finally, I'd like to thank God for his sense of humor, who else would create both an unpredictable system and people that want to explain it.

Michael J Calidonna

Table of Contents

List of Figures

List of Tables and Screens

Abstract

This thesis is a proof of concept work that will extend the Core Mapping Application Program Interface (CMAPI) components to include weather data. The CMAPI project is headed by Air Force Research Lab (AFRL)/ Information Directorate Information Handling Branch (IFEB) at Rome labs in Rome, New York. This work extends the CMAPI project in two distinct areas. The first goal is to figure out how to overlay and display weather data on a dynamically linked Internet platform. This was accomplished by incorporating existing data from the Air Force Weather Agency (AFWA) into the CMAPI program in a static environment. The other goal is to learn about the Extensible Markup Language (XML) and how it can contribute to characterizing structured data (i.e., weather data output from AFWA). Once this tool can be exploited, a dynamic interaction between the CMAPI program and all AFWA products could be developed. The overall goal is to make it easy for the system, and the application of that system, to ingest and manipulate data.

Extensible Markup Language

As a Weather Tool

I. Introduction

1. Motivation

The motivation for this project arose from the Air Force Research Lab's (AFRL) work with emerging technologies in their Common Mapping Application Program Interface (CMAPI) program. AFRL has been working on projects designed to improve information visualization and web based applications of this visualization. This work has served the intelligence community needs and could be formatted to fulfill some of the needs within the weather community. As such a logical next step was to see if these new technologies, specifically extensible markup language (XML) could be exploited to better serve the needs of both the Air Force and the Air Force Weather Agency (AFWA). AFRL was the driving force behind this project, and a design similar to the CMAPI concept was the goal to strive for. Coincidentally this program acts to achieve objective 5 of goal 2 from the Air Force Weather Agency Strategic Plan, which reads, "Field an automated, worldwide pilot briefing system that meets the needs of Total-Force customers within 24 hours" (Air Force Weather Agency, hereafter referred to as AFWA, 1999).

From an operational viewpoint, this project will enable both the base level and the regional forecasting centers to process information and maintain a higher level of horizontal consistency than is currently available. Each entity will process information in the same manner and utilize the same governing program.

A. AFRL/IFEB Mission Statement

"The Information Directorate Information Handling Branch (IFEB) at the Air Force Research Labs (AFRL) in Rome, New York, seeks the advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer need in the area of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange. The directorate's areas of investigation include a broad spectrum of information and fusion, communications, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information technologies. IFEB seeks to identify, develop, prototype, transition, and support advanced technologies and approaches to the acquisition, analysis and timely dissemination of intelligence information for the intelligence community. These techniques will acquire, assimilate, and disseminate intelligence products needed by decision-makers and warfighters to ensure battlespace dominance. The Common Mapping Application Program Interface (CMAPI) was developed to meet these needs." (Air Force Research Lab, 1999)

B. CMAPI Introduction

"The CMAPI components grew out of a Joint Reserve Intelligence Connectivity Program (JRICP) sponsored effort to prototype a web based component for mapping" (Air Force Research Lab, 1999). Upon reviewing the available products AFRL/IFEB decided to develop their own vendor-independent product that could insulate application developers from proprietary interfaces of off-the-shelf mapping products so they could

write to one interface and leverage multiple mapping products. The CMAPI components achieve that goal.

The CMAPI components are significant for the following reasons:

1. They provide a static, documented, public interface that provides vendor-independent access to off-the-shelf mapping products.
2. They enable application developers to write only one set of mapping calls while being able to take advantage of multiple backside engines (driving programs located outside the users access).
3. They enable sites already operating one supported mapping engine to leverage it across multiple intelligence applications.
4. They leverage component and web technology to maximize ease-of-integration.

2. Problem Statement

This thesis was designed to develop a proof-of-concept program using the extensible markup language (XML) as a vendor-independent, dynamically linked tool to manipulate weather data. It also evaluated this program language for use as a platform independent standard for weather visualization.

3. Scope

A. Thesis Scope

AFRL would like to see if an XML program could be used to develop software that could be accessed from various sources and accomplish the same task each time. This type of software is referred to as a write once, use often type software to enable end users the ability to view and manipulate all types of data from multiple sources. This project focuses specifically on weather data. To achieve this goal two major steps have been accomplished. First, a Document Type Definition (DTD) was developed, and secondly, the scientific data has been transformed into a viewable format.

The scope of this work concentrates on utilizing XML as the Internet programming language of choice. XML will allow any web-based browser to directly interact and manipulate data in an efficient manner. To exploit XML for maximum efficiency a thorough understanding of this language's abilities and limitations is needed. The majority of this thesis will address the development of various tools used in XML for data formatting, ingesting, manipulation and visualization.

XML is an emerging technology that is an extension of the familiar hypertext markup language (HTML). The major difference is that where HTML has only predefined tags for data formatting, XML has very few predefined tags. Tags are a way of assigning variable specific values. In both HTML and XML tags are written inside the "< >" symbols, and a closing tag is preceded by a "</ >" symbol. However, instead of the limited set of abbreviated tags allowed by HTML, the tags in XML may be much more

expressive. For example <BASE_NAME> could have the value "Travis" and is closed by a </BASE_NAME> declaration. XML is a language defined by the user. This makes XML easy to use. XML allows the programmer not only to format the output, but also to manipulate the output in ways not possible using HTML.

B. Constraints

XML is an emerging technology, and thus some of the functions do not work yet on all web browsers. As this thesis is designed to develop a vendor independent tool, Microsoft Internet Explorer (IE) 5.0 is the browser of choice. The areas where IE falls short of the full capabilities of XML are highlighted. The shortcomings of IE 5.0 are easily avoidable through proper manipulation of the data and a thorough understanding of the incompatibilities between XML and IE 5.0.

4. Overview

This chapter outlined the importance of exploiting XML, an emerging technology, to visualize scientific data, specifically the visualization of weather data. This work will be an important step toward the understanding and utilization of XML as the standard for web based manipulation of weather data.

The remainder of this work focuses on four specific aspects of the scientific process. Chapter 2 provides additional background on work that has already been done. Chapter 3 is a detailed account of the methodology associated with this project's development. Chapter 4 summarizes the progress made through this project. Chapter 5 provides recommendations for areas of continued research and further project development. The

appendices provide a glossary of acronyms used, and a complete copy of the code developed.

II. Background/ Literature Review

1. Chapter Overview

This chapter provides the background material for this project. As XML is an emerging technology, the literature on this subject is sparse. Most of the papers written deal mostly with what XML should be able to do, as opposed to what has already been done with it. The fields of study encompass a wide range of topics, from financial reporting, to manipulation of atomic structures, to analysis of baseball statistics. As such, each of the background papers are covered independently and common themes are summarized in the closing comments of this chapter.

2. XML in 7 Points (Bos, 1999)

The information in this paper gives a brief overview of XML. It is intended as a stepping stone to more complicated uses of the language. This article explores seven major benefits of using XML. Each of these points are summarized below.

1. XML is a method for putting structured data in a text file. All computer output is stored as either binary data or textual data. XML is a set of rules designed to produce files that are easy to generate and read, that are unambiguous, and that are platform independent.
2. XML looks like HTML but isn't HTML. Like HTML, XML uses *tags* (words bracketed by '<' and '/>') and *attributes* (of the form name = "value"), but while

HTML specifies what each tag and attribute mean, XML uses the tags only to delimit pieces of data, and leaves interpretation of the data to the application that reads it.

3. XML is written word, but it isn't meant to be read. XML files are textual but only to allow the programmers to debug them. The rules for XML are much stricter than for HTML. HTML will "best guess" what a missing attribute should be; XML will stop processing at the first ill-defined attribute. For this reason it is up to the program developer to either force the data to be in the correct format, or prompt the user when the data is not valid.
4. XML is a family of technologies. XML 1.0 is the parent specification of many optional modules that provide sets of tags and attributes, or guidelines for specific tasks. There is **XLINK** (still in development as of September 1999) which describes a standard way to add hyperlinks and other inter-document relations to an XML file. **XPointer & XFragments** (also still in development) are syntaxes for pointing to parts of an XML document. Cascading Style Sheet (CSS), is the style sheet language, and is applicable to XML as it is to HTML. **XSL** is the advanced language for expressing style sheets. It is based on **XSLT**, a transformation language that is often used outside of XSL as well, for rearranging, adding or deleting tags & attributes.
5. XML is verbose, but that is not a problem. Since XML is a text format it is nearly always longer than comparable binary formats. The main advantages of textual files are their ability to be read and understood by the user. While these files are significantly bigger than similar binary formatted files, the advantages of textual files far outweigh the size of the files. On most modern machines, disk space is really no

longer a problem, and modem protocols can compress files on the fly, thus saving bandwidth as effectively as a binary format.

6. XML provides a new twist on an old theme. The World Wide Web Consortium (W3C 1999) started developing XML in 1996, and it has been a standard since 1998. XML grew out of such scientific visualization tools as standard general markup language (SGML) and HTML. The developers simply took the best aspects of both languages and produced something that is as powerful, yet easier to use.
7. XML is license-free, platform independent and well supported by most Internet browsers. XML can be compared to Structured Query Language (SQL) in the database vernacular, since users still have to build their own database and their own programs/procedures that manipulate it, but there are many tools available. And since XML is license-free, users can build their own software around it without paying anybody anything. XML isn't always the best solution, but it is always worth considering.

