

Interactive Weather Station Data Display Through The Internet

By
Francisco Javier Espaillat Valcárcel

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE
in
COMPUTER ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGUEZ CAMPUS
2003

Approved by:

Dr. Amos Winter, Ph.D.
Member, Graduate Committee

Date

Dr. Néstor Rodríguez, Ph.D.
Member, Graduate Committee

Date

Dr. Ramón Vásquez, Ph.D.
President, Graduate Committee

Date

Dr. Eric Gamess, Ph.D.
Representative of Graduate Studies

Date

Dr. Jose L. Cruz, Ph.D.
Department Director

Date

Abstract

The study of weather phenomena is of growing importance. Changes in the overall conditions of the world, such as global warming, ozone depletion, and the uncertainty on the causes of these events, has made weather research more important than ever. To study these phenomena, researchers rely on archived weather data. Which are usually found by request at institutions that operate weather stations. The researcher's analysis is facilitated if stored in electronic form, such as computer documents or spreadsheets. The structure, and format of the data, along with the capacity to manipulate and display it in a user-friendly interface will greatly enhance the researcher's study of the climate data. Here I present an application for obtaining and displaying weather data on the Internet, a database system that allows the retrieval of data by other organizations, and proposed data structures to be adopted for the sharing of this type of data.

Resumen

El estudio de fenómenos atmosféricos ha crecido en importancia. Cambios en las condiciones generales del mundo, como calentamiento global, destrucción de la capa de ozono, y la incertidumbre sobre las causas de estos eventos, han hecho el estudio climatológico más importante que nunca. Para estudiar estos fenómenos, los investigadores se apoyan en datos archivados. Estos son usualmente encontrados por solicitud a instituciones que operan estaciones climatológicas. El análisis de data se facilita si es almacenada en forma electrónica, como documentos de computadora u hojas electrónicas. La estructura, formato de la data, y la capacidad de manipularla y presentarla en una interfaz amigable realzará significativamente el estudio de data climatológica. Aquí presento una aplicación para obtener y presentar data climatológica en el Internet, un sistema de base de datos que permite la obtención de data por otras instituciones, y propuestas para estructuras de datos que pueden ser adoptadas para el intercambio de data climatológica.

Acknowledgements

We would like to acknowledge Rose Loehr of the USDA and her team for providing us access to their weather databases and invaluable input on the development of the application, thus giving support to our project and making it of more use to weather researchers.

Special thanks to Dr. Raúl Zapata of the UPR for his assistance on the needs and wants of weather investigators when viewing weather data, and his contributions of data from weather stations throughout Puerto Rico

Table of Contents

List of Figures.....	vi
List of Sample Codes.....	vii
List Of Apendices.....	viii
Nomenclature.....	ix
Chapter 1. Introduction.....	1
1.1 Importance of Weather Data Research.....	1
1.2 Previous Work in the Field.....	2
1.3 Contributions.....	3
1.4 General Objectives.....	4
1.5 Specific Objectives.....	5
1.6 Chapter Outline.....	6
Chapter 2. Data Structure.....	8
2.1 Overview.....	8
2.2 File Structure.....	13
2.3 Readings Data.....	15
2.4 Month Data.....	15
2.5 Year Data.....	16
2.6 Compiled Readings Data.....	17
2.7 Compiled Day Data.....	18
2.8 Compiled Month Data.....	19
Chapter 3. XML Data.....	21
3.1 Location Data.....	23
3.2 Readings Data.....	26
3.3 Month Data.....	29
3.4 Year Data.....	31
Chapter 4. Software.....	34
4.1 Support Classes.....	37
4.2 Field Classes.....	39
4.3 Location Classes.....	40
4.4 Data Management Classes.....	42
4.5 Web Interface Classes.....	47

Chapter 5. Software Operation.....	50
5.1 Data Management.....	50
5.2 Web display.....	57
Chapter 6. Web Interface.....	60
6.1 Performance Tests.....	69
Chapter 7. Conclusions.....	71
7.1 Future Work.....	72
Bibliography.....	74

List of Figures

Figure 2-1.	Hierarchical Structure of Data Files.....	14
Figure 3-1.	Location Data Structure.....	26
Figure 3-2.	Day Data Structure.....	29
Figure 3-3.	Month Data Structure.....	31
Figure 3-4.	Year Data Structure.....	33
Figure 4-1.	Software Components.....	36
Figure 4-2.	WXTime and WXDay classes.....	37
Figure 4-3.	WXRecord UML Diagram.....	38
Figure 4-4.	WXRecordSet UML Diagram.....	38
Figure 4-5.	WXStatRecord UML Diagram.....	39
Figure 4-6.	WXStatRecordSet UML Diagram.....	39
Figure 4-7.	WXField class.....	40
Figure 4-8.	WXLocation UML Diagram.....	42
Figure 4-9.	WXDayData UML Diagram.....	43
Figure 4-10.	WXMonthData UML Diagram.....	44
Figure 4-11.	WXYearData UML Diagram.....	44
Figure 4-12.	WXCompiledReadingsData UML Diagram.....	45
Figure 4-13.	WXCompiledDayData UML Diagram.....	46
Figure 4-14.	WXCompiledMonthData UML Diagram.....	46
Figure 6-1.	Location Page.....	62
Figure 6-2.	DataTable Header.....	63
Figure 6-3.	DayData data table.....	64
Figure 6-4.	MonthData data table.....	65
Figure 6-5.	YearData data table.....	66
Figure 6-6.	DayDataPrintTable.....	67
Figure 6-7.	CompiledDataTables.....	68
Figure 6-8.	Graph table.....	69

Sample Codes

Sample Code 2-1.	Readings Data File Header.....	15
Sample Code 2-2.	Readings Data Field Headers.....	15
Sample Code 2-3.	Readings Data File.....	15
Sample Code 2-4.	Month Data File Header.....	16
Sample Code 2-5.	Month Data Field Headers.....	16
Sample Code 2-6.	Month Data File.....	16
Sample Code 2-7.	Year Data File Header.....	17
Sample Code 2-8.	Year Data Field Headers.....	17
Sample Code 2-9.	Year Data File.....	17
Sample Code 2-10.	Compiled Readings Data File Header.....	18
Sample Code 2-11.	Compiled Readings Data Field Headers.....	18
Sample Code 2-12.	Compiled Readings Data File.....	18
Sample Code 2-13.	Compiled Day Data File Header.....	19
Sample Code 2-14.	Compiled Day Data Field Headers.....	19
Sample Code 2-15.	Compiled Day Data File.....	19
Sample Code 2-16.	Compiled Month Data File Header.....	20
Sample Code 2-17.	Compiled Month Data Field Headers.....	20
Sample Code 2-18.	Compiled Month Data File.....	20
Sample Code 3-1.	XML Location Tags.....	23
Sample Code 3-2.	XML Location Information Tags.....	23
Sample Code 3-3.	XML Location Time Range Data Tags.....	24
Sample Code 3-4.	XML Location Field Information Tags.....	25
Sample Code 3-5.	XML Day Data Tags.....	26
Sample Code 3-6.	XML Day Data Information Tags.....	27
Sample Code 3-7.	XML Day Data File.....	28
Sample Code 3-8.	XML Month Data Tags.....	29
Sample Code 3-9.	XML Month Data Information Tags.....	29
Sample Code 3-10.	XML Month Data File.....	30
Sample Code 3-11.	XML Year Data Tags.....	31
Sample Code 3-12.	XML Year Data Information Tags.....	31
Sample Code 3-13.	XML Year Data Tags.....	32

List of Apendices

Appendix A.	Programmer's Manual.....	76
Appendix B.	User's Manual.....	90

Nomenclature

ASCII	American Standard Code for Information Exchange
CSV	Comma Separated Values
DBMS	Database Management System
HTML	HyperText Markup Language
J2SE	Java 2 Standard Edition
Java	Java Programming Language.
JDK	Java Development Kit
JRE	Java Runtime Environment
JSP	Java Server Pages
OOP	Object Oriented Programming
UML	Unified Modeling Language
UPR	University of Puerto Rico
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

Chapter 1

Introduction

1.1 Importance of Weather Data Research

The study of weather patterns has been of crucial importance to weather researchers for decades. It is through the study of climatic conditions from specific events that investigators can determine the causes of these events, and predict their reappearance in the future. Observation of past data is also necessary to notice trends or patterns in weather variables that could have noticeable consequences to a particular region. Problems such as global warming have intensified the need for these kinds of studies.

In order for climatologists to undertake these kinds of tasks, they must rely on archives of weather data. Instruments on weather stations take readings and samples of weather variables at particular times and locations. With the study of weather variables, such as temperature, wind speed, and pressure, during weather events, such as hurricanes, storms, and tornadoes, researchers can acquire better insight into the causes of these events, or the conditions that favor their appearance. These archives usually need to be extensive and representative, spanning several years at a significant location. To the researchers this usually entails a considerable amount of time gathering these archives and reviewing them for pertaining information.

Real-time data is also of importance to persons and institutions not necessarily in the field of weather research. Access to current climate conditions can be useful to farmers, traffic reports, emergency service units, and countless others. As of yet, these data are usually retrieved by special by agreements among the institutions that require them, and those working to retrieve the data.

1.2 Previous Work in the Field

Weather data can be obtained from several institutions through their websites. The National Oceanic and Atmospheric Administration (NOAA: <http://www.noaa.gov>) provide descriptive forecasts of weather conditions for the following week. Current weather conditions are posted hourly, with the option of viewing hourly changes in weather for the previous two days. Also they display general weather conditions in map display using average values for locations and color-coding for regions. On our application we will provide the same useful service for real-time data. Their color-coding mapped display of weather conditions can be used as a model for future implementations of mapped display on our system.

The Weather Channel is the most popular of these websites. It has a well-developed interface for current weather conditions such as readings for temperature, wind speed, wind direction, humidity, pressure for any point in the US. It also displays satellite images of varying types in latest movement animations. This site is developed for the general audience who inquires on weather conditions that affect them immediately, and wish to retrieve that information quickly. It is not intended for weather researchers, as it does not provide a mechanism for retrieving past archived data. We hope our application can serve both these interests providing the most solicited items of data so general users can find it quickly, and also allow experienced researchers to do more complex queries and operations on archived data.

One site that provides archived data is the University Corporation for Atmospheric Research (<http://www.ucar.edu>). It has real-time data maps and forms where the user can select fields to display on the map for a particular location. It also offers users a downloadable software package for weather data analysis called Unidata Software. Along with it come several components of weather data analysis such as LDM (Local Data Manager), netCDF, GEMPAK and McIDAS for data retrieval and satellite images. UCAR however demands a license to use any of these products. A similar

desktop application developed in the Java programming language has also been developed in [15]. It provides interactive data selection utilities from several sources and graphical views of data using the built-in Java tools. This application requires installation of the software on the client computer, along with the Java Runtime Environment to run java applications. By providing our data through the Internet on a web interface, we expect to maintain a portal of climate data which can easily change its interface and querying utilities to better suit the researcher's needs as the application continues to develop. This allows a product that is in a constant process of improvement, as opposed to conventional applications that are packaged once and require the user to update for any new changes. It also relieves the user of the need to install applications on his local computer to view data. Data is accessible through any computer with an internet connection and Internet browser.

1.3 Contributions

Our main goal is to provide researchers with a resource to facilitate their study of weather research. This website should be attractive enough for other institutions to share their data and create a repository of weather data that researchers throughout the world can use.

The Extensible Markup Language (XML) promises to become the standard of web languages for the future [7,11,14]. Its usability to describe any data structure also allows it to be used as a hierarchical database. XML is still not yet widely known and implemented by most browsers. For the time being, institutions are seeking to establish standards of structures for different types of XML data [4]. It is our hope that the structure we've developed in CSV and XML format, set a precedent for weather data storage through the Internet, and that other institutions can implement and facilitate the exchange of this type of data.

We have also designed a new interface to aid researchers in their study of climate data. This interface, accessible through the web, makes use of web resources to present the weather data to the client in a user-friendly display. This comprehensive interface for displaying weather data, providing appropriate visual display on request as well, is not yet available from any other source on the Internet today.

1.4 General Objectives

- *Provide weather researchers with a tool for retrieving and analyzing weather data.*

A website should have a database of climate data available to weather researchers. These data must be accessible through easy to use data request forms, which displays the requested data in several arrangements that best fit the type of data requested. The user must also be able to download these data into his computer for use in his own calculations and research.

- *The data must be available to third party organizations through the Internet.*

The data should also be retrievable to other institutions without consultation of our department. A mechanism will be developed for other researchers or programmers to use our database through the Internet and retrieve whatever data they require.

- *The user interface should be intuitive and optimally designed to minimize confusion and learn-time to new users.*

This website is destined to be used by all kinds of users interested in weather data. These can be weather researchers conducting scientific investigations, or laymen seeking the latest weather conditions of their local region. For this reason the interface must not be too complex for the regular user, yet provide researchers with a useful, not too limited, tool for conducting their investigations. The

interface implemented should appeal to both users, and facilitate their understanding of how to obtain the data they need.

1.5 Specific Objectives

- *All relevant data should be available to users through the website.*

Aside from the climate data that is needed by interested users, the website should display all relevant information deemed important or that would facilitate the user's understanding and research. This can be data about the location that he's researching, such as its geographical coordinates, or time zone.