3. Various sources

There have been a number of papers published on XML in financial (Hoffman and Kurt, 1999) and quasi-scientific journals (Bosak and Bray, 1999). These articles point to XML as the Internet language of the new millennium (Hoffman and Kurt, 1999). At present computer devices connected to the web can do little more than swap information back and forth. Current HTML protocols convert your computer to an on line fax machine. In other words, the information is transferred between systems, and the

computers simply swap information back and forth, not allowing any data manipulation. The computer only sends and receives pages, one at a time (Garber, 1999). XML will allow a great deal of processing on the spot. It does this through embedding objects within the source code of the document to be viewed. In fact the only limitation is the programmer's ability. XML lays down the ground rules that clear away a layer of programming detail so people with similar interests can concentrate on the basic part – agreeing on how they want to represent and visualize the data they're exchanging. To do this the following must be agreed upon:

1. Which tags will be allowed
2. How elements may be nested within one another
3. How data should be processed

The first two are typically codified in the Document Type Definition (DTD). XML is designed to make information self-describing. HTML uses predefined tags, which generally have little meaning. For example: a **<p>** tag within HTML means a paragraph; within XML you can use **<p>** to represent anything you want. Additionally you can use the word <paragraph> (or <Satellite Image>, or <Forecast>) to represent exactly what that tag means. " The current problem with web based technology is not the modem or connection speed, it is often that speed of light networks often move along the information highway in the slow, if not disabled, lanes" (Bosak and Bray, 1999). This problem arises from the fact that HTML was never designed for dynamic interactivity (Pardi, 1999). Generally you have to access a page, find what you want, go to the nest page, fill in information, and continue ad nauseam. This also assumes everything is written in the same language (English, Spanish, etc.).

XML provides part of the solution. XML requires tags to come in pairs. In other words <Satellite Picture> must always be accompanied by </Satellite Picture>. The first opens the tag and the second closes it. This may seem trivial but it allows XML to nest tags in a tree like structure. HTML also has this feature but in a very limited capacity. HTML has only basic hierarchical nesting. This XML nesting structure then follows certain inheritance rules needed for data manipulation. Secondly XML relies on a new standard called a Unicode (Bosak and Bray, 1999). In short this allows text to be written in most of the world's languages. The importance of this becomes clear when we try to retrieve a forecast or observations from, say, China. One of the first things declared in an XML document is the language it is written in. Since XML provides for different languages, the characters written in Chinese now have meaning when a program ingests them in English.

A thorough understanding of HTML is not necessary for programming in XML. XML is designed to be a stand-alone system capable of formatting, manipulating and viewing most types of data formats (McGrath, 1999). Using XML in this capacity limits the complete exploitation of the World Wide Web. XML should be seen as an extension of HTML, keeping the good aspects of data formatting and imaging, while extending the capabilities beyond their current ceilings. The advancement of HTML and scripting languages, specifically JavaScript (Holzner, 1999), allows the programmer to easily process large data sources to manipulate and extract needed information.

4. XML Bible, XML by Example, and XML in Action

These books are the main source for learning XML as a language (Harold, 1999; McGrath, 1999; Pardi, 1999). The authors take a step-by-step journey through the process of what XML should be able to do. Their knowledge came about through statistical research of various data mediums to include baseball statistics, financial information, and basic web publications. They saw the complexity of inputting large amounts of data into HTML and not being able to directly manipulate the data without overtaxing a network server. It should be pointed out that the weakest link in the data stream is communication with the server. Stand-alone machines do not have to compete with a slow communications network. The more a system can do independent of a network, the more efficient it is. The ultimate goal of XML is to make systems more efficient. While these authors did not develop the standard for what typical XML documents should look like, they consistently point out how to develop a well-formatted XML document. These books are referenced throughout the methodology section of this thesis.

5. Summary

It is obvious from all papers available that in order to exploit the Internet some form of standardization is required. Incompatible software problems run rampant through the worldwide web and detract from the ability to manipulate various data formats. HTML has served as the standard for some time for most of the Internet home

pages. While this serves its purpose as a formatting tool, it falls short of most scientific applications. SGML was then developed to allow scientific visualization of complicated data sets. Again, SGML serves its purpose but is rather limited in scope. While different programming languages, e.g., Java or Java3D, can help they do not alleviate the problem of Internet accessibility. W3C then decided to develop a language to serve all purposes. XML became the language of choice. To be of maximum use, XML must allow all types of data to be input and manipulated. The broader the range of input allowed, the more powerful the language. W3C decided to allow the users to format any and all data using XSL style sheets. This allows the user to define all tags associated with the programs they are using.

III. Methodology

1. Chapter Overview

This chapter provides a detailed description of the thesis work. A description of how the program works is provided, but the emphasis is on how the weather community could exploit XML for data dissemination. As stated earlier, this thesis is a proof-of-concept designed to use XML as the primary data manipulation language. The goals of this work were to learn and exploit XML in a web based application. This was accomplished using two approaches, creating a stand-alone XML document, and combining XML, HTML, and JavaScript. While the first method was necessary for understanding how XML works, the second method produced results more in line with the needs of the Air Force. A discussion of these methods, including the strengths and weaknesses of each, is provided.

A necessary step to understanding how to best use XML is gaining some rudimentary knowledge of HTML and some of the basic terminology associated with publishing a web based application. Once this is in place, an understanding of what data will be manipulated is needed. For this program the data will be textual only. The next phase of the application is to see how XML uses the data and what properties are associated with each step along the way. The final product is displayed using some tailor-made JavaScript in an HTML format.

2. *Basic Terminology*

An explanation of terms like elements, children, nesting, inheritance, and embedding follows.

A. Elements: An element is simply a piece of information within the program's structure. One of the benefits of XML over HTML is that elements can be named, or tagged, in a way that makes more sense to readers of the code. For example a base's name can have a tag called " BASE_NAME", such as in Figure 3-1. Elements can then be placed in a tree like structure where the topmost element is referred to as the root element.

```
<BASE_NAME> Wright-Patterson </BASE_NAME>
```

Figure 3-1 Element

B. Children: All elements in the structure that fall underneath the root elements are children of that element, as the example in figure 3-2 shows. Children elements may have children elements of their own adding to the tree like structure of the data. This structure is prevalent in most programming languages. Until the development of XML this structure has not been nearly as easily defined in web-based applications.

```
<BASE>
   <BASE_NAME> Wright-Patterson </BASE_NAME>
   <BASE_LOCATION> Ohio </BASE_LOCATION>
</BASE>
```

Figure 3-2 BASE Element with two children

C. Nesting: Grouping elements together in a logical manner is nesting. This is an important aspect of the program because it allows the user to follow the data structure through some logical progression until the desired element is reached. Figure 3-2 shows elements BASE_NAME and BASE_LOCATION nested under the BASE root element.

D. Inheritance: Just as in physical families, where children acquire certain traits from their parents, child elements in XML inherit their parent's attributes. For example, if BASE is the root element, and it is defined to be displayed in a bold format with a font size of 20, it is necessary to program the computer to do just that; the code looks like this: BASE {font-style: bold; font-size: 20pt}. The benefit of inheritance is that all children elements of BASE will now be displayed as bold 20 pt. From a pilot's viewpoint it would be effective if all weather tagged with specific information (perhaps a ceiling below 200 ft) was displayed red. From a programmer's side it is now easier to force all children elements to conform to a specific input parameter.

E. Embedding: The basic premise behind embedded objects is keeping data available to the computer but hidden from the user. The user may access this information through some graphical user interface; for example a "click here" button will display a list previously hidden. The ability of XML to embed information allows the user to quickly access and manipulate all information available to the computer.

3. Using XML

XML is designed to complement HTML, not replace HTML entirely. To begin the process of understanding how to best manipulate XML, a stand-alone document was written. This was done solely using XML as the data source and, initially, a cascading style sheet (CSS). The XML Bible (Harold, 1999) uses baseball statistics as an initial example. To follow the author's logic similar textual weather data was used. Prior to explaining the utilization of XML a brief explanation of style sheets is in order.

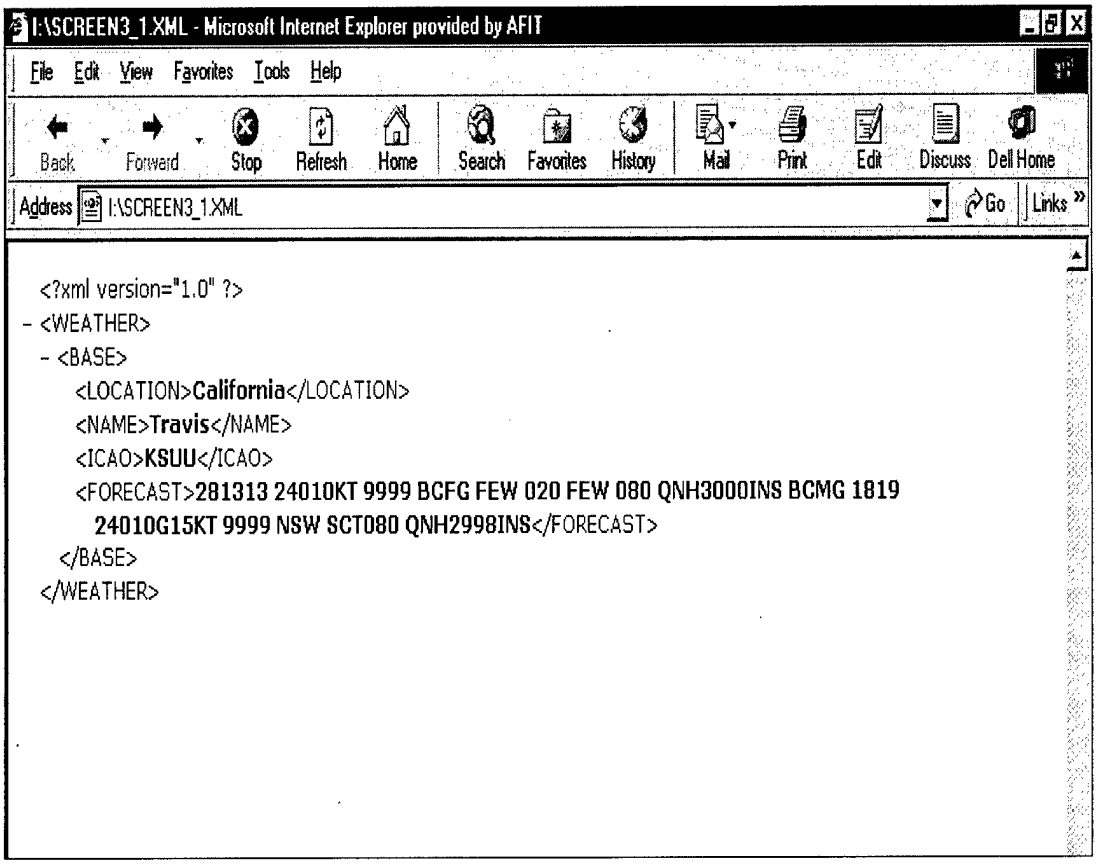
XML stand-alone documents are written with three types of files. The fist file is the XML document itself. This is the controlling document and contains all the elements needed for the output. The second type of document is a style sheet. There are two basic types of style sheets, a cascading style sheet (CSS), and an extensible style sheet language (XSL). The final type of file is a document type declaration (DTD). XML does not need all three types to produce output; however, each type of file has strengths and weakness associated with it.

XML documents with no attached style sheets will simply display the program input. Figure 3-3 shows a basic XML document with no attached style sheet, and screen 3-1 shows that same program as displayed with IE5.0. Notice the first line of the XML document tells the browser what it is looking at, in this case XML version 1.0. Additionally the browser uses the default settings to display the data, in this case notice the text is bolded.

```
<?xml version ="1.0"?>
<WEATHER>
  <BASE>
    < LOCATION>California</ LOCATION>
    < NAME>Travis</ NAME>
    < ICAO>KSUU</ ICAO>
    <FORECAST>
    281313 24010KT 9999 BCFG FEW 020 FEW 080 QNH3000INS
    BCMG 1819 24010G15KT 9999 NSW SCT080 QNH2998INS
   </FORECAST>
  </BASE>

</WEATHER>
```

Figure 3-3 XML Sample code



Screen 3-1 Web page with no attached style sheet

This type of file does not produce any useful output. It does highlight some differences between XML and HTML. The first difference, and the most important, is that the data is stored in tags that the programmer designs. The tags can be named for what they represent for example, the name of the base is in the "BASE_NAME" tag. This is different from HTML, which only has a set number of predefined tags (Darnel, et al. 1997). The next difference is XML requires a closing tag. When programming in HTML the browser often knows when the predefined tags should end. In XML this is not the case. All elements must end in a "</ELEMENT_NAME>" tag; not having this

closing tag will produce an error on the page. Unfortunately, most browsers offer little guidance as to what the error may be; they simply will not publish the document. Also notice the root element, in this case <WEATHER>, contains no textual data. This is the element that completely contains all other elements. The choice of <WEATHER> as the root element will allow additional <BASE> elements to be added at will.

Although necessary and instructive, the XML document alone will not produce a useful web page. To do so, some form of style sheet is needed.

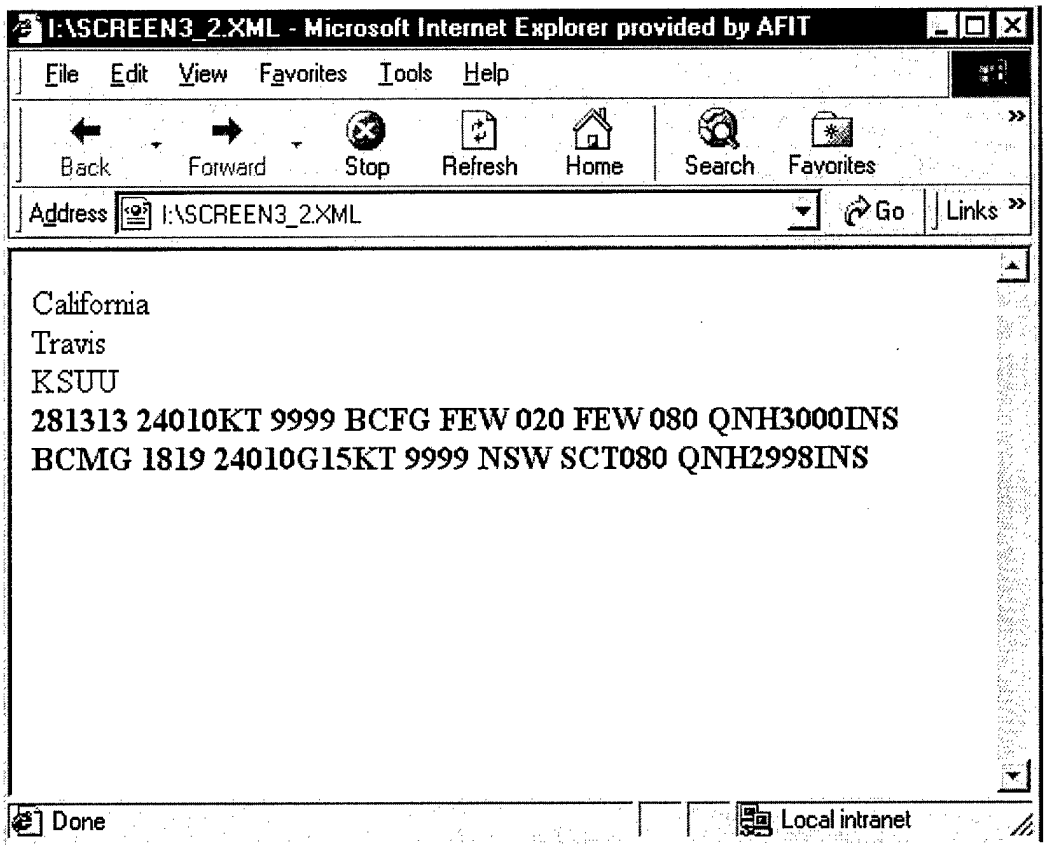
CSS's are the most basic types of style sheets. They contain information that tells the browser how, and where, to display certain types of data. Figure 3-4 attaches a CSS to Figure 3-3. Figure 3-5 shows the CSS, and Screen 3-2 shows the resulting output. This overrides the browser's default settings and only displays the data associated with each tag.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="3-1.css"?>
<WEATHER>
 <BASE>
   <ICAO>KSUU</ICAO>
   <NAME>Travis</NAME>
   <LOCATION>California</LOCATION>
   <FORECAST>
   281313 24010KT 9999 BCFG FEW 020 FEW 080 QNH3000INS
   BCMG 1819 24010G15KT 9999 NSW SCT080 QNH2998INS
   </FORECAST>
 </BASE>
</WEATHER>
```

Figure 3-4 Sample XML code with attached style sheet