- *The website should perform the pertinent statistical calculations for fields that require them.*

Not only should our site display the archived data, it should also carry out operations on the data that are important to most users. These operations can be calculating averages from data sets, or finding its highest and lowest statistics. Our work should anticipate which of these calculations are necessary to most researchers and provide the mechanism to retrieve these kinds of data.

- *Quality control. The application must allow field specific algorithms to be implemented, which recognize and discard anomalous readings.*

For different fields and types of data, such as temperature or precipitation data sets, the application must allow the implementation of some quality control algorithm. This algorithm should function only for the field it was designed to validate, given that a quality control algorithm would be unique for every field implemented.

- *The database should be extensible.*

The system must be designed to allow new locations and fields to be added or removed from the database with minimal modifications to the software.

- *Queries of data should be available to the user.*

It is important for users to be allowed to request a full range of data for a time range without having to seek each individual portion of data. A data compilation form must be available to take in requests of data compilation, which the application should retrieve and provide to the user in a continuous time series.

- *All data displayed should be downloadable to the user.*

All the data that the user requests on the site must have a representation in a file that can be downloaded by the user. This file should then allow users to review or manipulate the data at their convenience.

- *Charts and graphs of data will be available for users, which suit their respective fields.*

The data can be much easier to observe and navigate if seen in visual form. The advantage of displaying data visually is that data can be perceived and interpreted globally, without needing to scroll through it from beginning to end [2]. The system should allow the user to view the data he has requested in a chart or graph, which corresponds to the particular field, he requested and the data should be displayed in their most representative type of chart.

1.6 Chapter Outline

In chapter 2 the data structure of our database is outlined. Here the format of the data files is defined and how they will be organized by the application. The different types of data files are also described and how their data are derived from lower order data files.

Chapter 3 defines the XML structures for the data. The various hierarchies for location data, and the readings data, along with the statistical data upon them, is outlined. Diagrams representative of the hierarchical structure they create will also accompany the XML definitions.

A description of the software is provided in Chapter 4. Here the different components that make up the application is identified. And an explanation on how these components work together and which of them can be manipulated or replaced to extend the application's database or to improve on the user interface.

The operation of these components is explained in Chapter 5. The process by which new data is retrieved from the source and stored into the database is described. Also how the data is retrieved from the database and displayed to users through the Internet in its different forms.

Chapter 6 describes the implemented web interface. Providing a description of every page, and how it enhances the display of data to facilitate its study by the users. The steps through which users retrieve data and navigate between locations, dates and data pages are also explained.

Chapter 2

Data Structure

2.1 Overview

This chapter will outline the structure of the database. It describes the format of how data will be organized within each data file. Also the naming conventions for each data file types, and their position in the directory structure hierarchy.

The database must begin with the Location data. Each weather station must have its entry in the database and all the relevant information particular to it. Some information is indispensable for a Location to operate properly in the application. Such as the keyword, a unique identifying word for file storage purposes. The Location name, which usually should describe the area from which the station collects data; in the case of several or more than one location recording data in the same area, the name should identify the owner or institution that operates the weather stations. The time range of data, the date of the first and last recorded readings for that Location. Other information is the Location's altitude, geographical coordinates, time zone, station's owner or institution, etc... This information is not requisite for the Location to function in the application, but can still be useful to the user in interpreting the Location's data.

One set of information about the Location that is key to its operation is the Fields data. Fields represent the particular sensors for which the location records data from. For example: Temperature, Relative Humidity, Pressure, Precipitation, etc. A Location can have Fields without supporting a corresponding sensor, if that field can be calculated from other fields for every record of data. For instance, a Heat Index reading can be calculated given a Temperature and Relative Humidity reading. All the Fields a Location

supports must be stored with the Location's information, along with their relevant statistical data, which varies from Field to Field. This statistical data is not crucial for the application's regular data retrieval and storage operation, but it can be important in user interface components. Such as drawing graphs, for example.

After a Location has been registered into the database, and provides all the necessary information about itself to function correctly, it can begin to gather readings data. These readings data are the bulk of the database. They are first grouped into records. One record contains one reading for every Field in the Location at one particular time of the day. A Day Data is then the set of records for a whole day's readings. These need to be stored each in their own separate tables.

It should be noted that the day's time range goes from 0001 (12:01 A.M.) to 2400 (12:00 M.). This is due to the fact that many, though not all, weather stations use averaging algorithms of readings taken in faster intervals to arrive at a more representative reading of the stored readings interval. For example, a weather station stores temperature data for every 10 minutes. This can mean that the temperature sensor takes a sample of the current air temperature every ten minutes and sends that reading to be stored into the database. But not all stations work this way; some stations take sample readings at higher intervals, for example one minute, and then average those readings when the 10-minute interval is reached. This can lead you to interpret an entry in the database of:

```
Time, Temperature (°F)
2350, 65.4
2400, 65.7
```

As either “the temperature taken at 2400 was of 65.7 degrees Fahrenheit”, or “the average temperature of the time interval from 2350 to 2400, taken at 1 minute intervals, was of 65.7 degrees Fahrenheit”. Likewise the first entry in the table for the next day would be:

```
Time, Temperature (°F)
10, 65.5
```

Which can also be interpreted as the average temperature from 2400 in the previous day to today's 10. This depends on how the station is set up to take its readings. We don't

count on always knowing which method a station implements, so we feel it safer to err on the side of anticipating that a reading at one point in time is representative of the time interval before it. This explains why the reading at 12:00 Midnight is stored as the 2400 reading of the day before, rather than the 0000 reading of the day after it.

The application will implement a Quality Control system where erroneous readings must be eliminated from the archived data. These occur because of malfunctions in the station's sensors or software type errors. A special symbol is then assigned to fill in the space of where an erroneous reading was discarded. If gaps in the data occur for several time intervals in one day, in other words, there are several Records that contain nothing but invalid markers, these records will still be recorded into the data file. This means a data file might look something like this:

Time	Temperature (°F)	Relative Humidity (%)	Pressure (inHg)	Wind Speed (mph)
1310	75.7,	89.5,	766.3,	10.4
1320	75.5,	89.3,	-,	8.7
1330	-,	-,	-,	-
1340	-,	-,	-,	-
1350	-,	-,	-,	-,
...				
1720	82.4,	60.4,	766.2,	3.5

Where “-“ is the “invalid reading” or “not available” marker symbol. This is done, as opposed to skipping 4 hours of data entry, to preserve the integrity of the data structure. And provide the user with a continuous time series of data. Which might also be helpful for users who perform operations on the data with spreadsheet applications that might get “confused” otherwise.

After these readings are stored, statistical values upon these readings need to be calculated and stored as well. These statistical values depend on the Fields the Location supports. Currently there are only four statistical values to be calculated out of any readings set. These are Low, Average, High, and Total. Fields support a number of these statistics, but not all. For example, a Temperature Field supports Low, Average, and High, since these are relevant statistics. Precipitation however, supports High, and Total. These statistical values are calculated at two levels, day and monthly.

The Low statistic is the lowest reading in the set; Average is the sum of all valid readings divided by the number of valid readings in the set; High is the highest reading, and Total is the sum of all valid readings.

The Month Data is the set of statistical values for every day in one month. The statistical values of a Day Data is calculated and stored in a Statistical Record. This holds the relevant statistical values of every Field in the Location for that day. The Statistical Records for all the days in a given month constitute a Month Data.

After we have a Month Data, we can also calculate statistical values on its data, and arrive at a new Statistical Record for that month. These are calculated from corresponding statistical values in the Statistical Records of the Month Data. That is, the Month's Low reading is taken by calculating the lowest reading of all the Low readings in every Day Statistical Record, and so on. The set of all the Month Statistical Records then is grouped into a Year Data.

Considering that weather records can span several years and decades, we can expect a massive amount of data to be handled by this system. It is for this reason I have chosen not to use any mainstream DBMS, but to store these data in the basest format available, Comma Separated Values (CSV) ASCII files. This format is very resistant to data corruption. The loss of large sequence of bytes does not corrupt the remaining data. It is easily readable, both by third party applications or operators who are manipulating the database directly. It also stores its data independent of any DBMS, so the data is available for use regardless of any problems with the application [4]. This structure adopted to store the data resembles a traditional two-dimensional table format, where commas are used as field delimiters. This format holds no overhead; it provides the least amount of storage space for our data. They are self-explanatory and independent of our system, so if any malfunction should occur on our software, the data is still usable by any other software. Because they are also self-explanatory, any other organization can retrieve data directly from these files without needing our software to mediate. The CSV

file format is also supported by mainstream spreadsheet applications, such as MS Excel and Lotus 123; so data can be easily imported into spreadsheets and then manipulated at the user's convenience.

The Data in the database will not be stored in one specific unit system. Currently, our application supports two unit systems, British and Metric. Each Field must be assigned one of these unit systems as their "Most Precise Unit System". This is done because readings are always rounded off to the 2nd decimal place, and it is more accurate to save these readings in a unit system that is more precise. For example, Temperature readings are more accurately stored in the British unit system (°F) than in Metric (°C). Because a change in 0.01°F is smaller than a change in 0.01°C. While Precipitation is more accurately stored in the Metric System; anything from 0.01mm to 0.25mm might still register as 0in.

Data will also be stored in Extensible Markup Language (XML) format. These XML files will also be stored in using ASCII format. This ASCII XML format allows the data in a format that uses semantic tags, which are self-describing and make it easy for *"humans to read and computers to process"*[6]. The data will always be accessed through the CSV files by our software, but the XML format may facilitate the data retrieval for other institutions and may be used for developing a more convenient web user interface in the future when more browsers implement XML to the fullest.

Compiled Data files are data files of specific time ranges and Fields that the user requests through the webpage. They can be either of readings, day, or month, compiled data and follow almost the same structure as the Readings, Month, and Year Data files respectively. Following a similar format to the previous Data files, they span several days, years or months, and gather data only for specific fields the user requests. They are always generated dynamically upon request from the user through the webpage, and temporarily stored in a common Compiles directory, directly under the Records directory.

Unlike the previous Data files, they are always generated in either the Metric or British unit systems.

The user requests them through the web interface. In the Location's page, he will have a Compile Data form, where he can input the time range of data he wishes to compile, the Fields of the Location he wants data for, and the Unit System the data is to be given in. The Data Management software then generates the data file in the compiles directory. While the user views the data through the website in its web format, he'll have the option of downloading the file in the Compiles directory, which is in CSV format.

The file is stored in the Compiles directory only temporarily. All compiled data files, of all types (Readings, Day, Month) are stored in this common directory. But the naming convention used to create every file should avoid any conflict of data. This Compiles directory is emptied of all its files periodically.

2.2 File Structure

As previously stated, the software will read archived data from the CSV data files. These files provide a much easier and faster way of retrieving the data by the software (as opposed to the XML files, which provide an easier way for users to view data).

The data files will be stored in a hierarchical or tree structure. At the root of the structure is a Records Directory. This directory will be assigned to the software as the root directory for archived data. This directory will then branch out into subdirectories for each specific Location. These directories will be named by the Location's keyword property (stored in the particular Location class). Inside this directory, we'll have subdirectories for each year in the Location's time range. Each of those directories holds the Year Data CSV file for that year and subdirectories for each month within that year. These subdirectories will be named by the month name of the particular month; January,

February, March, and so on... Inside each of these month directories, will be stored all the Readings Data Files for that particular month, along with the Month Data file for the particular month.

The naming convention for these data files will be as follows:

- The Year Data file will have the year followed by the location's keyword.
- The Month Data file will have the month name, followed by the year and the location keyword
- The Readings Data files will use an eight digit date representation, where the first two digits represent the day, the next two the month, and the next four the year, followed by the location keyword.

All these files carry the corresponding .csv extension identifying them as CSV data files.

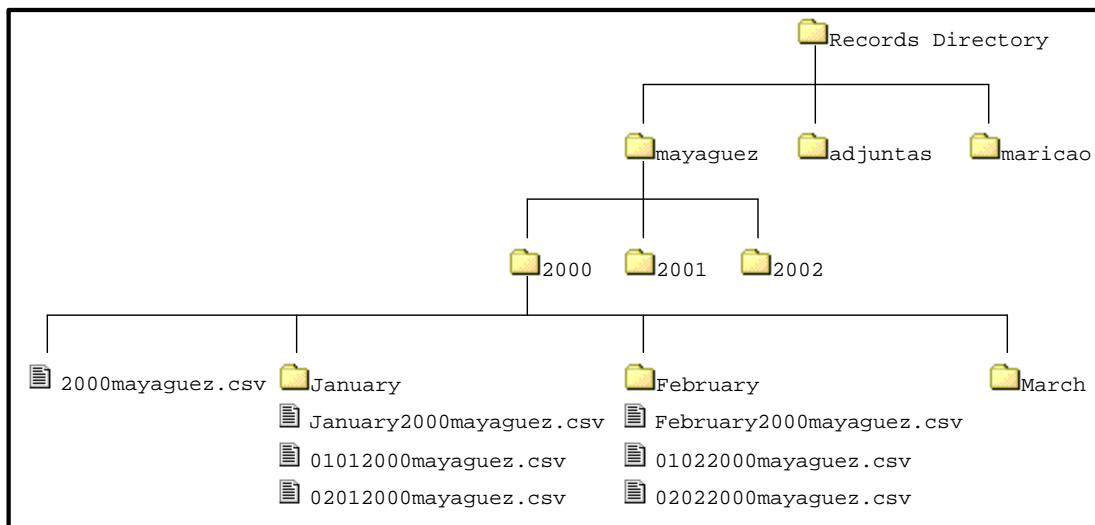


Figure 2-1. Hierarchical structure of data files.