```
BASE, ICAO, NAME, LOCATION  { display:block }
FORECAST {font-weight: bold; background-color: Window }
```

Figure 3-5 Sample style sheet



Screen 3-2 Web page with attached style sheet

While the previous example may seem rather simplistic it highlights some of the advantages of using XML with cascading style sheets.

1. XML has no predefined tags. All of the data is self-describing.
2. The CSS can be anywhere. Notice in Figure 3-4 the second line reads "<?xml-stylesheet type="text/css" href="3-1.css"?>". An "href" is a specified uniform resource location (URL) and is used the same as in HTML. The "href" could be a style sheet located anywhere, accessed by numerous programs having the same tags. These tags are defined as either an absolute location, one that gives the exact path to where the information is located, or a relative location. This is a relative location; in other words this file is on the

same computer, in the same subdirectory as the XML file; as such the "href" line contains only the file name. If this were an absolute location the "href" would contain additional information, perhaps a URL or a location in another directory on the same computer. A style sheet for displaying all forecasts could be stored in one location and every forecast could access it. This would ensure every forecast is displayed in the same manner and uses the write once used often anthem present in computer sciences.

3. Adding additional bases is simple. Either create an additional XML file or simply add additional bases to the existing file.
4. Notice the FORECAST tag has a font-weight attribute of bold in the style sheet. Any information within this tag will be displayed bolded. In fact the information can be displayed differently by accessing only the style sheet and leaving the XML data alone.

CSS's can format information in all ways that HTML can. Using CSS's allows the information to be displayed as required but does not allow for data manipulation. The more advanced XSL allows more to be done with the data.

XSL operates in two different modes, formatting and transformation. Each of the modes operates independently from the other. It is important to note that XSL is the bleeding tip of this cutting edge technology. Not all browsers support XSL entirely, and the standardization is still in the development stage. The program written for this thesis started using XSL as the primary method for data manipulation but quickly ran into displaying problems. The formatting aspect of XSL is fairly easy to comprehend and serves as an extension of CSS. The program attempted to emulate programs written by

Elliotte Harold (Harold, 1999) but ran into compatibility problems. The programs as written in XML Bible could not be reproduced using IE 5.0 on five different computers both at personal residences and government owned computer labs. The programs were run directly from the CD provided with the book, and still they could not be run on either IE 5.0 or Netscape Navigator 4.7. The problem had to do with IE 5.0 and Netscape not being able to process some of the XSL transformations. The solution seemed to be using a third browser called Mozilla. This method was not used because Mozilla is not standard on government computers. At this stage of development it was thought that XML had some benefits but fell short of initial ideals. The last road to explore was the DTD's.

DTD's provide a list of elements, attributes, notations, and entities, along with their relationship to one another. This is the part of the file that tells the computer what to expect and places limitations on those expectations. DTD's are useful for validating an XML file. They make sure all tags fit within a pre-defined container. DTD's do little for data manipulation. With this in mind a new approach was taken. This approach consisted of incorporating XML into an HTML file.

The full functionality of XML was not exploited using only XML type of files. Incorporating XML into an HTML document seemed the next logical step. Using HTML by itself would not advance the needs of the Air Force. Combining JavaScript with HTML really took advantage of the functionality of XML. While this is easy to implement, it is difficult to fully exploit.

The program in its entirety is located in appendix B. At this point the program will be described as it was written.

The biggest benefit to using XML is the ability to define any tags needed and make these tags self-describing. The most effective way to do this is to understand the structure of the data to be defined. In this example only the base ICAO, name, location, and forecasts are displayed. All of this data is simply text, and there have been no limitations on what data can be input. There needs to be some logical breakdown of the data. In this example the root element *BASE* has been given the children *ICAO*, *NAME*, *LOCATION* and *FORECAST*. To further illustrate the point the child *FORECAST* could have two children: one tagged *TAF*, to represent the normal terminal forecast; and one tagged *AMD*, to represent an amendment to the forecast. XML can then force the data to be in a very specific format using the format tags associated with entity references.

Entity references are designed to allow the computer to process input in very specific ways. This example has only included character data. In general anything inside of the brackets (<>) the computer treats as markup language and anything outside is character data (CDATA). Most of the times this is what is wanted; however, there are a few exceptions to this rule. The less-than and greater-than signs must be treated as a special case. These are input as "&lt" and "&gt". The general rules associated with this syntax are referred to as "well formed." A well-formed XML document adheres to these eight rules (Harold, 1999):

1. The XML declaration must begin the document.
2. Elements that contain data must have both start and end tags.
3. Elements that do not contain data and use only a single tag must end with a />.
4. The document must contain exactly one element that completely contains all other elements.

5. Elements may nest but may not overlap.
6. Attribute values must be quoted.
7. The characters "<" and "&" may only be used to start tags and entity references respectively.
8. The only entity references which appear are "&"; ">"; "'"; and """.

It is important to note these rules do not apply to HTML. For example HTML documents do not need close tags in all cases. While most browsers will process an HTML document that is not well formed, this project adheres to these eight rules for both the XML file and the HTML file.

The next phase in the program was developing a document type definition (DTD). To accomplish this a complete list of all elements is needed. Table 3-1 contains an example of some of the elements needed to display a forecast; this is not meant to be a complete table, only a portion of the data that could be included.

ELEMENT	*Element it Must contain*	*Elements it May contain*	*Element in which it must be contained*
Catalog	Base		
Base	Name, Location, ICAO, Forecast, Observation		Catalog
Name	Text		Base
Location	Text		Base
ICAO	Text		Base
Forecast	TAF | AMD		Base
TAF | AMD	Valid time First time group	Second time group Third time group Etc	Forecast
Valid time	Text		TAF | AMD
First time group	Clouds, visibility, winds, cat	weather, Remarks	TAF | AMD
Clouds	Text	Ceiling	First time group
Visibility	Text		First time group
Winds	Text		First time group
Cat	A | B | C | D | E		First time group

Table 3-1 partial DTD

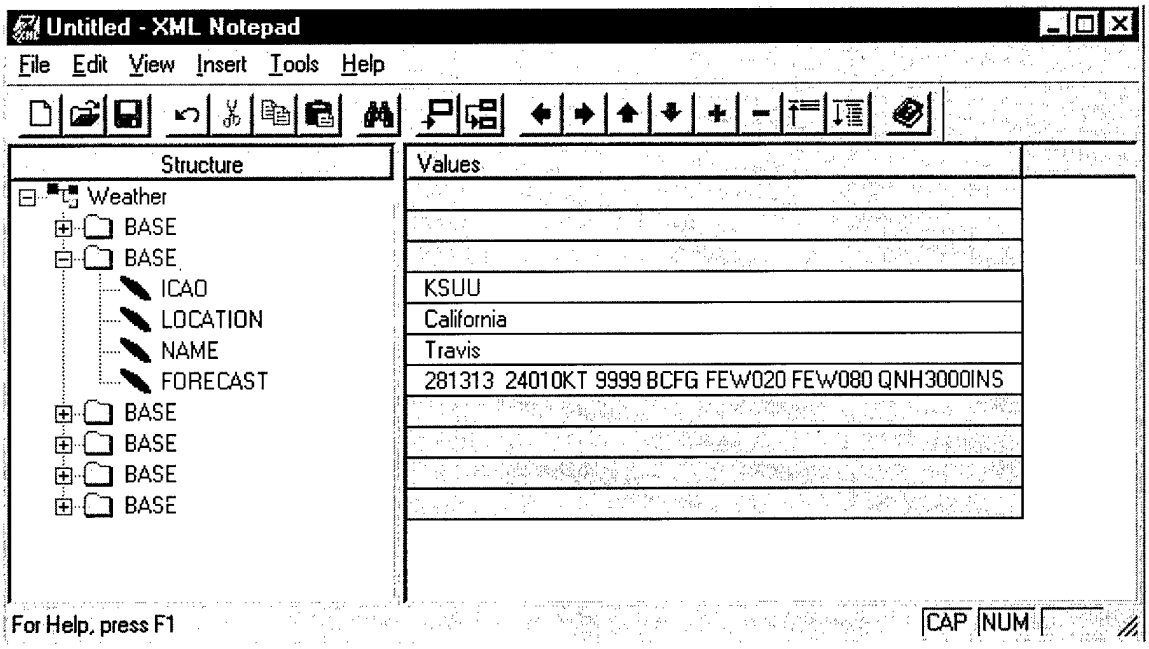
Now that the elements have been defined, the XML code can be written. The options at this point are to write one file for each base and collect them at one central point, or to write one extensive program containing all bases. For this effort one program was written. Experimentation with multiple files proved possible but more an exercise in JavaScript manipulation. For this program five bases were chosen as representative, and the forecast was limited to one textual block as it is currently displayed through Air Force Weather Information Network (AFWIN). An XML notepad was used to initialize and populate the database. One hundred bases were input to prove it could be done. To accomplish the same feat in HTML, the source code would have to be changed. The bases would have to be input and formatted individually. To add these bases in XML, they were simply added with the proper tags. Once assigned the proper tags, a single formatting statement ensured uniformity. This highlighted a basic benefit of using XML code.

The program is designed to simulate a pilot preparing their flight plan. This is a fulfillment to AFWA's strategic plan goal 5 (AFWA, 1999). Part of the planning process involves requesting weather for take-off, landing, and alternate locations. When the weather is fine (ceilings above 3000 ft, visibility above three miles, winds less than 15 knots) this part of the process is easy. When the weather is less than splendid it is time consuming for the pilots to search for bases that meet their specific weather requirements. AFWIN has very similar capabilities. The difference is that all the information is contained within the program accessed. Current programs simply access different pages to acquire different bits of the information. This program allows the pilot to search many

bases in an efficient way. As the technology progresses the program will be able to find bases based on specific criteria.

This program contains two important files. First there is the HTML code. This tells the browser what to do. Part of this HTML code is a JavaScript, which tells the computer to open certain XML files and embed that information within the HTML file. The other important file is the XML code. This is where all the actual information is stored. The benefit of storing the data separate from the source code is that any change to the data does not require a change to the code itself. As long as the tags are the same, name information can be added either manually or automatically. The following section demonstrates how additional bases could be added. A program called XMLNotepad was used to manipulate the XML code as shown in screen 3-3.

This example will demonstrate how, with the proper tags, information can be added to a web page without changing the HTML source. Using appendix A as the XML source code, the code for the HTML is located in appendix B. To add another base to this code another element is added with the tag <BASE> and all the same children. As the HTML code cycles through the XML document it simply finds the new <BASE> tag and processes it like it did the previous one. This process is continued until no new child elements are found. The ease with which new elements can be added maximizes the utility of using XML as the preferred data storage medium.



Screen 3-3 XMLNotepad

XML is still in the development stage. All of the capabilities of this language cannot be exploited. Microsoft Internet Explorer 5.0 offers most of the functions; however, it falls short in a few key areas, for example in trying to incorporate the !ATTLIST function. This function is designed to describe attributes of an element within the DTD. The use is <!ATTLIST `Element_name Attribute_name Type Default_value`>. Where `Element_name` is the element possessing the attribute, `Attribute_name` is the name of the attribute; Type is the kind of attribute (one of the 10 in appendix D); and `Default_value` is the value the attribute takes on if no other is specified. An unspecified error was encountered each time the !ATTLIST function was used. Currently to incorporate this function the Mozilla browser could be used. This is just one example of where the idea of XML has not caught up to the application of XML. The work around to this and other problems was to be the use of some scripting

language. This program uses JavaScript to accomplish all of the data manipulation. A brief explanation of the program is now in order. Refer to appendix B for the code itself.

4. The Program

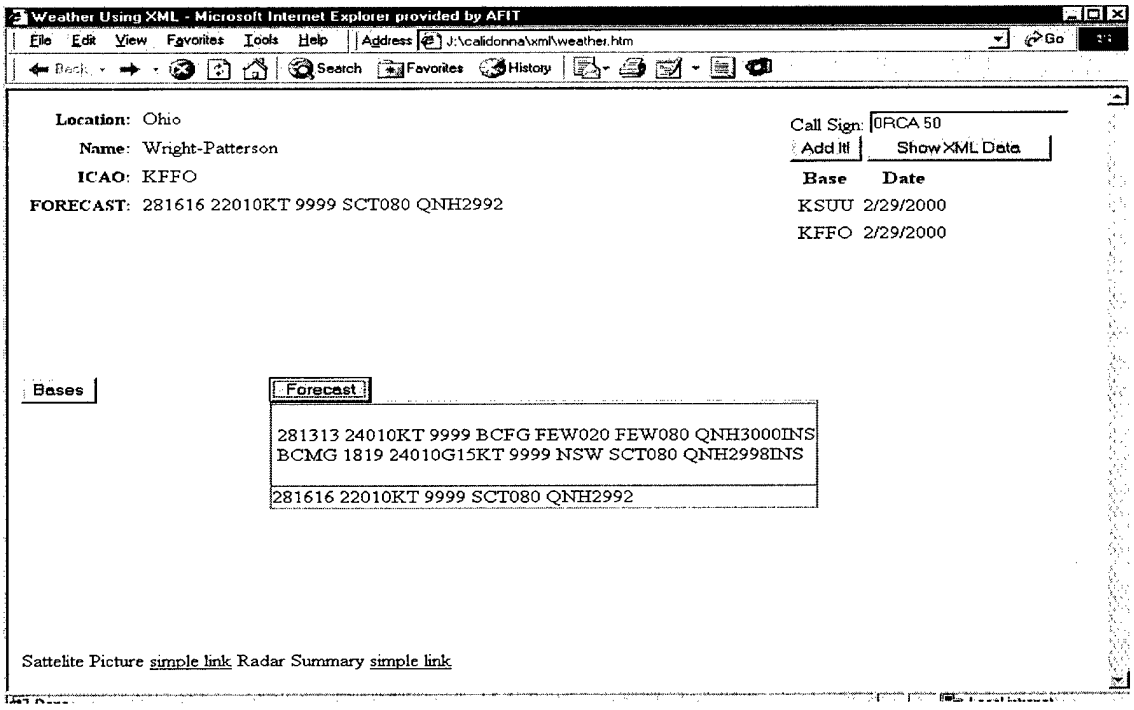
The first part of the program is simply the declarations. It is in this portion where the browser sets the standards. The next declaration tells the browser to use JavaScript as the scripting language. The various files are then loaded into the system and given distinct variable names. This is where the XML files are loaded. The next section contains a number of functions responsible for displaying and manipulating the data as needed. These functions are discussed in the next few paragraphs. The final section, the <BODY> tag, is really the controlling area of the program. This is where the display properties are established and the buttons are described. This is also where the functions are called. These functions are explained in the order of their appearance.

The first function call is to the "updateList ()" function. This function will check to make sure the input, in this case the aircraft call sign, has been entered. If no value has been added an alert message will appear. If a value has been added this function calls the "addtoList ()" function. The "addtoList ()" function will check to see if the call sign is a new one, or one that has already been input. This function was the trickiest to get just right. The basic premise is that a pilot will enter a call sign and select forecasts from various bases. This function has been designed to be as smart as possible. It checks to make sure a base has been selected, it makes sure bases are not selected twice, and it only displays the forecast for that call sign. For example, if Orca 50 is taking off from Travis AFB and going to Wright-Patterson with alternates of Scottt AFB and Tyndall AFB, only

those four forecasts will be displayed. This function walks through the main XML source document, in this case Mr_Cat.xml, and gets all the appropriate information. This function then writes the information to the new file, in this case dash-1.xml.

The information from the new document can then be displayed. This function calls the "showList ()" function. The "showList ()" function incorporates the new data in the dash-1.xml file into a table. This table shows all bases requested for the active aircraft call sign. It checks again to make sure bases haven't been selected twice. This function only shows the base ICAO and the current date. The next button, the ShowXML data button, was incorporated from a program written by William J. Pardi (Pardi, 1999). Its basic design is to simply display the data being written to the XML file. The Bases button activates the "doMenu ()" function. The "doMenu ()" displays the entire list of bases in the original XML file. For the purposes of this program the number of bases has been limited. The bases could then be broken into specific theaters and tailored to some specific criteria. This function displays a table with all the bases' ICAO's, then writes out the forecast of the highlighted base. Color could be integrated into this aspect to show certain set minimums, ceiling, visibility, category, etc. The final few lines are simple links written as any link would be in HTML.

Screen 3-2 shows the final screen for aircraft call sign Orca 50 flying from Travis AFB, CA (KSUU), to Misawa AB, Japan (KKQQ), with an alternate Kadena AB, Japan (RODN).



Screen 3-4 the final product

The program as designed contains five possible bases to choose from with the entire forecast written out in one textual block. Adding additional bases with textual forecasts is simply a matter of replicating applicable tags and inputting appropriate values within the XML source code. From the XML viewpoint adding additional lines to the forecast to point at specific time groups is also very easy. The difficulty arises when the data sets are written in an HTML format. Additional work and potential endeavors will be discussed in the subsequent chapter.

5. Summary

In this chapter the basics of XML were reviewed. From this knowledge it was shown that developing a well-formed XML document was a matter of following eight rules. Once the data to be ingested has been defined, XML provides various mechanisms for processing that data. Using a JavaScript language it was shown how an XML document could be traversed to glean pertinent information from a lengthy data set.

IV. Results Analysis

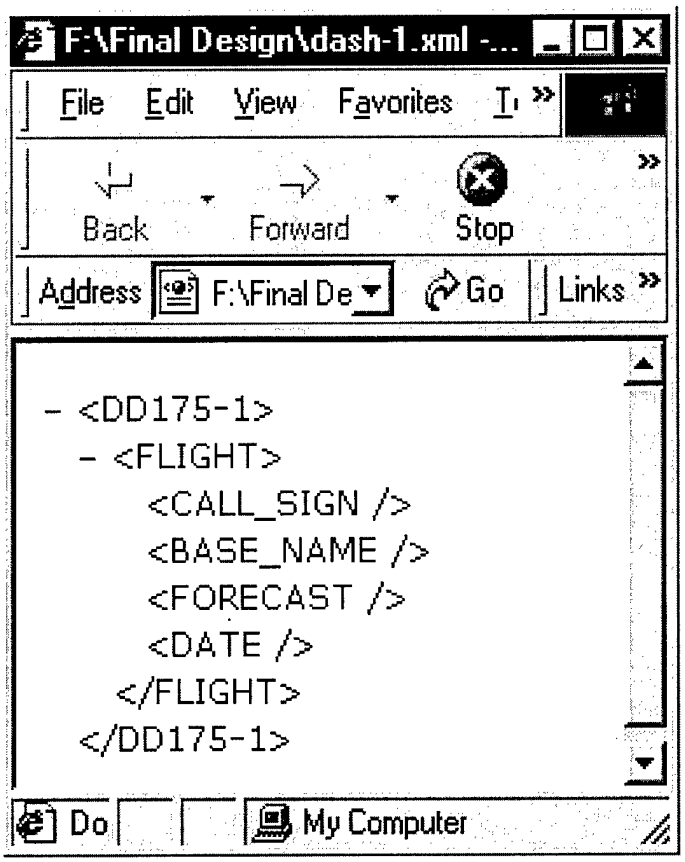
1. Chapter Overview

This chapter provides a proof that XML can be exploited to view and manipulate weather data. It also provides a description of where current technology limits the uses of XML, along with some work arounds for those same problems. Most importantly it shows proof that, with proper forethought, XML can improve the way AFWA ingests and displays data.

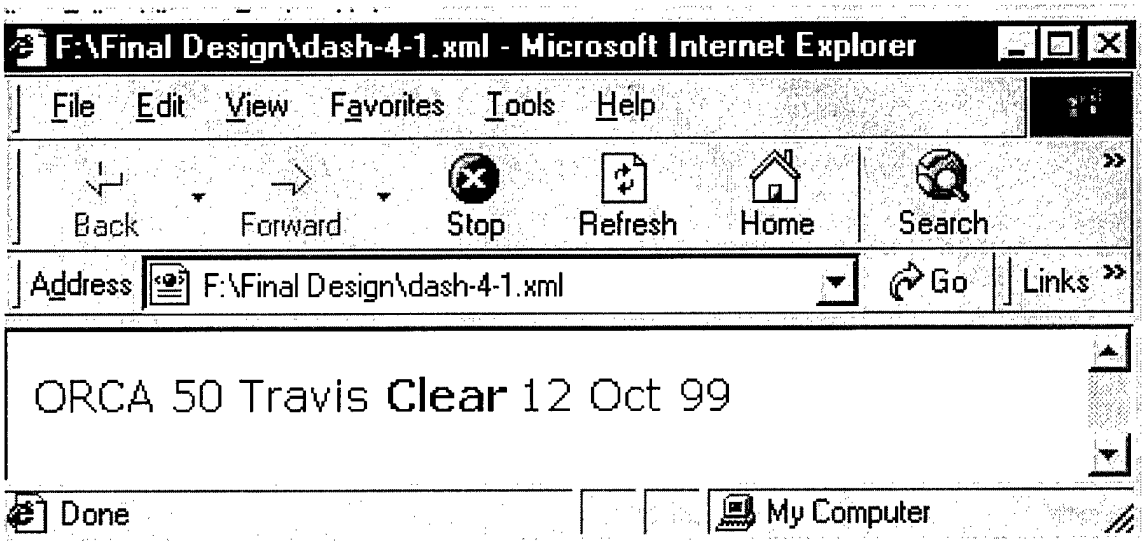
2. Results

The most obvious result obtained was that yes, XML can be used to display weather data. This can either be done in a stand-alone XML document, where the data is all stored in XML files or linked to either CSS's or another type of document called extensible style language (XSL), or through some scripting language. Both of these processes were researched and JavaScript was chosen as the scripting language. The stand-alone XML document types are difficult to understand and not very extensive in their ability. The scripting language allows the program to transcend and manipulate the data thoroughly.

Stand-alone documents are best used for very simplistic programs. XML documents are displayed as simply text without any style sheets attached. Screen 4-1 shows an example of an empty dash-1.xml document with no style sheet attached. Screen 4-2 shows the same code with a style sheet attached. It should be noted that the style sheet used for screen 4-2 is the same one used for screen 3-1.