2.3 Readings Data

The Readings Data files stores all the readings for every interval and every Field recorded for that day. The data in these files will be separated into fields delimited by commas. They will have the following structure:

The first line is a location and date indicator. The first field is the location name, followed by the month name, then the day of the month, and the year.

Mayaguez, January, 3, 2003

Sample Code 2-1. Readings Data File Header

The second line is the Field headers. The first field is the time of day, followed by the sensors that particular location supports. In parenthesis the unit the data for each Field is stored in.

Mayaguez, January,	3,	2003
Time,	Temperature (°C),	Precipitation (mm), Pressure (mmHg)

Sample Code 2-2. Readings Data Field Headers.

What follows is the data recorded for that day. The first field is the time of day of the recording, followed by all the readings corresponding to that time. The time of day is represented in 24 hour format. The readings are ordered respectively to the ordering in the field headers:

Mayaguez, January,	3,	2003
Time,	Temperature (°C),	Precipitation (mm), Pressure (mmHg)
10,	66.89,	0, 766.4
20,	66.67,	0.1, 766.8
30,	66.39,	08, -

Sample Code 2-3. Readings Data File.

2.4 Month Data

The Month Data file has a similar structure. The records are identified by the day of the month, and there is a header for every statistic that a Field supports. Some entries can be tagged as invalid if there aren't any valid readings for that Field in the

corresponding day. This may be caused by the weather station going offline for more than a day.

The first line in the Month Data file is the Location and Date information:

Mayaguez, January, 2003

Sample Code 2-4. Month Data File Header.

Following, the Field headers. The first header is the day of the month, then one header for every statistic every Field supports.

Mayaguez, January, 2003

Day,	Low Temperature (°F),	Avg Temperature (°F),	High Temperature (°F)...
------	-----------------------	-----------------------	--------------------------

Sample Code 2-5. Month Data Field Headers.

Then the statistical data for every day in the month:

Mayaguez, January, 2003

Day,	Low Temperature (°F),	Avg Temperature (°F),	High Temperature (°F)...
------	-----------------------	-----------------------	--------------------------

1,	67.97,	74.95,	85.4,
----	--------	--------	-------

2,	66.41,	72.82,	85.5,
----	--------	--------	-------

3,	65.31,	74.19,	88.0,
----	--------	--------	-------

4,	65.07,	74.1,	86.1,
----	--------	-------	-------

5,	-,	-,	-,
----	----	----	----

Sample Code 2-6. Month Data File.

2.5 Year Data

The Year Data file follows the same structure as the Month Data file. The records are ordered by the month number (starting with 1 as January) and, like the Month Data file, there is one header of every statistic supported by each Field. It is possible to find an Invalid Reading tag if there is no data for that particular field during the whole month. This could happen, if a sensor was installed on a later date than the year in the Year Data file. “Not Available” tags will still be assigned to all entries in that Field prior to the time the sensor was installed to maintain the structure of that Locations’ data files.

The First line for the Year Data file is the Location and date info:

```
Mayaguez, 2003
```

Sample Code 2-7. Year Data File Header.

Next the headers. The First header is the month, followed by headers for every statistic supported by that field.

```
Mayaguez, 2003
```

```
Month,      Low Temperature (°F), Avg Temperature (°F), High Temperature (°F)...
```

Sample Code 2-8. Year Data Field Headers.

Then the statistical data for every month in that year:

```
Mayaguez, 2003
```

```
Month,      Low Temperature (°F), Avg Temperature (°F), High Temperature (°F)...
```

```
1,          64.46,                75.13,                89.4
2,          63.25,                75.87,                103.3
3,          63.48,                76.59,                91.9
```

Sample Code 2-9. Year Data File.

2.6 Compiled Readings Data

These files store all the readings for a specified time range. This format follows very much the format of the day data files. Every record is identified by the time of day and the day of the reading, given that one Data file can span several days. The naming convention for these files in the Compiles Directory is:

- The time range of the file in the 8-digit date convention.
- The Location's keyword.
- The unit system the data is stored in.

A example of a compiled readings file name is:

```
04052001-17082001mayaguezmetric.csv
```

The Compiled Readings Data has a format very similar to the Readings Data. With the exception that records need to be fully identified by the date and time of the record.

First the time range, and Location name of the data:

Mayaguez, May, 4, 2001, to, August, 17, 2001
--

Sample Code 2-10. Compiled Readings Data File Header.

Then the field headers with their corresponding units. Since compiled data can stretch across months, or years, all date fields are necessary.

Mayaguez, May, 4, 2001, to, August, 17, 2001
Year, Month, Day, Time, Temperature (°C), Relative Humidity(%)

Sample Code 2-11. Compiled Readings Data Field Headers.

And the data:

Mayaguez, May, 4, 2001, to, August, 17, 2001
Year, Month, Day, Time, Temperature (°C), Relative Humidity(%)
2001, 5, 4, 10, 65.4, 89.6
2001, 5, 4, 20, 65.3, 89.4
...
2001, 5, 4, 2400, 66.7, 89.6
2001, 5, 5, 10, 66.3, 89.2

Sample Code 2-12. Compiled Readings Data File.

2.7 Compiled Day Data

These files store statistical values of the Fields and time range specified on a day basis. Very similar to the Month Data files, but not limited to any one month. Records are identified by the date of their statistical data. The naming convention for these files is:

- The time range of the data using in their 8-digit date format.
- The Location's keyword.
- The Unit System the data is stored in.
- A "Stats" suffix to distinguish it from the Compiled Readings Data files.

An example of a Compiled Day Data file:

17122001-03012002mayaguezmetricStats.csv

The first line of the data file presents the time range and Location information just as the Compiled Readings Data file:

Mayaguez, December, 17, 2001, to, January, 3, 2002
--

Sample Code 2-13. Compiled Day Data File Header.

Then the Field headers. The date is represented by the year, month and day. And a header for every statistic supported by the chosen Fields.

Mayaguez, December, 17, 2001,	to,	January, 3, 2002
Year,	Month,	Day, Low Temperature (°C), Avg Temperature (°C), High Temperature (°C)

Sample Code 2-14. Compiled Day Data Field Headers.

And the data:

Mayaguez, December, 17, 2001,	to,	January, 3, 2002
Year,	Month,	Day, Low Temperature (°C), Avg Temperature (°C), High Temperature (°C)
2001,	12,	17, 65.7, 87.9, 92.8
2001,	12,	18, 66.8, 82.5, 96.7
...		
2002,	1,	3, 68.2, 89.5, 95.1

Sample Code 2-15. Compiled Day Data File.

2.8 Compiled Month Data

Like the Year Data files, these gather statistical data for specified Fields at a monthly level for a specified time range. Their records are identified by the year and month of the data they represent. The naming convention for these files is:

- The time range of data compiled marked by their month name and year.
- The Location's keyword.
- The Unit System the data is stored in.

A Compiled Month Data file is:

April2001-November2001mayaguezbritish.csv

The data format is very much the same as the Compiled Day Data files. With the exception that records are only identified by a Year and Month field. The first line of the file is the Location and time range indicator:

Mayaguez, April, 2001, to, November, 2001

Sample Code 2-16. Compiled Month Data File Header.

Then the field headers. Year and Month for date, and one header for every statistic every field supports.

Mayaguez, April, 2001,	to,	November, 2001
Year,	Month,	Low Temperature (°F), Avg Temperature (°F), High Temperature (°F)

Sample Code 2-17. Compiled Month Data Field Headers.

And the statistical data:

Mayaguez, April, 2001,	to,	November, 2001
Year,	Month,	Low Temperature (°F), Avg Temperature (°F), High Temperature (°F)
2001,	4,	65.7 , 84.6 , 92.4
2001,	5,	66.8 , 86.7 , 95.7

Sample Code 2-18. Compiled Month Data File.

Chapter 3

XML Data

The following chapter describes the XML structures developed. The structures will be built starting from the most general XML tags and then subsequent tags will be added into their position of the structure. Every XML data type will also be presented in a diagram which illustrates its hierarchical structure.

XML stands for Extensible Markup Language. It is defined as “*set of rules for defining semantic tags that break a document into parts and identify the different parts of the document. It is a meta-markup language that defines a syntax used to define other domain specific, semantic, structured markup languages*”. [4]

The structure of our XML files will follow the guidelines set forth by the W3C organization. XML files have a different structure than CSV files, but just as helpful for web interface uses. The W3C intends to present it as the common format for storing data on the web. Once its position is more accepted, web developers will be able to retrieve data from these files to create better and more efficient user interfaces without relying on client server software to generate dynamic WebPages. Web languages such as XSL, XUL, XQL, and UIML have already been developed for such purposes, but web browsers have yet to fully implement them or are not stable enough [4,9,16].

Its structure is more hierarchical than the traditional table format. It should be noted, that the Data files in XML format carry considerable overhead when having to name all the fields and tags for every single reading. So a database of XML data would increase several times the database size.

XML structures and schemas can be bound by Document Type Definitions [4,12]. DTDs are defined as “*a document’s legal structure. It describes the elements and*

attributes available, where and how many times they can occur, how elements can nest, and how elements and attributes can fit together.”[13] These DTDs set forth the structure that an XML document must follow if it is to comply with the standards of the institution that standardized the DTD. This avoids incompatibility problems in XML structures between organizations. An XML document is called “valid” if it meets the constraints defined in a DTD [7]. Each of them must agree on a common DTD to follow, and any document failing to conform to that DTD is disregarded. The structures declared in this work are preliminary, and do not yet contemplate several Fields of important use for the climate research community. Therefore, no DTD is proposed in this document to serve the standard for climate data storage in XML, as such a DTD would invariably be subjected to important changes in the future. Support for changes in existing DTDs is troublesome and leads to confusion between organizations with incongruent schemas yet expect the same data types. In [5] the issue of disparity in XML schemas is addressed, and proposed a prototype system to handle these changes between organizations.

XML structures also facilitate the creation of their software component counterparts. When using a OOP Language, between it is easier to identify the future objects and their responsibilities by modeling them on the XML elements in the database. [8,11] In spite of their Object-type structure however, XML documents still retain the significance in the ordering of elements [12]. So each record in the Data type XML documents, will keep their chronological ordering, even though every record will be have its time mark element.

XML files will follow the same file structure as data files, and the same naming conventions. With the exception that they will carry the corresponding .xml extension.

3.1 Location Data

We start with the Location's information file. As explained in the Data Structure section, this file is contained directly under the Location's directory of the Records Directory. The main tags for defining a Location are:

```
<LOCATION>
</LOCATION>
```

Sample Code 3-1. XML Location Tags.

Next is the general information about the Location. The name is indispensable, while other information can be added at the operator's convenience, if it is of importance to the users:

```
<LOCATION>
  <NAME>Mayaguez</NAME>
  <ALTITUDE>10m</ALTITUDE>
  <LATITUDE>18°13'</LATITUDE>
  <LONGITUDE>67°09'</LONGITUDE>
  <INSTITUTION>UPR-RUM Climatology Department</INSTITUTION>
  <TIME_ZONE>GMT-4</TIME_ZONE>
</LOCATION>
```

Sample Code 3-2. XML Location Information Tags.

The time range of data must be included as well. Inside the Time Range tags, we have two date representations. For the date of the first and last recorded readings for this Location:

```

<LOCATION>
  <NAME>Mayaguez</NAME>
  <ALTITUDE>10m</ALTITUDE>
  <LATITUDE>18°13'</LATITUDE>
  <LONGITUDE>67°09'</LONGITUDE>
  <INSTITUTION>UPR-RUM Climatology Department</INSTITUTION>
  <TIME_ZONE>GMT-4</TIME_ZONE>
  <TIME_RANGE>
    <FIRST_READING_DAY>
      <DAY>18</DAY>
      <MONTH>10</MONTH>
      <YEAR>2000</YEAR>
    </FIRST_READING_DAY>
    <LAST_READING_DAY>
      <DAY>5</DAY>
      <MONTH>6</MONTH>
      <YEAR>2003</YEAR>
    </LAST_READING_DAY>
  </TIME_RANGE>
</LOCATION>

```

Sample Code 3-3. Location Time Range Data Tags.

Finally, there are the Fields tags. These contain information about the Location's Fields such as the user defined valid readings ranges, and statistical information. The data is Field specific, and each WXField class must implement it's own methods of reading and writing it's information into the XML data file. All the statistics and readings stored here are in their respective Most Precise Unit System. Here is an example of a few field tags:

```

<LOCATION>
  <NAME>Mayaguez</NAME>
  <ALTITUDE>10m</ALTITUDE>
  <LATITUDE>18°13'</LATITUDE>
  <LONGITUDE>67°09'</LONGITUDE>
  <INSTITUTION>UPR-RUM Climatology Department</INSTITUTION>
  <TIME_ZONE>GMT-4</TIME_ZONE>
  <TIME_RANGE/>
  <FIELDS>
    <TEMPERATURE>
      <LIMITS>
        <HIGH>120</HIGH>
        <LOW>50</LOW>
      </LIMITS>
      <STATS>
        <HIGH>
          <READING>95.4</READING>
          <DATE>
            <DAY>4</DAY>
            <MONTH>6</MONTH>
            <YEAR>2001</YEAR>
          </DATE>
        </HIGH>
      </STATS>
    </TEMPERATURE>
    <WIND_SPEED>
      <LIMITS>
        <HIGH>100</HIGH>
      </LIMITS>
      </WIND_SPEED>
      <STATS>
        <HIGH>5.6</HIGH>
        <DATE>
          <DAY>18</DAY>
          <MONTH>2</MONTH>
          <YEAR>2002</YEAR>
        </DATE>
      </STATS>
    </FIELDS>
  </LOCATION>

```

Sample Code 3-4. XML Location Field Information Tags.

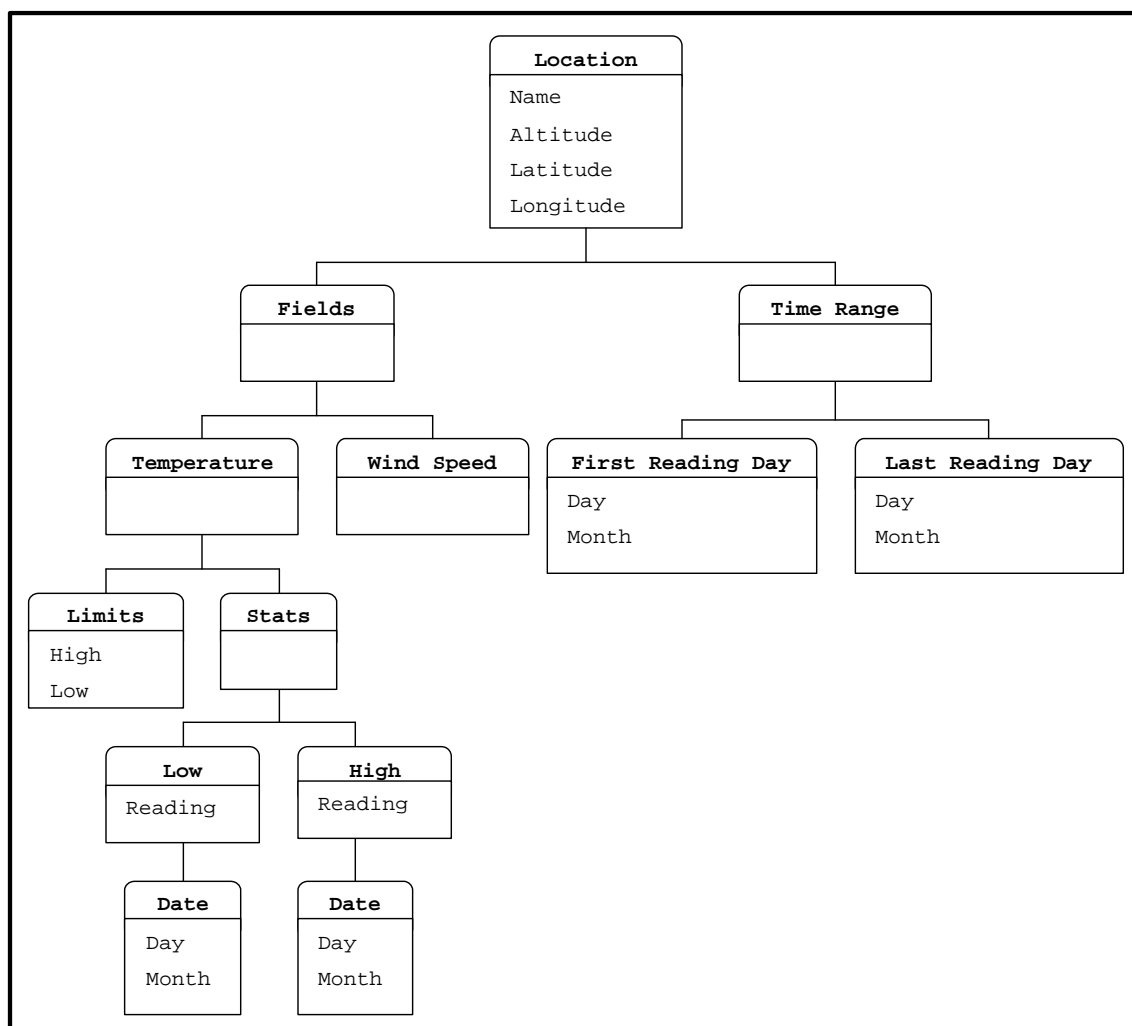


Figure 3-1. Location Data structure

3.2 Readings Data

The Readings Data files will begin with the following encompassing tags:

```
<DAYDATA>
</DAYDATA>
```

Sample Code 3-5. XML Day Data Tags.

The Location and date information is included.

```
<DAYDATA>
  <LOCATION>Mayaguez</LOCATION>
  <DATE>
    <DAY>6</DAY>
    <MONTH>5</MONTH>
    <YEAR>2001</YEAR>
  </DATE>
</DAYDATA>
```

Sample Code 3-6. XML Day Data Information Tags.

Then the data for the day. The data is separated into records; where each records carries a corresponding Time tag and tags with the Field names for the respective readings at that time:

```

<DAYDATA>
  <LOCATION>Mayaguez</LOCATION>
  <DATE/>
  <DATA>
    <RECORD>
      <TIME>
        <HOUR>0</HOUR>
        <MINUTES>10</MINUTES>
      </TIME>
      <TEMPERATURE>68.7</TEMPERATURE>
      <WIND_SPEED>0</WIND_SPEED>
      <RELATIVE_HUMIDITY>89.5</RELATIVE_HUMIDITY>
    </RECORD>
    <RECORD>
      <TIME>
        <HOUR>0</HOUR>
        <MINUTES>20</MINUTES>
      </TIME>
      <TEMPERATURE>68.3</TEMPERATURE>
      <WIND_SPEED>0.2</WIND_SPEED>
      <RELATIVE_HUMIDITY>89.9</RELATIVE_HUMIDITY>
    </RECORD>
    <RECORD>
      <TIME>
        <HOUR>0</HOUR>
        <MINUTES>20</MINUTES>
      </TIME>
      <TEMPERATURE>67.9</TEMPERATURE>
      <WIND_SPEED>0.5</WIND_SPEED>
      <RELATIVE_HUMIDITY>88.3</RELATIVE_HUMIDITY>
    </RECORD>
  </DATA>
</DAYDATA>

```

Sample Code 3-7. XML Day Data File.

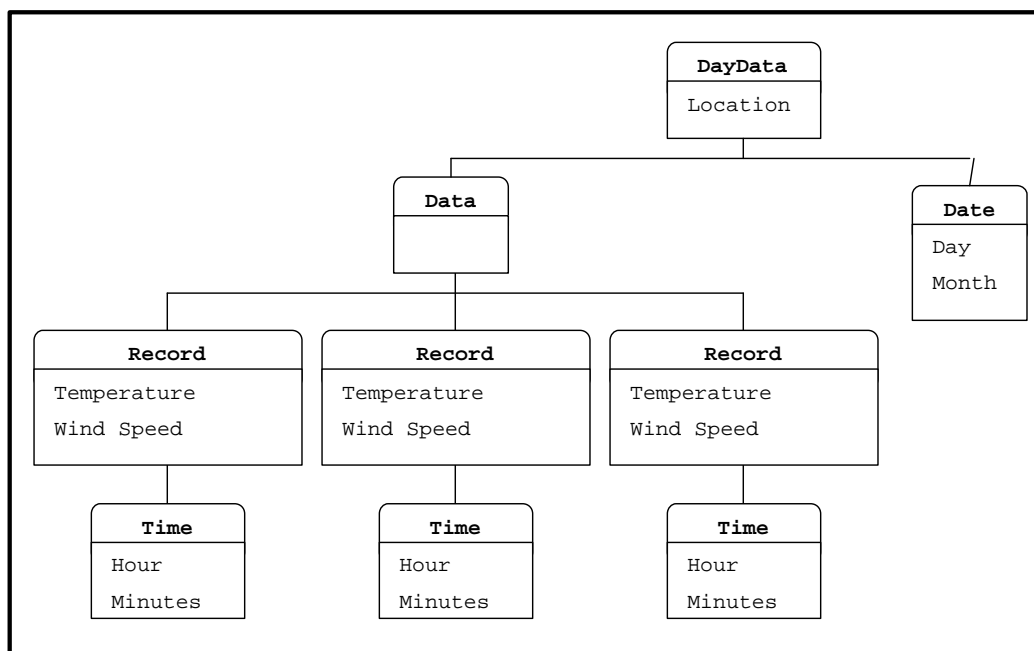


Figure 3-2. Day Data structure

3.3 Month Data

Monthly Data will be delimited by:

```
<MONTHDATA>
</MONTHDATA>
```

Sample Code 3-8. XML Month Data Tags.

Followed by the Location and month information:

```
<MONTHDATA>
  <LOCATION>Mayaguez</LOCATION>
  <DATE>
    <MONTH>10</MONTH>
    <YEAR>2002</YEAR>
  </DATE>
</MONTHDATA>
```

Sample Code 3-9. XML Month Data Information Tags.

The data is then organized into records, each of which identified by the day of the month. Every record is then divided into the Location's Fields that holds all the relevant statistics of that Field for the current day.

```

<MONTHDATA>
  <LOCATION>Mayaguez</LOCATION>
  <DATE/>
  <DATA>
    <RECORD>
      <DAY>1</DAY>
      <TEMPERATURE>
        <LOW>65.4</LOW>
        <AVG>86.6</LOW>
        <HIGH>97.4</HIGH>
      </TEMPERATURE>
      <PRECIPITATION>
        <HIGH>2</HIGH>
        <TOTAL>18</TOTAL>
      </PRECIPITATION>
      <WIND_SPEED>
        <AVG>1.7</AVG>
        <HIGH>3.5</HIGH>
      </WIND_SPEED>
    </RECORD>
    <RECORD>
      <DAY>2</DAY>
      <TEMPERATURE>
        <LOW>65.2</LOW>
        <AVG>86.7</LOW>
        <HIGH>97.1</HIGH>
      </TEMPERATURE>
      <PRECIPITATION>
        <HIGH>0</HIGH>
        <TOTAL>0</TOTAL>
      </PRECIPITATION>
      <WIND_SPEED>
        <AVG>1.2</AVG>
        <HIGH>2.7</HIGH>
      </WIND_SPEED>
    </RECORD>
  </DATA>
</MONTHDATA>

```

Sample Code 3-10. XML Month Data File.

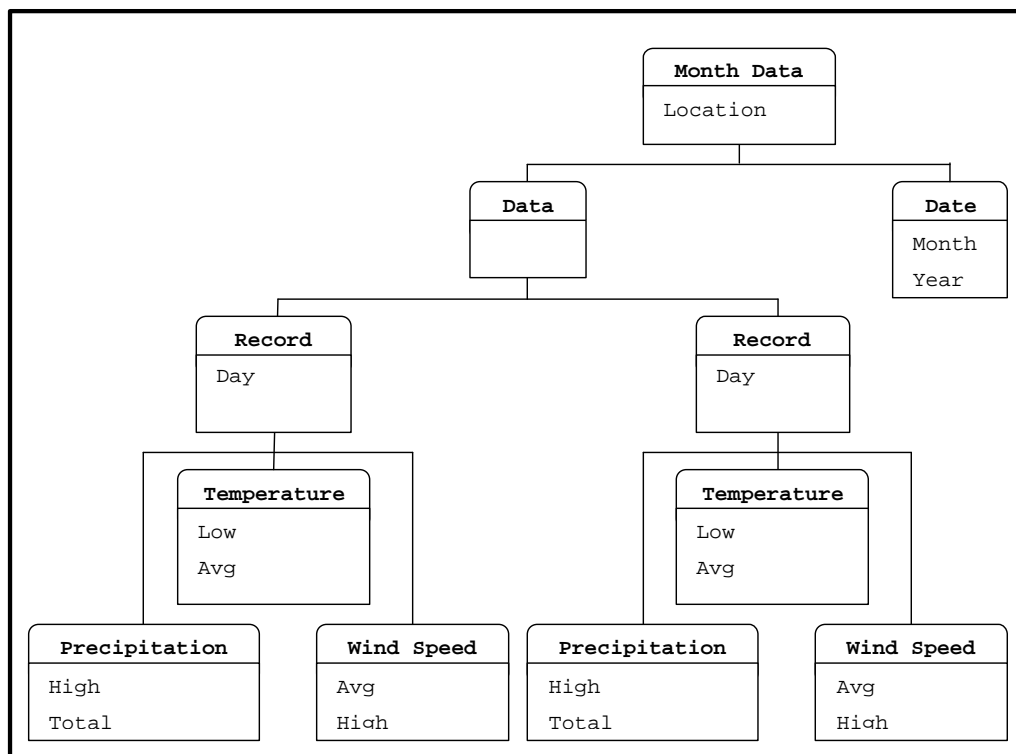


Figure 3-3. Month Data Structure

3.4 Year Data

The Year Data XML file follows the same structure as the Month Data XML file. It begins with the following encompassing tags:

```
<YEARDATA>
</YEARDATA>
```

Sample Code 3-11. XML Year Data Tags.

The Location and year information:

```
<YEARDATA>
  <LOCATION>Mayaguez</LOCATION>
  <YEAR>2001</YEAR>
</YEARDATA>
```

Sample Code 3-12. XML Year Data Information Tags.