Screen 4-1 Dash-1.xml output



Screen 4-2 with the style sheet

All browsers can manipulate basic XML documents. At this level of complexity it is nearly impossible to see the advantages of using XML over HTML. In fact it becomes apparent that displaying data is often times more difficult using XML. The benefit of XML comes from its fluidity and the ability to incorporate other programming languages.

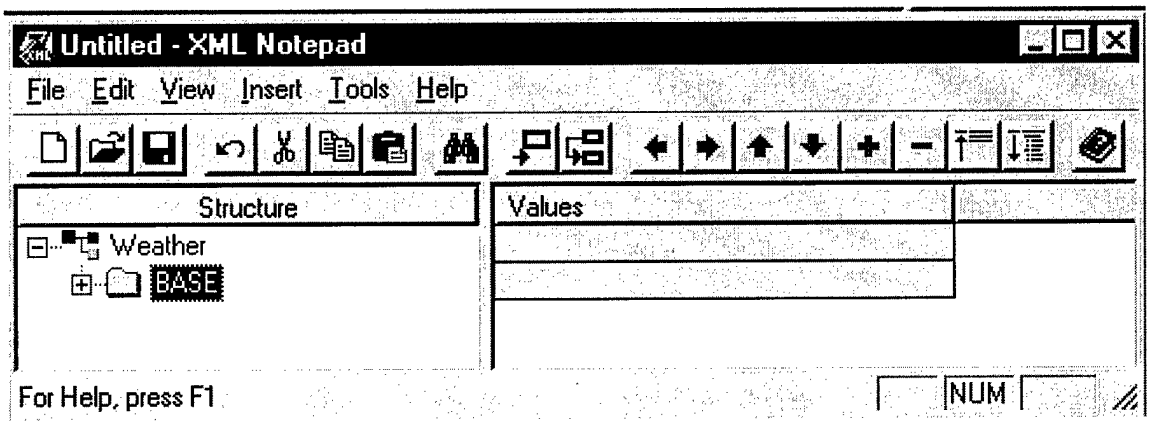
This program uses JavaScript. It could have just as easily been programmed using a computer language such as C++ or Java. JavaScript was chosen due to the ease of integration within an HTML document. The following results are based on the program as shown in appendix B.

The exploitation of emerging technologies is bound to keep the Air Force running smoothly throughout this century. However, as new technologies emerge there is, at times, an associated paradigm shift. The decrease in overall manning and the emphasis shifted to a smaller more well-rounded work force has, as of late, required forecasters to become experts in many, non-weather areas. For example, at base weather stations a forecaster is often sacrificed to additional duties, which include developing local web pages. At this point, however, the work force is reaching saturation, and standardization is key to reducing workload, both for the forecasters and the customers. With one standardized format every base could have the same web page, displaying the same information in the same format. A pilot flying through the system would not have to waste time figuring out many local home pages at the cost of flight safety. To further exploit XML, the hub concept could generate one database for all areas they are concerned with and display that information in one click of a button. Finally, with the demise of the automated weather dissemination system, or at least the proposed demise of

said system, AFWA is becoming increasingly reliant upon web based technologies for data transfer.

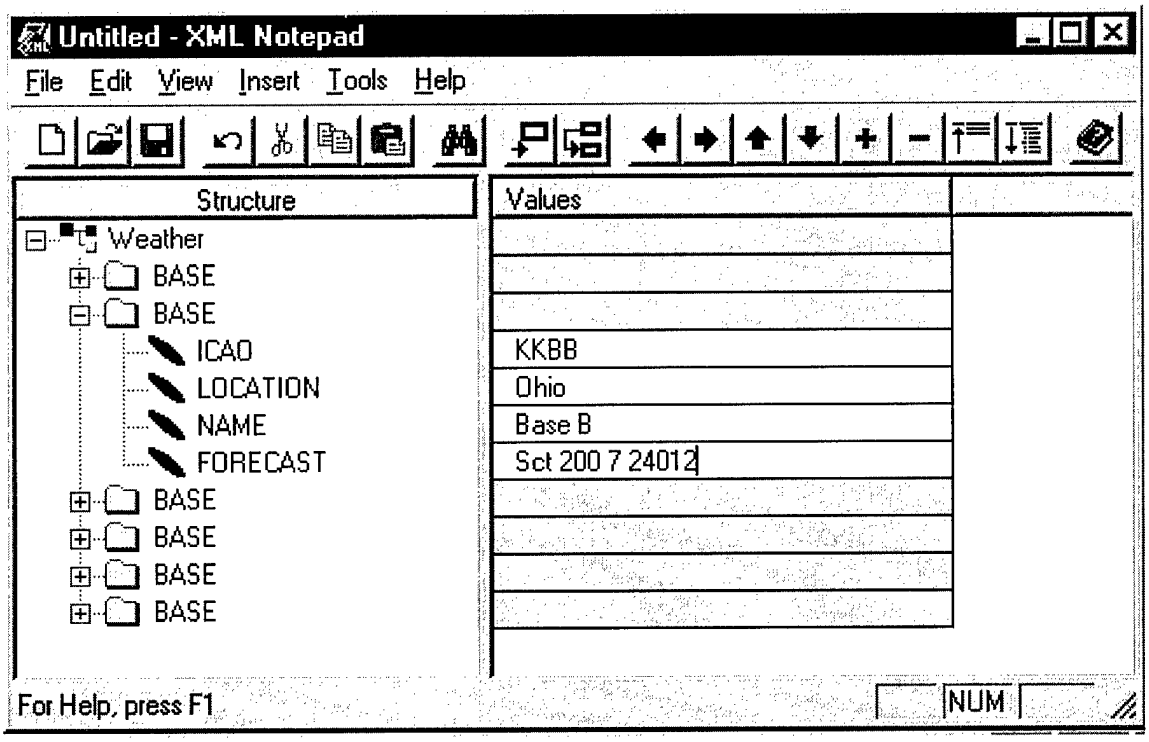
This effort demonstrates one such web based program for data delivery using XML for maximum control. The following is a simulated example of some possible features.

1. The Hub at Scott AFB is responsible for putting out forecasts for bases A-G. Team 1 forecasts for bases A, B, & C. Team 2 handles the others.
2. The code to display the data is similar to that in appendix B.
3. The following XML code is generated and available on a LAN system in Microsoft XMLNotepad, as displayed in screen 4-2.



Screen 4-3 a sample XML document

4. Under the Tools menu there is a function to duplicate additional child elements so as many bases are added as needed.
5. The weather is input into the proper tag as shown in screen 4-3.



Screen 4-4 a sample with 5 bases

6. This file is saved and loaded into the proper HTML document.
7. Any change in the forecast, that is additional bases, amended forecasts, or new forecasts, are done in the XML document. The HTML code is never changed.

This is exactly how this program was developed. The pilot, or forecaster, can then request any bases needed and display those queries as needed.

3. *Summary*

This chapter showed that XML could be used to display weather data in a textual format. The stand-alone type documents and the more complicated embedded files were discussed. It was shown that through proper data definitions and the correct use of standardized tags, along with a thorough knowledge of a scripting language, data

could be manipulated to suit the needs of the user. Overall it is obvious that XML is a more efficient way to handle tree-like input.

V. Conclusions/Recommendations

1. Conclusions

The results presented in chapter 4 support the use of XML as an extension of HTML. The ease with which XML data can be manipulated far exceeds the current limitations placed on HTML. Again, it is important to note that XML is not meant to replace HTML, only to complement the functionality of HTML. The most useful programs were developed using a combination of HTML, XML and JavaScript.

XML will eventually have enough capabilities to be used throughout the weather community. The current status of these capabilities does not fully support widespread implementation. It is important to note that XML is still in the development stage. In fact, the guidance received from AFRL at the project's inception is different than the guidance received at the conclusion. The basic premise that less is more applies to the rules of XML. With only eight governing rules the possibilities seem limitless. As the technology of the language and browsers continues to advance, it would behoove the Air Force to develop programs using this tool. In particular, XML has shown utility in the access and display of weather data. The Air Force Weather Agency can use XML programming to enhance the data delivery through web-based applications. AFWA clearly saw this as a need and made this one of its goals. XML shows the ability to see those goals to fruition. Through proper manipulation XML can eliminate the need for multiple data delivery systems and streamline the dissemination process through a pre-existing medium—the Internet.

2. Recommendations

The ends to which technology may take us are impossible to predict. If XML stays in its current form or at least pretty similar, improving on this initial concept would be cost effective. There are at least three areas to consider in the near future. First, a quality assurance routine should be developed. Once there is confidence in the data, AFWA would be well served to develop one standardized web page for every weather flight. Finally an upper air routine could be developed to store the basis for all upper air plots.

The first avenue of exploration would be along the quality assurance path. Obviously flight safety is of paramount importance, and a good starting point is to ensure the pilots are given the most current data. This could be accomplished by placing time group tags within the source document and only accessing the latest one. This could also be accomplished through some other type of governing tag, perhaps an enumerated type, again ensuring only the latest one is displayed. The other quality checks could be incorporated as the browsers gain capabilities, for example, forcing the wind direction to be between 0 and 360 degrees using the !ATTLIST attribute. Inherently these quality checks are difficult to initially program; however, once in place they need never be changed, again exploiting the write once, use often concept. Once confidence in the system has been achieved Air Force wide dissemination is possible.

Given a known system with predefined tags for every type of input, development of a standard local weather web page, dynamically linked to the central hub, is possible. This

would improve the base weather station operations in two distinct ways. First, it would free up a forecaster from the additional duty of web master and put that individual back to station operations. This would alleviate some of the stress put on forecasters for duties not associated with station operations. Secondly, it would assure pilots continuity throughout the system. If every base had its weather displayed in the same format with the same options, pilots could spend less time searching for data and more time focussing on the mission-planning phase.

The final recommendation is to incorporate an XML program into the upper air plotting process. The data from soundings could be stored in an XML file with tags unique to each level. The mandatory reporting levels could have one set of tags and the significant levels another. Once the data is stored in this format it is possible to generate code to display the flight level winds from a three dimensional aspect. This would require some additional programming in languages like JAVA3D or OpenGL. This certainly would highlight the utility of predefined tags. For example, many different programs could access the 800-mb tag for multiple purposes. The only limitation to this would be the programmer's ability to process the data.

3. *Summary*

In conclusion, this proof-of-concept effort demonstrates the usefulness of XML in data manipulation and dissemination. The current limitations on exploiting XML are a function of the language's newness. Strong support from the Worldwide Web Consortium ensures XML will mature into a new standard of data dissemination. The Air Force Weather Agency has a golden opportunity to incorporate this emerging

language of choice into an ever expanding set of web based weather applications and lead the way in web exploitation.

APPENDIX A

Code for MR_CAT.XML