And the data, where the records are organized by month numbers:

```

<YEARDATA>
  <LOCATION>Mayaguez</LOCATION>
  <YEAR>2001</YEAR>
  <DATA>
    <RECORD>
      <MONTH>1</MONTH>
      <TEMPERATURE>
        <LOW>64.2</LOW>
        <AVG>82.7</AVG>
        <HIGH>94.1</HIGH>
      </TEMPERATURE>
      <HEAT_INDEX>
        <LOW>70.1</LOW>
        <AVG>85.30</AVG>
        <HIGH>96.5</HIGH>
      </HEAT_INDEX>
      <WIND_DIRECTION>
        <AVG>45.3</AVG>
      </WIND_DIRECTION>
    </RECORD>
    <RECORD>
      <MONTH>2</MONTH>
      <TEMPERATURE>
        <LOW>62.2</LOW>
        <AVG>82.9</AVG>
        <HIGH>92.1</HIGH>
      </TEMPERATURE>
      <HEAT_INDEX>
        <LOW>70.4</LOW>
        <AVG>83.2</AVG>
        <HIGH>97.8</HIGH>
      </HEAT_INDEX>
      <WIND_DIRECTION>
        <AVG>91.5</AVG>
      </WIND_DIRECTION>
    </RECORD>
  </DATA>
</YEARDATA>

```

Sample Code 3-13. XML Year Data Tags.

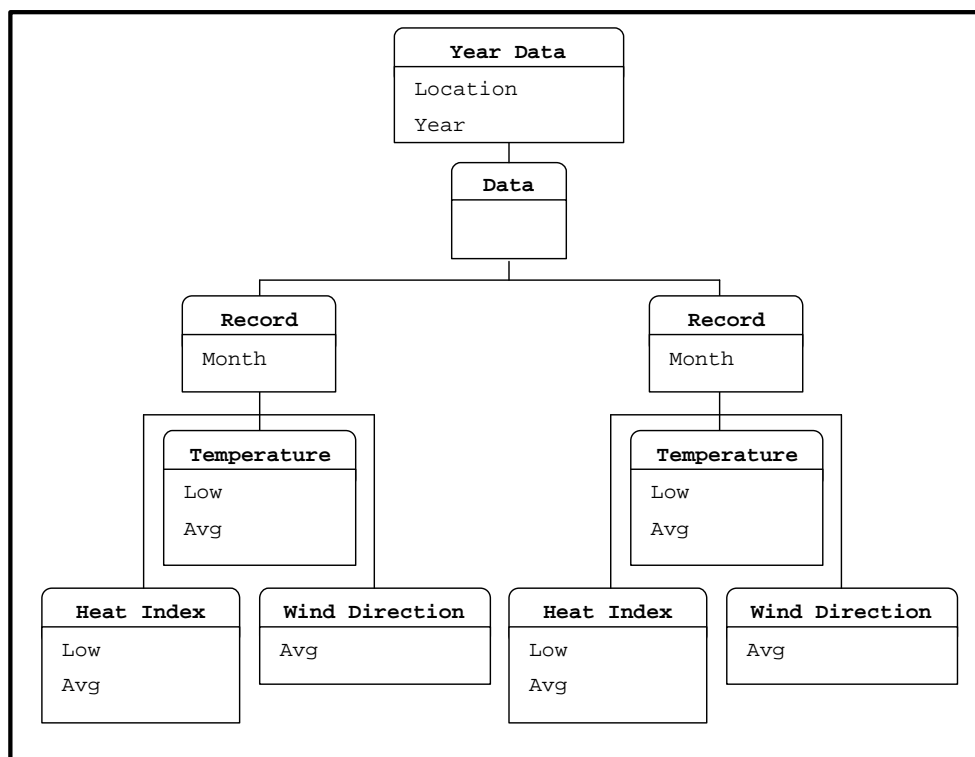


Figure 3-4. Year Data Structure

Chapter 4

Software

This chapter describes the application. First a mention of the software used to develop the application. Then a description of each sub-component and how it functions in the application.

The software was developed using Sun's Java 2 Standard Edition (J2SE) programming language. The object-oriented functionality that characterizes this language makes it ideal for separating the distinct modules that make up the system. These modules correspond to the abstract conceptual elements in the system. In the Java environment, these modules are called "classes". Classes are abstractions of objects. Java's platform independence also facilitates its portability into other sites.[3]

The application has been developing using Sun's Java Development Kit (JDK), freely available for download at their web site (<http://java.sun.com>). The JDK provides the basic classes to develop Java applications, and the Java Runtime Environment (JRE) for executing them. For editing the software code, I've used the text editor Textpad, available for download at <http://www.textpad.com>.

The entire application is separated into Location classes, Data Management, Fields Classes, Web User Interface and the Database. These all rely on each other to retrieve and display the data for the user. The goal is to have complete functional autonomy from these components, so that modifications to one component need not adjust any of the others.

The Location classes represent a Location from which data is received. For every Location in the system, there must be a corresponding Location class. These classes must store the Fields that the Location supports, its keyword property, and most importantly,

the function or algorithm for retrieving new data. They are not in themselves part of Data Management, since they can be deleted or added by future operators, as they deem necessary, with minimal adjustment to the Data Management components. Future operators can add new Location's given that they satisfy the requisites demanded by the Data Management components (such as providing a function for retrieving new data).

The Data Management components are those that interact between retrieving fresh data from the Location classes, and the Web User Interface. It is its responsibility to store fresh data into the Database and retrieve the archived data for use by the Web User Interface components. It is also in charge of generating compiled data when requested by the Web User Interface.

The Database is the collection of archived data described in the Data Structure chapter. As explained, it is stored in an ASCII CSV format, which allows its permanence independent of any other software. It is handled only by the Data Management component.

The Field classes are those that represent Field or sensors for which a Station collects data for. Like the Location classes, they are not a part of the Data Management component and can be added or deleted by the operators, depending on what their collection of Locations support. These classes provide the application with several Field specific functions. Some of these not indispensable, such as a Quality Control Algorithm for that field, a graph or chart generator. But some are requisite, such as the Field's title, a Unit System conversion function (for converting between metric and british units), and which is it's Most Precise Unit System.

Finally there is the Web User Interface component. This is the software running on the web server that displays the data to the user. It provides the user with the necessary forms so he/she can request data. And works with the Data Management component to retrieve the data it needs and display it to the client in a user-friendly interface. It also

works with Location classes for displaying information about the particular Location the user is viewing data for. And Field classes for generating graphs and charts.

Figure 4.1 shows us how these components work together:

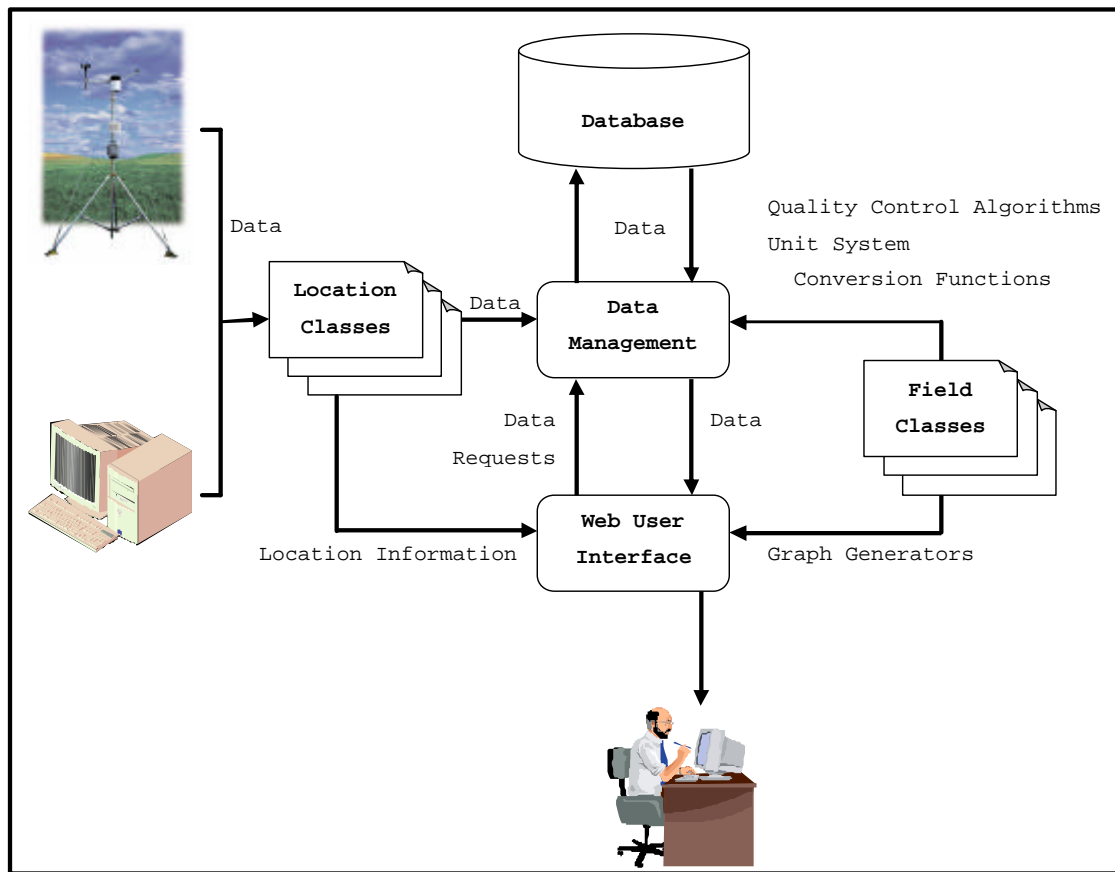


Figure 4-1. Software components

What follows is a description of the classes implemented in each of these components. A UML diagram of how each of these classes relate to each other will accompany each description.

4.1 Support Classes

Some classes do not belong or fall into any of the previous components. They are used by all of them as support classes to facilitate the representation of certain concepts or to group sets of similar elements with a common quality. These support classes are:

- WXTime
- WXDay
- WXRecord
- WXRecordSet
- WXStatRecord
- WXStatRecordSet

The first two, WXTime and WXDay, are time representations. WXTime represents a time of day when sets of readings were taken. WXDay represents a Day for statistical data. WXDay can also be used to represent a month or a year, in each case, the unnecessary date field is ignored; such as they Day field for a month, or the Day and Month field for a year. They provide functionality for operations that constantly are required on the time and day they represent, by several classes on all components. Comparison operations, such as asking which time or day is before or after it, retrieving commonly used time and day representations, such as the 24-hour time format, or the 8 digit date representation.

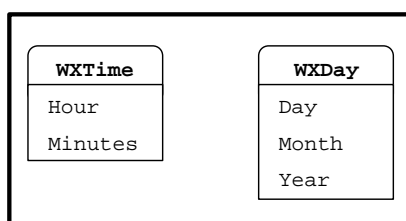


Figure 4-2. WXTime and WXDay classes

WXRecord represents a record of data. A set of readings taken at a particular time. The time is represented by a WXTime object, and the readings stored in a one

dimensional floating point array. It provides some functionality on comparing records, such as which record is before or after the current one, counting the number of invalid readings, and retrieving a particular reading from the set.

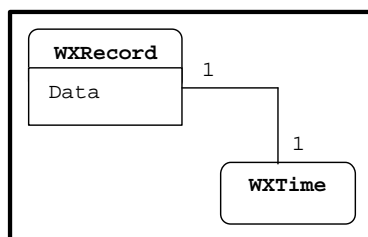


Figure 4-3. WXRecord UML diagram

WXRecordSet is a set of **WXRecords** that contain all the readings data for one day. A **WXDay** object represents the Day, and it holds a collection of **WXRecord** objects. This provides functionality on a set of **WXRecords**, like percentage of invalid readings in the data set, retrieving a set of data for one particular field, iterating through records by their time order. It also provides functions for retrieving statistical information of the data set it contains. Such as the highest or lowest reading for a particular field.

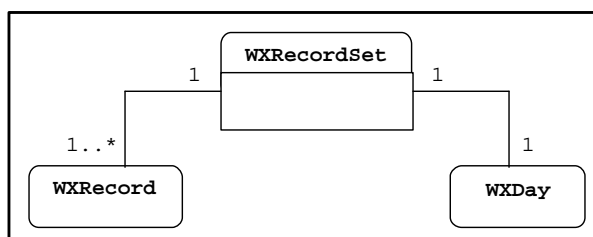


Figure 4-4. WXRecordSet UML Diagram

WXStatRecord is a record of statistical data. It can be used to hold statistical data for a day or a month. A **WXDay** object represents the day or month, and the data is in a two-dimensional floating-point array. Two-dimensional, because there are several statistics for every field.

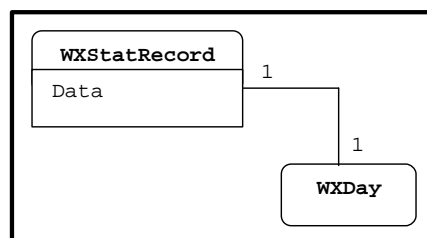


Figure 4-5. WXStatRecord UML Diagram

Finally, **WXStatRecordSet** is a set of **WXStatRecords**. It represents the set of statistical records for a month, or a year. If the **WXStatRecords** correspond to daily statistics, then the **WXStatRecordSet** should represent a monthly set; if the **WXStatRecords** correspond to monthly statistics, then it should represent a yearly set. This class also implements functions for calculating statistical information from the data it contains.

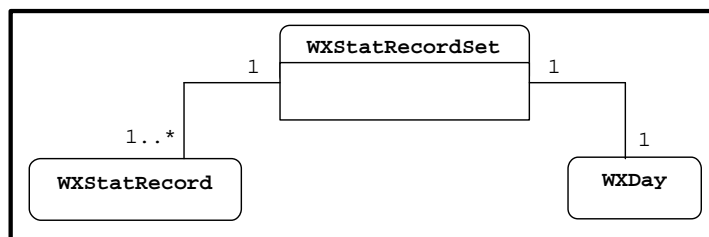


Figure 4-6. WXStatRecordSet UML Diagram

4.2 Field Classes

Field classes represent sensors or weather data that the stations records data from. They can correspond to actual sensors implemented in the weather station, or other Fields that can be derived from data taken by sensors in the weather station. Every Field to be supported by a Location needs to have it's own Field class implemented. They must provide certain functionality that is requisite to its successful operation in the application. Like the Location class, there is already an abstract **WXField** class that sets the standard for all Field classes to follow. They must all extend **WXField**, and implement the abstract methods declared there. One of these methods is `convert(Reading, UnitSystem)` for converting a reading from Metric to British and viceversa.

Other generic methods are already declared in `WXField` that can be overwritten in a subclass for better functionality. For example, the method for quality control `validate(DataSet)`. In `WXField`, it simply limits itself to discarding those readings not within the valid readings range for that Field. But a more complex algorithm for eliminating erroneous readings can be written into the particular Field class, which will then override the parent function and apply its own algorithm without any change to the rest of the Data Management component.

Graph generators are also methods that might be overwritten. The generic method in `WXField` generates a standard line graph of data, but their methods can be overwritten in the particular Field class if a more appropriate graph is suitable for that Field. For example, a wind rose chart can be inserted into the `WindDirection` class for graphs, or a bar chart for `Precipitation`. Overriding these methods takes place in the Field subclass and requires no modification to the Web User Interface classes. However, if more than one graph option is to be desired for one particular Field, such as having a line graph and a wind rose chart available for the user, and then modifications need to be made at the Web User Interface Components to accommodate the extra option for that Field.

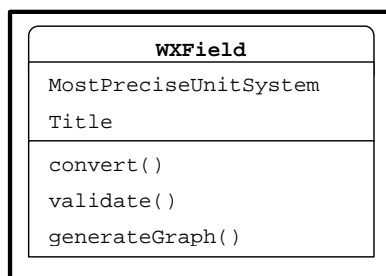


Figure 4-7. `WXField` class

4.3 Location Classes

The Location classes represent weather stations at different points of the island. They hold information about the station, such as the Keyword, the Fields it supports, its time interval of readings. Other information, such as the Location's Name, its

geographical coordinates, the institution that operates it, etc. is saved in the Location's XML information file, and can be changed at will by the operators without any consequence to the functionality of the application. The Fields it supports are represented by a collection of WXField objects. An operator must create one Location class for every Location in the database. Though different Locations may use the same algorithm for retrieving data, such as one third party data provider offering data for several weather stations in the same database format, every Location needs a unique Keyword to identify it. And this has to be written into the Location class. It also provides the time range of archived data for this Location, represented by two WXDay objects, the date of the first and last recorded reading.

The most important aspect of a WXLocation class is its method for reading new data. Given that every weather station is unique, or that every third party database that offers us their data is distinct, the application could never anticipate the format the data would be offered to us from its source. It is for this reason, that every Location class must implement a function for retrieving data from the source, and pass it along to the Data Management components in the format it would expect it to. The Data Management components can then store the data in the database and manipulate it in its own format.

To do this, we've declared an abstract class called Location. This abstract class declares, but doesn't implement, the method `getReadingsFromSource(Day)`. This method takes a Day as a parameter, the day we want to retrieve data for, and it returns all the readings for that day in a WXRecordSet object, the type of object used by the Data Management classes to manipulate day readings. All Locations to be used in the application, must extend this parent class and implement a method for retrieving fresh data.

This absolves the Data Management classes from concerning itself with the particulars of handling different data sites. Whenever it needs to retrieve fresh data from a Location's source, it merely calls the Location's method, and knows it will return the

data in the common format it's using. That way when changes occur in the third party database, or new sites are to be added, only the affected Location class needs to be adjusted, the rest of the system continues to function without need for any changes.

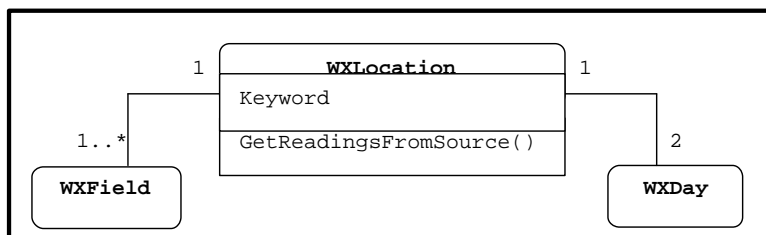


Figure 4-8. WXLocation UML Diagram

4.4 Data Management Classes

These are the classes that manipulate the database. They act as a mediator for the Web User Interface and the database. When a user requests data through the website, it is these classes that retrieve the data from the database and provide it to the Web User Interface components; and compile data if it has been requested. They also retrieve new data from the Location's sites periodically, and validate it before storing them into the database.

The classes that make up the Data Management component are:

- WXData
- WXDayData
- WXMonthData
- WXYearData
- WXCompiledReadingsData
- WXCompiledDayData
- WXCompiledMonthData

WXDayData is the class that represents a day's worth of readings for a particular Location. The day is represented by a WXDay object, and the Location by a WXLocation object. It holds its data in a WXRecordSet object, and the statistical information of its data in a WXStatRecord object. WXDayData objects can be created either from new data, in which case it invokes its Location's method for retrieving fresh data, or from the data archived in the database, it can then store its data into the database. Creating a WXDayData object with fresh data and then storing it into the database is the way to increase the database's size with new data. It also provides the statistics of the data it contains.

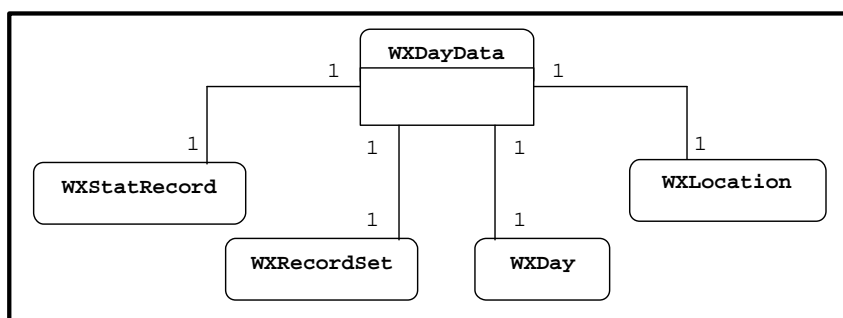


Figure 4-9. WXDayData UML Diagram

WXMonthData represents statistical data for every day in a month for a particular Location. A WXDay object represents the Month, disregarding its Day field and the Location by a WXLocation object. It holds its data in a WXStatRecordSet, given that all its records are WXStatRecords (WXStatRecord of every day in the month), and the month's statistical information in a WXStatRecord of its own. It can be created from the Readings files, or by the archived data in Month Data files. By creating a WXMonthData object from the Readings archived files, and the storing it into the database is how MonthData files are updated.

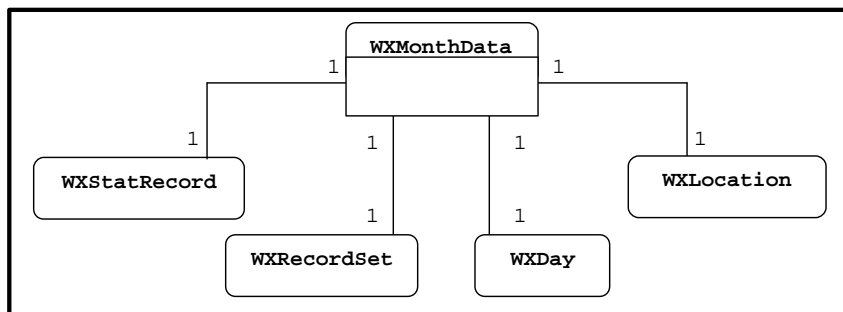


Figure 4-10. WXMonthData UML Diagram

WYYearData holds the statistical information of every month in the year. A WXDay object, disregarding the Day and Month fields, and the Location by a WXLocation object represents the year. Its data is stored in a WXStatRecordSet, all the WXStatRecords of every month in that year, and it's own statistical information in a WXStatRecord. This object can be created from archived Month Data files, or from archived Year Data files. By creating a WYYearData object from archived Month Data files and storing it's data in the database is how Year Data files are updated.

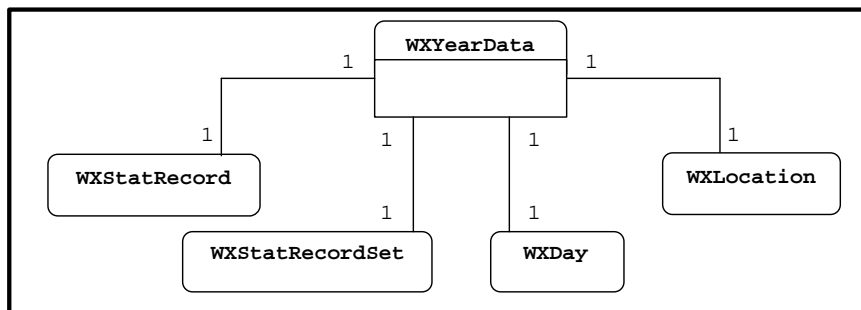


Figure 4-11. WYYearData UML Diagram

Compiled Data files don't directly manipulate the database. They rely on the WXDayData, WXMonthData, and WYYearData to retrieve the data for them. The data they gather is not stored into the database, but into a Compiles Directory, that makes the file available for download to the client through the webpage.

The WXCompiledReadingsData class creates the CompiledReadings files. When created, it requires a Location, and two WXDay objects, the time range to compile data

for. Its data is stored as a collection of `WXDayData` objects for all the days in between the time range. It also calculates the relevant statistics for the data it has and stores it in a `WXStatRecord` of it's own.

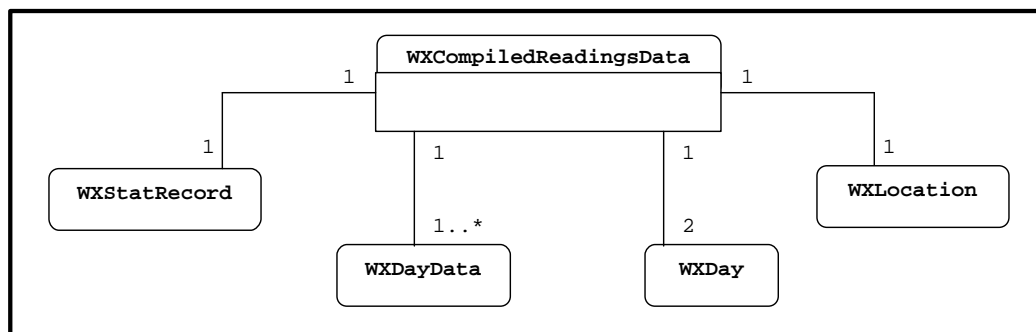


Figure 4-12. `WXCompiledReadingsData` UML Diagram

The `WXCompiledDayData` stores statistical data on a daily basis, for all the days between a give time ranges for a `Location`. The time range is represented by two `WXDay` objects, and the `Location` by a `WXLocation` object. Its data is stored as a collection of `MonthData` objects, for all the months in between they two `WXDay` objects inclusive. It would seem that a collection of `WXDayData` objects should be more appropriate. But for reasons of performance, it is preferable to retrieve the statistical data from the `MonthData` files. The information required is stored in them, and it avoids having to read all the readings in the `Day Data` files, for what could be an extensive time range, to then retrieve their statistical information. So the `WXCompiledDayData` will retrieve they day statistical data from the `WXMonthData` objects, in spite of the days in the month before the beginning of the time range, and the days in the month after the end of the time range, will never be used. Its own statistical data, for the given time range, will be stored in a `WXStatRecord` object.

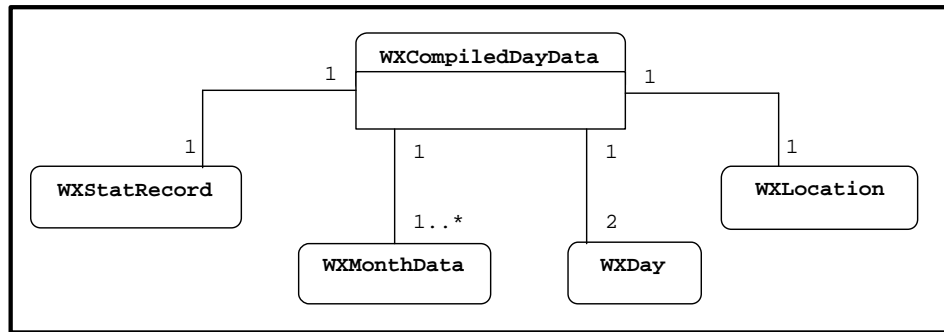


Figure 4-13. WXCompiledDayData UML Diagram

Finally, the **WXCompiledMonthData** gathers statistical information for all the months within a specified time range at a particular **WXLocation** object. It requires a **WXLocation** object, and two **WXDay** objects, to set the time range of data to be gathered. It builds its data from **WXYearData** objects, for the same reason as explained above. After retrieving all its data, it performs the relevant statistics calculations on its data, and stores it in a **WXStatRecord** object.