```
--This is XML code written by Captain Michael J Calidonna, last updated
 07 Jan 00.  It is designed to store data for different bases.

<CATALOG>
  <BASE>
    <LOCATION>California</LOCATION>
    <NAME>Travis</NAME>
    <ICAO>KSUU</ICAO>
    <FORECAST>Forecast for KSUU</FORECAST>
  </BASE>
  <BASE>
    <LOCATION>Ohio</LOCATION>
    <NAME>Wright-Patterson</NAME>
    <ICAO>KFFO</ICAO>
    <FORECAST>Forecast for ffo</FORECAST>
  </BASE>
  <BASE>
    <LOCATION>Arizona</LOCATION>
    <NAME>Davis-Monthan</NAME>
    <ICAO>KDMA</ICAO>
    <FORECAST>forecast for dma</FORECAST>
  </BASE>
  <BASE>
    <LOCATION>Japan</LOCATION>
    <NAME>Misawa</NAME>
    <ICAO>KKQQ</ICAO>
    <FORECAST>Forecast for kkqq</FORECAST>
  </BASE>
  <BASE>
    <LOCATION>Japan</LOCATION>
    <NAME>Okinawa</NAME>
    <ICAO>RODN</ICAO>
    <FORECAST>Forecast for RODN</FORECAST>
  </BASE>
</CATALOG>
```

APPENDIX B

Code for Weather.HTML

--This HTML code incorporates William J. Pardi's code for a virtual flower shop. Captain Michael J. Calidonna extended this code to include XML data for different bases. This code uses HTML, XML, and JavaScript as programming languages. Last updated 07 Jan 00

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>

<!—This sets up all the initial field -->

 <HEAD>
  <STYLE>
   .category {font-weight:bold; font-size:14}
   .fdata {font-size:16; color:#000099}
   .pdata {font-size:14; font-weight:bold}
   .cdata {font-size:16; color:#993300}
  </STYLE>

<!—This initializes the scripting language and opens the needed files -->

  <SCRIPT LANGUAGE="JavaScript" FOR="window" EVENT="onload">
   var xmlDso = xmldso.XMLDocument;
   xmlDso.load("Mr_Cat.xml");
   var xmlDso2 = xmldso2.XMLDocument;
   xmlDso2.load("dash-1.xml");
  </SCRIPT>

  <SCRIPT SRC=ShowXML.js>
  </SCRIPT>

  <SCRIPT LANGUAGE="JavaScript">

<!—This toggles the menu on and off -->

    function doMenu()
       {
     if (fmenu.style.display == "none")
      fmenu.style.display = "";
     else
      fmenu.style.display = "none";
     }
```

```
function doForecastMenu()
    { alert ("doForecastMenu")
 if (forecastmenu.style.display == "none")
  forecastmenu.style.display = "";
 else
  forecastmenu.style.display = "none";
 }

<!—This sets up all the bases to be displayed -->

function goRecord(indexNum)
    {
 var row = window.event.srcElement;
 var c = row.recordNumber - 1;
 xmldso.recordset.MoveFirst();
 while (c > 0)
  {
  xmldso.recordset.MoveNext();
  c = c - 1;
  }
 doMenu();
 }

<!—This is left for future development of some specified criteria -->

function goCat()
    {
    alert ("Hello")
    }
function mouseHover(state)
 {
 var row = window.event.srcElement;
 var colorChange = ((state == "over") ? "#ffff00" : "");
 row.style.backgroundColor = colorChange;
 }

<!—This sees if there is an aircraft number assigned to the record -->

function updateList()
 {
 var acVal = acName.value;
 if (acVal == "")
  alert ("You must enter a call sign.")
 else
```

```
    addToList();
   }

<!—This adds aircraft to the output file as needed -->

   function addToList()
      {
    var pmatch = 0;
    var cmatch = 0;
    var rootElem = xmldso2.XMLDocument.documentElement;
    var rootChild = rootElem.childNodes;
    var childNum = rootChild.length;
    var currDate = new Date();
    fullDate = (currDate.getMonth() + 1) + "/";
    fullDate += currDate.getDate() + "/";
    fullDate += currDate.getYear();

    for (i = 0; i < childNum; i++)
     {

     var currNode = rootChild.item(i);
     if (currNode.nodeName == "FLIGHT")
      {
        var acChild = currNode.childNodes;
      var call_signNode = acChild.item(0);
      var base_nameNode = acChild.item(1);
        var forecastNode = acChild.item(2);
        if (call_signNode.text == acName.value)
       {
       cmatch = 1;
       }
      else
       {
       cmatch = 0;
       }
        if (base_nameNode.text == icao.innerText)
       {
       pmatch = 1;
       }
      else
       {
       pmatch = 0;
       }
      }
        if (cmatch == 1 && pmatch == 1)
```

```
    {
      xmldso2.recordset.moveFirst();
    for (ds = 0; ds <= i; ds++)
     {
     if (ds != i)
      {
      xmldso2.recordset.moveNext();
      }
     }
    break;
    }
  }

if (cmatch != 1 && pmatch != 1)
  {
  xmldso2.recordset.AddNew();
  xmldso2.recordset("CALL_SIGN") = acName.value;
  xmldso2.recordset("BASE_NAME") = icao.innerText;
     xmldso2.recordset("FORECAST") = forecast.innerText;
  xmldso2.recordset("DATE") = fullDate;
  }
else
  {
     if (cmatch ==1 && pmatch == 1)
       {
            alert ("base already selected")
       }
     else
       {
            if (cmatch != 1)
       {
       xmldso2.recordset.AddNew();
       xmldso2.recordset("CALL_SIGN") = acName.value;
       xmldso2.recordset("BASE_NAME") = icao.innerText;
            xmldso2.recordset("FORECAST") = forecast.innerText;
       xmldso2.recordset("DATE") = fullDate;
       }
       else
       {
        if (cmatch == 1)
          {
          xmldso2.recordset.AddNew();
          xmldso2.recordset("CALL_SIGN") = acName.value;
          xmldso2.recordset("BASE_NAME") = icao.innerText;
               xmldso2.recordset("FORECAST") = forecast.innerText;
```

```
            xmldso2.recordset("DATE") = fullDate;
            }
          }
        }
        }

 showList();
    }

function showList()
 {
 var hold = 0;
 var rootElem = xmldso2.XMLDocument.documentElement;
 var rootChild = rootElem.childNodes;
 var childNum = rootChild.length;
 var purTable = "<TABLE CELLSPACING='5'><THEAD ALIGN='center'>" +
  "<TH>Base</TH><TH>Date</TH></THEAD>"
 for (i = 0; i < childNum; i++)
  {
  var currNode = rootChild.item(i);
  if (currNode.nodeName == "FLIGHT")
   {
   var acChild = currNode.childNodes;
   var acNum = acChild.length;
   for (ci = 0; ci < acNum; ci++)
    {
    var currAcNode = acChild.item(ci);
    if (currAcNode.nodeName == "CALL_SIGN")
     {
     var currAcName = acName.value;
     if (currAcNode.text == currAcName)
      {
      hold = 1;
      }
     else
      {
      hold = 0;
      }
     }
    if (currAcNode.nodeName == "BASE_NAME")
     {
     if (hold == 1)
      {
      purTable = purTable + "<TR ALIGN='center'><TD>" +
       currAcNode.text + "</TD>";
```

```
            }
           }
          if (currAcNode.nodeName == "DATE")
           {
           if (hold == 1)
            {
            purTable = purTable +"<TD>"+ currAcNode.text +"</TR>";
            }
           }
            }
       }
      }
     purTable = purTable + "</TABLE>"
     PTData.innerHTML = purTable
     }

<!—This shows all the forecasts selected with a particular aircraft-->

function showForecast()
        {
        var hold = 0;
        var rootElem = xmldso2.XMLDocument.documentElement;
        var rootChild =rootElem.childNodes;
        var childNum = rootChild.length;
        var fctTable = "<TABLE CELLSPACING ='5'>" +
                "<THEAD ALIGN = 'center'>"+
                "<TH>Base:</TH><TH>Forecast:</TH></THEAD>"
        for (i = 0; i < childNum; i++)
                {
                var currNode = rootChild.item(i);
                if (currNode.nodeName == "FLIGHT")
                        {
                        var acChild = currNode.childNodes;
                        var acNum = acChild.length;
                        for (ci = 0; ci< acNum; ci++)
                        {
                        var currAcNode = acChild.item(ci);
                        if (currAcNode.nodeName == "CALL_SIGN")
                                {
                                var currAcName = acName.value;
                                if (currAcNode.text == currAcName)
                                {
                                hold = 1;
                                        }
                                else
```

```
                        {
                        hold = 0;
                        }
                }
            if (currAcNode.nodeName == "BASE_NAME")
            {
                if (hold == 1 )
                        {
            fctTable = fctTable + "<TR ALIGN = 'center'><TD>"+
                        currAcNode.text + "</TD>";
                                }
                        }
            if (currAcNode.nodeName == "FORECAST")
                {
                if (hold == 1)
                        {
        fctTable = fctTable + "<TD>" + currAcNode.text + "</TR>";
                        }
                }
            }
        }
        fctTable = fctTable + "</TABLE>"
        FCTData.innerHTML = fctTable

        } <!-- end showForecast -->
</SCRIPT>

<TITLE>Weather Using XML</TITLE>
</HEAD>

<BODY>

<OBJECT WIDTH="0" HEIGHT="0"
    CLASSID="clsid:550dda30-0541-11d2-9ca9-0060b0ec3d39"
    ID="xmldso">
</OBJECT>

  <OBJECT WIDTH="0" HEIGHT="0"
    CLASSID="clsid:550dda30-0541-11d2-9ca9-0060b0ec3d39"
    ID="xmldso2">
  </OBJECT>

  <TABLE STYLE="position:absolute; left:10; top:10"
    CELLSPACING="6" ID="catalog">
```

```
 <TR>
  <TD ALIGN="right" CLASS="category">Location:</TD>
  <TD><DIV CLASS="fdata" ID="location"
   DATASRC=#xmldso DATAFLD="LOCATION">
  </TD>
 </TR>

 <TR>
  <TD ALIGN="right" CLASS="category">Name:</TD>
  <TD><DIV CLASS="fdata" ID="name"
   DATASRC=#xmldso DATAFLD="NAME">
  </TD>
 </TR>

 <TR>
  <TD ALIGN="right" CLASS="category">ICAO:</TD>
  <TD><DIV CLASS="fdata" ID="icao"
   DATASRC=#xmldso DATAFLD="ICAO">
  </TD>
 </TR>

<TR>
  <TD ALIGN="right" CLASS="category">FORECAST:</TD>
  <TD><DIV CLASS="fdata" ID="forecast"
   DATASRC=#xmldso DATAFLD="FORECAST">
  </TD>
 </TR>

</TABLE>

<DIV STYLE="position:absolute; left:300; top:20">
 <SPAN>Call Sign:</SPAN>
 <INPUT TYPE="Text" NAME="acName">
 <BR>

 <INPUT TYPE="Button" NAME="SL" VALUE="Add It!"
  onClick="updateList()">
 <INPUT TYPE="Button" NAME="Show" VALUE="Show XML Data"
  onClick="ShowXML(xmldso2.XMLDocument);">

 <DIV ID=PTData></DIV>

</DIV>

<INPUT TYPE="Button" NAME="Bases" VALUE="Bases"
```

```
  STYLE="position:absolute; left:10; top:170"
  onClick="doMenu()">
</DIV>


<DIV STYLE="position:absolute; left:200; top:170">
<INPUT TYPE="Button" NAME="Forecast" VALUE = "Forecast"
      onClick="showForecast()">
       <DIV ID=FCTData> </DIV>
</DIV>

<TABLE ID="fmenu"
  STYLE="position:absolute; left:10; top:192;
   display:none; cursor:hand"
  DATASRC=#xmldso
  CELLSPACING="0" CELLPADDING="0" BORDER="1"
  onMouseOver = "mouseHover('over')"
  onMouseOut = "mouseHover('out')"
  onClick = "goRecord()">


  <TR>
   <TD><DIV DATAFLD=ICAO></TD>
  </TR>
</TABLE>

<TABLE ID="forecastmenu"
  STYLE="position:absolute; left:200; top:192;
  display:none; cursor:hand"
  DATASRC=#xmldso2
  CELLSPACING="0" CELLPADDING="0" BORDER="1"
  onMouseOver = "mouseHover('over')"
  onMouseOut = "mouseHover('out')"
  onClick = "goCat()">
  <TR>
      <TD>
      <DIV DATAFLD=FORECAST>
      </TD>
  </TR>
 </TABLE>
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><B
R><BR><BR><BR><BR>
Sattelite Picture
      <A HREF="http:\\www.cnn.com">simple link</A>
Radar Summary
```

```
        <A HREF="c:\usrad.bmp">simple link</A>
 </BODY>
</HTML>
```