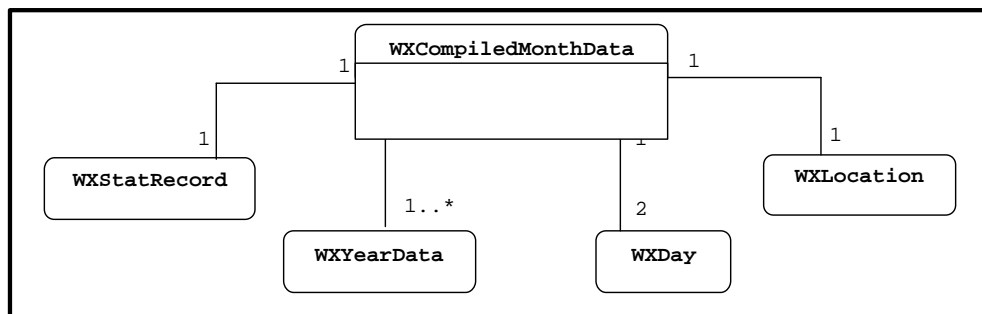


Figure 4-14. WXCompiledMonthData UML Diagram

4.5 Web Interface Classes

These are classes that serve to take requests of data from the user and display the requested data in the form they wish to view it. Each of them generates on webpage where the user views or requests data. They don't access the database directly, but rely on the Data Management components to retrieve or generate the data for them. The following classes fall into this category:

- LocationTable
- DayDataTable
- MonthDataTable
- YearDataTable
- DayDataPrintTable
- MonthDataPrintTable
- YearDataPrintTable
- CompiledReadingsDataTable
- CompiledDayDataTable
- CompiledMonthDataTable
- CompiledReadingsDataPrintTable
- CompiledDayDataPrintTable
- CompiledMonthDataPrintTable
- DayGraph
- MonthGraph
- YearGraph

The client first accesses the data system through the LocationTable page. This page displays all the information stored about the Location. It's Name, Altitude, Geographical Coordinates, etc... It might also display relevant statistics the operator may deem important to visiting users, such as the highest and lowest recorded temperatures for this Location.

It also provides the client with forms for requesting data. There are two essential forms that the LocationTable page must display. First, the Data Request Form, this must allow the user to input a date, a unit system, and a Data Type of date he wants to view. The Data Type can be readings for the day, daily Statistics for the month, or monthly statistics for the year of the date given. Through this form, the user is taken to the DayDataTable, MonthDataTable, or YearDataTable, depending on the option he selected. The second form is the Compile Data form. Here the user must be allowed to select which Fields he wishes to compile data, from a list of all the Fields supported by this Location. He must input two dates, the time range of data he wishes to compile, the unit system, and the Data Type he wishes to compile. The Data Type can be all readings, daily statistics, or monthly statistics, for all days or months within the time range.

The DayDataTable, MonthDataTable, and YearDataTable, display data contained in WXDayData, WXMonthData, and WXYearData objects respectively. DayDataTable presents the user with a table view of all the readings for the date he requested; MonthDataTable the statistical data for every day in the month of the date; and YearDataTable all the relevant statistics for every month in the year of the data requested by the user. Each of them also displays the relevant statistics of the data they store. Common elements within all these tables is a link to open the current LocationPage, given that it is possible that the user may have arrived at this page not through the LocationPage; links to download the data they are viewing in their corresponding CSV file; a link to quickly switch unit systems; and a link to their corresponding PrintTable page. Navigation links and forms must be available as well, so the user can jump from different dates or locations without returning to the LocationPage. Also, they must provide links at every Field header for retrieving graphs of one particular Field of data.

DayDataPrintTable, MonthDataPrintTable and YearDataPrintTable are the printer friendly versions of their web counterparts. They display the same data as DayDataTable, MonthDataTable and YearDataTable, in a format more suitable for printing. This means no color, no links, and only the relevant data in the table.

The CompiledDataTables display data that has been compiled for the user. They implement almost the same interface as the DayDataTable, MonthDataTable and YearDataTable. With the same links and relevant statistics of their data, with the exception of graph links.

Their printer friendly versions are implemented in the CompiledDataPrintTables. They follow the same standard set by the DataPrintTables, for displaying the compiled data in a printer friendly manner.

Finally, we have the GraphTables. These display a graph of a Day, Month or Year for a particular field. They must provide the user with the graph and a navigation form to scroll or jump through the days, months, or years of data they are viewing. Links to their corresponding DayDataTable, MonthDataTable or YearDataTable must be available as well.

Chapter 5

Software Operation

In this chapter is explained the basic operation of the application. The mechanism for retrieving data from the weather station or the third party database, and how it is stored into the database. It also describes how the process of quality control takes place within the Field classes. Finally it explains how the Web Interface components receive data requests from clients and display it to the user in the various tables or graphs.

5.1 Data Management

Data retrieval begins at the Location class. The abstract parent class WXLocation defines an abstract method `getReadingsFromSource()` which takes a WXDay object as a parameter. This method has to be implemented by every single location in the system. As it provides the mechanism for retrieving data directly from the source of the location. And this is something unique for every Location and impossible to predict by any developer. This method then retrieves the data from the station or the third party database, in the format it is stored or received through the station's institution, and collects the readings for the day given as a parameter. Each collection of readings for a given time of day are stored in a WXRecord object. This object holds a WXTime object (the time of day) and all the readings corresponding to that time of day. All the WXRecord objects for the all readings of that day are then wrapped in a WXRecordSet object. This is then returned by the method to the calling class.

The WXDayData object handles the Location's data for one day. We create a WXDayData object with its initializer, which takes as parameters a Location object (the location from where we want the data), a WXDay object (the day we want data for), and a value for the source of the data. These are declared in the WXData class and can either

be: `WXData.SOURCE`, or `WXData.ARCHIVE`. `WXData.SOURCE`, tells the `WXDayData` object to retrieve the data directly from the location's source. This would be done through the Location's `getDataFromSource()` method, providing the `WXDay` object given in the `WXDayData` initializer. `WXData.ARCHIVE`, would have the `WXDayData` object retrieve data from the files in the archives, stored in the manner explained in the Data Structure section. The mechanism for reading the data from the archive files is written into the `WXDayData` object.

After the data is retrieved, from either source, the data is run through the quality control algorithms for each field the Location supports. The Location class' `getFields()` method returns an array of `WXField` object with all the Fields the Location supports. The parent abstract class `WXField` defines a method called `validate()`. This method is implemented in the parent class, but it limits itself to simply removing all the readings that are not within the field's range of acceptable data. Every `WXField` object has a range of acceptable data, which mark the lowest and highest limits that a reading has to be within to be considered valid. These values are field specific, and are stored in the Location's XML data file. The Location class reads them at the moment it is initialized and passed as parameters when initializing the individual `WXField` objects. Most fields require the high and low values be stored in the XML file. Such as Temperature, its range of valid temperature readings can vary from location to location. However, Wind Speed and Solar Radiation, always have a low limit of 0, and have this low limit value by default. Wind Direction, always has a range of 0 to 360. In this last case however, it is possible that certain locations may use numbers beyond this range to mark acceptable readings. For instance, a Wind Direction reading of -10° can be used to represent 350° , or 390° to represent 20° . This will be dealt by the Location's `getDataFromSource()` method, that will make the necessary conversion into a valid reading within the limits, if the particular location is known to make these kind of readings. The `validate()` method takes the `WXRecordSet` from the `WXDayData` object, and iterates through all the `WXRecord` objects within it comparing the readings with the limits in the corresponding `WXField`

object. All readings not within the acceptable range are replaced with the invalid reading indicator `WXData.INVALID_READING`.

The `validate()` method can be overwritten by any of the subclasses of `WXField`. This allows for more complex field specific algorithms of quality control. The same algorithm for removing invalid readings in a temperature data set would be different from a wind direction data set for instance. By simply overriding the `validate()` method in a subclass of `WXField`, the application uses that algorithm to purge the data of invalid readings without making any modifications to any Data Management classes.

The data is taken through the quality control algorithms regardless of whether it was just acquired from the station's source or from the archives. This would allow a future developer to modify one of the quality control algorithms and be able to use the data in the archive to update the whole database.

After having validated the data, the `WXDayData` object calculates the relevant statistics of its data. The algorithms for calculating these statistics are implemented in the `WXRecordSet` class. It defines a method called `getStat()` which takes as parameter two indexes. The first is the field index of data to calculate, such as the Temperature index, or Precipitation Index, and the other is a statistic index, which indicates which statistic is to be calculated. There are only four statistic indexes and they are defined in the `WXData` class. They are `WXData.LOW`, `WXData.AVG`, `WXData.HIGH`, and `WXData.TOTAL`. So when calling `getStat(TemperatureIndex, WXData.LOW)` on a `WXRecordSet`, I am asking for the "lowest reading of the temperature field".

Every `WXField` must also implement a `hasStat()` method. This method tells us which statistics are relevant for the Field. It takes as parameter one of the statistical indexes and returns true or false depending on whether that statistic is relevant to the Field. For instance, a Temperature object would return true on `hasStat(WXData.LOW)` and on `hasStat(WXData.HIGH)` but false on `hasStat(WXData.TOTAL)`. Likewise Precipitation

would return `false` on `hasStat(WXData.LOW)` and `true` on `hasStat(WXData.TOTAL)`. After performing these calculations, all the statistics for the day are stored in a `WXStatRecord` object.

`WXRecordSet` provides four methods for calculating these statistics. These methods are `findLow()`, `findHigh()`, `getAvg()`, and `getTotal()`. Each of them requiring a field index for the data they are going to find or calculate. `findLow()` and `findHigh()` return the lowest and highest reading of the field given. `findAvg()` returns the average readings in the data set, this is the sum of valid readings divided by the number of valid readings in the data set. `findTotal()` returns the sum of all the valid readings in the data set.

An exception has to be made in the case of a Wind Direction data set. The only relevant statistic for Wind Direction is `WXData.AVG`, or the “General Wind Direction”, and it needs a unique averaging algorithm, implemented in the `WindDirection` class, because it’s readings are given as angles and they cannot be averaged like any other data set. The formula used to find this general direction of angles is:

$$Avg = atan \left(\frac{\sum_{i=0}^n \sin(X_i) / n}{\sum_{i=0}^n \cos(X_i) / n} \right)$$

Where X is the set of Wind Direction readings and n is the number of valid readings.

Whenever `WXDayData`, `WXMonthData`, or `WXYearData` is iterating through the fields and arrives at a `WindDirection` field, it avoids using the `findAvg()` in the `WXRecordSet` class and resorts to the `WindDirection.findAvg()` method.

It would seem like the same situation for `validate()`, where each field needs its own validating algorithm. But this is the only case (so far) where one of the statistic finding algorithms needs to be specially written for a Field. And no others are anticipated. So for the time being, it is easier to simply modify the Data Management classes to accommodate this unique situation.

After this the `WXDayData` object is initialized and ready to provide data for that day and Location. It can be used to provide data for generating a webpage of day data, or for drawing a graph. The way the common database is updated is by creating a `WXDayData` object using `WXData.SOURCE` (retrieving fresh data from the source) and then saving the data into the common database using its `saveData()` method.

The `saveData()` method takes one value as a parameter, the unit system to save the data in. It can take one of these three values: `WXData.BRITISH`, `WXData.METRIC`, or `WXData.MOST_PRECISE`. `WXData.MOST_PRECISE` is the value used to store data into the Database. This value tells `WXDayData` to save the data in the most precise unit system of every field. A value of `WXData.METRIC` or `WXData.BRITISH`, is an indicator that data is being saved in one specific unit system because a user has requested data for this day and may want to download the CSV file of the data. So if any of these last two values are used, the file is stored in the Compiles Directory in the unit system specified. The file is stored there temporarily and deleted after a fixed amount of time.

The system is designed to always expect the readings to be in the most precise unit system of the Field. Whenever there is a data set for Temperature, it's assumed that the readings are in Fahrenheit, and whenever there is a Wind Speed data set the readings are in meters per second. Conversion to other systems is only done at the moment the data is being stored in a file requested for the user in a specific unit system, or when displaying data in a webpage with a specified unit system. The `WXField` parent class defines an abstract method called `convert()` which takes a reading, and a unit system to convert the reading to. All subclasses of `WXField` must implement this method. Inside this method are two options, depending on which unit system is given. If the unit system given is the field's most precise unit system, then the reading was given in that unit system, so no conversion is necessary and the same reading value is returned. If the unit system given is not the field's most precise unit, then the conversion formula, field specific, is used to convert the reading and return a converted value.

The `WXMonthData` object represents statistics on a day-by-day basis for a whole month. It is initialized in the same way as a `WXDayData`, it expects a `WXLocation` object, a `WXDay` object, and a `Source` value. The `WXDay` object is used, but only its month and year values are necessary, the day value is irrelevant to the `WXMonthData` object. When given a `Source` value of `WXData.SOURCE`, the `WXMonthData` creates `WXDayData` objects for all the days within the month of the `WXDay` object. They are created using the `WXLocation` given as parameter, the current `WXDay` object, and a `Source` value of `WXData.ARCHIVE`. From each of these `WXDayData` objects, it retrieves their `WXStatRecord`, through `WXDayData`'s `getStatRecord()` method, which holds the relevant statistics for the day. These `WXStatRecord` objects are inserted into a `WXStatRecordSet` object, and this comprises the data of a `WXMonthData` object. When given a `WXData.ARCHIVE` value as `Source`, it merely opens the corresponding month data file in the database structure, and builds the `WXStatRecordSet` and its `WXStatRecords` from the data stored in the file.

After it has retrieved its data from either source, it finds the statistical values relevant to its `Fields`. It does this through the `getStat()` method of the `WXStatRecordSet` that holds its data. This method takes three parameters; the first is the `Field` index, then the statistical set of that `Field` from which the statistic will be calculated, and the statistic to calculate. Bearing in mind that `WXStatRecords` are three-dimensional data sets, where the first dimension is the `Day`, the second is the `Field`, and the third the `Statistic` data. For example, when invoking `getStat(TemperatureIndex, WXData.LOW, WXData.HIGH)` on a `WXStatRecordSet`, we are asking for the “Highest reading in the set of low temperatures”. The statistics are then stored in a `WXStatRecord` of its own.

`WXYearData` is the object that stores statistical information on a monthly basis for all the months in a year. To create one it requires a `WXLocation`, and `WXDay` objects, and a `Source` Value. Only the `Year` field of the `WXDay` object is relevant. When retrieving data from the source, it creates `WXMonthData` objects for every month in the year, using the `WXLocation`, the corresponding `WXDay` object and a `Source` value of

WXData.ARCHIVE. Its own data is then populated with the individual WXMonthData WXStatRecords, or the statistical data for every month. And then they are stored in a WXStatRecordSet object. When retrieving data from the archive, it opens the corresponding Year Data file in the database, and reads the data from there. Its statistical values are then retrieved from its WXRecordSet in the same way as the WXMonthData object.

In the cases of WXMonthData and WXYearData, after the data is retrieved, it does not go through the validating process as WXDayData does. This would be unnecessary, since all the data it acquires from WXDayData objects are already filtered, and statistic values never generate erroneous readings.

The WXCompiledDataClasses rely on the previous mentioned classes to gather their data. They never manipulate the database directly, but create the necessary WXData Objects they need. Though it is their responsibility to write their corresponding data files into the Compiles Directory. When creating one of them, a WXLocation is needed and two WXDay objects, the time range of data to compile. A Source value is not needed, since there are no archived CompiledData, they are always generated upon request by the user.

The WXCompiledReadingsData is created with a WXLocation, two WXDay objects, and a number array with the indexes of the Fields requested. It then iterates through all the days from the first WXDay object to the second, creating WXDayData objects with the provided WXLocation and the current WXDay. After they are all initialized, it calculates the relevant statistics for the Fields in the request. It must implement its own algorithms for finding these statistics. Given that there is no continuous series of data, rather it is segmented into several days, special functions need to be executed to retrieve a valid representative statistic. The High statistic for instance, needs to be found from the High reading statistic of all the WXDayData objects. The

Average statistic needs to be calculated from all the readings of all the `WXDayData` objects, as opposed to averaging the set of average statistic of each day.

`WXCompiledDayData` and `WXCompiledMonthData` operate in similar ways. `WXCompiledDayData` creates all the `WXMonthData` objects within the time range of the two `WXDay` objects. It then creates an empty `WXStatRecordSet` of its own, and populates it with all the `WXStatRecord` of every `WXMonthData` object, disregarding those days before and after the time range. This will give us a continuous series of data, and allow the use of the statistic functions in `WXStatRecordSet` to retrieve the statistical data. `WXCompiledMonthData` works the same way, using `WXYearData` objects to retrieve month statistical information.

They all implement the `saveData()` method. It requires a Unit System, and `WXData.MOST_PRECISE` is not acceptable. Whichever unit system is provided, the data will be saved in the Compiles Directory, using the corresponding file name, explained in the Data Structure chapter.

5.2 Web Display

All Web Display classes are embedded into Java Server Pages (JSP) files. These are web applications that allow the Java applications to generate WebPages dynamically, given certain parameters and user requests. Parameters will be taken from the user through HTML web form components (text boxes, check boxes, selections, etc.). These parameters will be passed between classes and JSP files as parameters in URLs. This approach allows the clients to continue making use of standard browser navigation functions such as “BACK”, “FORWARD”, “HISTORY”, as opposed to applications that request data by following a several-step process and in a pre-arranged order, where a fault in one of the steps forces the user to start over [1]. All of them use the data management classes to retrieve data from the database and generate the corresponding

Compiled Data files when necessary. All pages will always open links or data requests in new browser windows, except for navigational links. This is done because researchers often request new data, to compare with the data they are currently observing.

The Location Page is accessed through static links on the main webpage. It expects a parameter in the URL of `LocationCode=x`, where `x` is a number for identifying Locations in the database. This code is given to the `WXLocation` class which returns the corresponding `WXLocation` object. After the `WXLocation` object is created, this page then displays the relevant information about this Location, and the two data request forms. The first form, the Data Request Form, can open the `DayDataTable`, `MonthDataTable` or `YearDataTable`, depending on the option selected of `DataType`. The Compile Data Form, can lead the user to the `CompiledReadingsTable`, `CompiledDayDataTable`, or the `CompiledMonthDataTable`.

The `DayDataTable`, `MonthDataTable`, and `YearDataTable` all require the same parameters. A `LocationCode`, to indicate the Location from which data will be retrieved. a `Date=xxxxxxxx` in the eight digit date representation, for the date of data to be retrieved of the Location. This date is used as a day for `DayDataTable`, only the year and month fields for `MonthDataTable`, and the year field for `YearDataTable`. A `UnitSystem=x` is also expected, where `x` is one of the unit system values, if none is given a default system of British is used. Each of them then creates their corresponding Data object, which retrieves the data they need, and then use the data in that object to display it to the user. They also call on their corresponding Data object `saveData()` method and pass their current unit system. This creates the Data File in the Compiles Directory with the given unit system. The DataTable object then provides the user with the link to download this file.

`CompiledDataTable` pages require more parameters. They expect a `LocationCode`, and a `UnitSystem`. A `StartDate` and `EndDate`, each of which with an eight-digit date representation, marking the beginning and end of the time range to compile data. And a

comma separated array of indexes in `Fields=x,y,z`, to indicate the indexes of those Fields data is to be compiled for. With these parameters, they create their corresponding `CompiledData` objects, which provide them with the data to display on the webpage. They also then invoke the `CompiledData`'s `saveData()` method and save the data in the `Compiles Directory` for downloading to the user if he wishes to do so.

Finally, the `Graph` classes are called by links on the `DataTables`. As usual, they require a `LocationCode`, a `Date`, and a `UnitSystem`. They also need a `Field=x`, the index of the field for which a graph is to be generated. `MonthGraphTable`, and `YearDataTable`, may also expect a `Stat=x` index. This is the index of the statistical data upon which a graph will be generated. With these parameters, they obtain the `WXField` class corresponding to the index given in the `Field` parameter, and call its graph generating method, which can be the one they inherit from `WXField`, or one they implement themselves. These `GraphTables` then display the graph image to the user along with a navigational bar to scroll through dates, and times of data.

Chapter 6


Web Interface

This chapter presents the web interface as it is currently implemented on the website. It describes the elements for each page, and how they are designed to facilitate its usability.

In the article “*Websites That Satisfy Users*”, the authors describe websites as browsing or information seeking. [10] Information websites are those where the user already has a goal of information he wishes to obtain, and the website provides him with the mechanism for quickly obtaining his request, as in a query form, and performing the necessary operations for displaying his requested data. Browsing sites are those more interface-dependant where the user has not a specific data request but goes through an overview of generalized data and navigates to and from different but related information sites. Our goal is to provide a website that becomes a middle point between these two forms of data sites. The user will always approach the website with a specific request of data in mind, so the site must provide the mechanism for obtaining his request quickly, but it should also provide the means to navigate to other Locations, Dates, Fields, after the initial data request has been performed.

The interface was designed with the hopes of providing users with an optimum medium for researching weather data. An interface that makes it easy for researchers to find the data they need, with minimal time to learn how to use. It is also oriented at anticipating what information is relevant and important and providing it without requiring him to request it. This has been achieved relying on important user interface notions that make it easy to understand how the system will work and on recommendations by weather researchers on what information is most important and would be requesting most often.

Beginning with the LocationTable. First we provide the user with the general information about the Location. Its name, coordinates, altitude, institution that operates it, the time range of data. And as some important statistics, the highest and lowest recorded temperatures for that location in both unit systems, with links to the DayDataTable of their dates. A small map illustrates the Location in the island of Puerto Rico. Then we have form for viewing data for a particular date. It provides fields so the user can input a date, the unit system, and data type he wishes to view. When loading the page, the form always has a date set to the current data, or the last date in the Location's time range. This makes it easier for researchers to quickly access the latest weather information. The system always remembers the unit system the user has selected in the past and always initializes the page with that unit system set. If it is the user's first visit to the site, a default system of British is set. The default setting for Data Type is the Readings Data, as it is the setting for viewing the latest up to the minute data. Finally, there is the Compile Data Form. Here the user must select the Fields he wants to compile data for, the time range of data to compile, and the unit system he wishes the data in. All of these are blank when the page is loaded. The Data Collection type is initially set to the Readings Table, given that it's the most widely used setting for data compilation.



Mayaguez



Owner	University Of Puerto Rico	Location General Information
Latitude	18°13'	
Longitude	67°11'	
Altitude		
Data collection range	October 18, 2000 - June 4, 2003	
Lowest recorded temperature	59.5 °F (15.3°C) on April 27, 2001	
Highest recorded temperature	108.9 °F (42.7°C) on June 6, 2003	

Go to date

Date	Unit System	Data Collection Type
Day <input type="text" value="4"/>	<input checked="" type="radio"/> British <input type="radio"/> Metric	<input checked="" type="radio"/> 10 minutes interval for day <input type="radio"/> Daily summaries for month <input type="radio"/> Monthly summaries for year
Month <input type="text" value="6"/>		
Year <input type="text" value="2003"/>		

Compile data

Fields	Date range	Unit system	Data Collection Type
<input type="checkbox"/> Temperature	Day Month Year	<input checked="" type="radio"/> British <input type="radio"/> Metric	<input checked="" type="radio"/> 10 minutes interval for day <input type="radio"/> Daily summaries for month <input type="radio"/> Monthly summaries for year
<input type="checkbox"/> Dew Point	From: <input type="text"/> / <input type="text"/> / <input type="text"/>		
<input type="checkbox"/> Relative Humidity	To: <input type="text"/> / <input type="text"/> / <input type="text"/>		
<input type="checkbox"/> Heat Index			
<input type="checkbox"/> Pressure			
<input type="checkbox"/> Precipitation			
<input type="checkbox"/> Wind Speed			
<input type="checkbox"/> Wind Direction			
<input type="checkbox"/> Flux Density			

Figure 6-1. Location Page.

From the Data Request form, the user is taken to the DayDataTable, MonthData, or YearDataTable. All these tables have a common header, and their data tables. These headers display the current date of data on screen, and some important links relevant to that data. At the top, there is a form for opening a similar page of data for another location in the same data and current unit system. The user can select which Location to open with a selection box, with all the Location's installed in the database. Lower in the

table and properly iconized are relevant links to the data. The first is a link to open the LocationTable page of the current Location. Next there is a link to download the CSV file of the data on screen, which corresponds to a CSV file in the Compiles directory created by the DayDataTable page when the data was requested. Then a link to reload the data page in the opposite unit system. A link to open a data page of a higher order than the page currently on screen. This means if viewing a DayDataTable page, a link to the MonthDataTable page of the current month, in the MonthDataTable page, a link to the YearDataTable page. And finally, a link to the printable version of the data on screen.

The screenshot displays the CARIB web interface. At the top, there is a header with logos for 'STATION RADAR', 'FORECASTS', 'SATELLITE IMAGES', 'ATMOS', and 'CARIB'. The main title is 'CARIBBEAN ATMOSPHERIC RESEARCH CENTER UNIVERSITY OF PUERTO RICO AT MAYAGÜEZ'. Below the header, the location 'Mayaguez' is displayed, with a date input field set to 'June 7, 2003'. A 'Go!' button is present next to the date input. Below this, there are five links: 'Open location page', 'Download CSV file of this data', 'Switch to metric units', 'View summary for June 2003', and 'Printable version of this data'. At the bottom, there is a 'Navigation Bar' with links for '<< Previous June 6, 2003', 'Next >> June 8, 2003', and 'Today June 7, 2003'.

Figure 6-2. DataTable Header.

Below these important links, there is a navigational bar. This bar provides links to DayDataTable pages immediately before and after the current date. A DayDataTable provides links to days before and after the current day, a MonthDataTable links to the previous and following month, and a YearDataTable links to previous and following years. These links are not activated, if the date they lead to go outside the Location's time range of data. There's also a link to the actual current day, month or year if that Location is getting real-time data.

The data table then is unique for every page. Common elements between them are a graph bar, and a statistics bar. The graph bar provides all the possible graphs to be generated from