APPENDIX C

Code for Dash_1.xml

This code is an XML program designed to be updated through the code found in appendix B. It was written by Captain Michael J. Calidonna, last updated 07 Jan 00.

```
<DD175-1>
  <FLIGHT>
    <CALL_SIGN/>
    <BASE_NAME/>
    <FORECAST/>
    <DATE/>
  </FLIGHT>
</DD175-1>
```

APPENDIX D

Types of attributes and their meaning

Attribute Type	*Meaning*
CDATA	Character Data – anything not instructions to the computer
Enumerated	A list of possible values from which one will be chosen
ID	A unique name
IDREF	The value of an ID type attribute of an element in the document
IDREFS	Multiple Ids of elements separated by white space
ENTITY	The name of an entity declared in the DTD
ENTITIES	The names of multiple entities declared in the DTD, separated by white space
NMTOKEN	An XML name
NOTATION	The name of a notation declared in the DTD
NMTOKENS	Multiple XML names separated by white space

APPENDIX E

Acronyms

AFRL	Air Force Research Labs
AFWA	Air Force Weather Agency
AFWIN	Air Force Weather Information Network
AMD	Amendment
CDATA	Character Data
CMAPI	Core Mapping Application Program Interface
CSS	Cascading Style Sheet
DTD	Document Type Definition
HTML	Hypertext Markup Language
ICAO	International Civil Aviation Organization
IE 5.0	Microsoft Internet Explorer (version 5.0)
IFEB	Information Handling Branch
JRICP	Joint Reserve Intelligence Connectivity Program
SGML	Standard General Markup Language
SQL	Structured Query Language
TAF	Terminal Airdrome Forecast
XFragments	Extensible Markup Language's Fragmenting Language
XLink	Extensible Markup Language's Linking Language
XML	Extensible Markup Language
XPointers	Extensible Markup Language's Pointing Language
XSL	Extensible Style sheet Language
W3C	World Wide Web Consortium

Bibliography

Air Force Research Labs. AFRL Mission Statement Rome Labs. Rome Labs, 1999.

Air Force Weather Agency (AFWA). Air Force Weather Agency Strategic Plan. AFWA, 1999.

Bos, Bert. XML in 7 points http://www.w3.org/XML/1999/XML-in-10-points, 20 Sep 1999

Bosak, Jon and Bray, Tim. "How XML beats HTML," Scientific American; 89-93 (May 1999)

Darnel, Rick and others. HTML4 The Comprehensive Solutions Package! Unleashed. Sams.net Publishing, 1997.

Garber, Lee.. "Newsbriefs," Computer 20-22, (May 1999)

Harold, Elliotte R. XML Bible. IDG Books Worldwide, 1999.

Hoffman, Charles and Kurt, Christopher. "The XML Files," Journal of Accountancy; 71-77 (May 1999)

Holzner, Steven. JavaScript Complete. McGraw-Hill Companies, Inc., 1998.

McGrath, Sean. XML by Example Building E-Commerce Applications. Prentice Hall, 1999

Pardi, William J. XML in Action Web Technology. Microsoft Press, 1999.

World Wide Web Consortium. Extensible Markup Language 1.0. http://www.w3.org/TR/1998/Rec-xml-19980210.html, 10 Feb 1998

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2000	Master's Thesis

4. TITLE AND SUBTITLE

EXTENSIBLE MARKUP LANGUAGE AS A WEATHER TOOL

5. FUNDING NUMBERS

6. AUTHOR(S)

Calidonna, Michael, J., Captain, USAF

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 P Street, Building 640
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GM/ENP/00M-2

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Michael Mayhew
26 Electonic Pkwy
Rome Labs AFRL/IFEB
Rome, NY 13441

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Chairman: Lt Col Cecilia A. Miner, ENP, DSN:785-3636 ext.4645
Member: Lt Col Michael K. Walters, ENP, DSN:785-3636 ext.4681
Member: Lt Col Timothy M. Jacobs , ENG, DSN:785-3636 ext.4279

12a. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

ABSTRACT (*Maximum 200 Words*)

This thesis is a proof of concept work that will extend the Core Mapping Application Program Interface (CMAPI) components to include weather data. The CMAPI project is headed by Air Force Research Lab (AFRL)/ Information Directorate Information Handling Branch (IFEB) at Rome labs in Rome, New York. This work extends the CMAPI project in two distinct areas. The first goal is to figure out how to overlay and display weather data on a dynamically linked Internet platform. This was accomplished by incorporating existing data from the Air Force Weather Agency (AFWA) into the CMAPI program in a static environment. The other goal is to learn about the Extensible Markup Language (XML) and how it can contribute to characterizing structured data (i.e., weather data output from AFWA). Once this tool can be exploited, a dynamic interaction between the CMAPI program and all AFWA products could be developed. The overall goal is to make it easy for the system, and the application of that system, to ingest and manipulate data.

14. SUBJECT TERMS	15. NUMBER OF PAGES
XML, Extensible Markup Language, JavaScript, HTML, Meteorology, Internet	66
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL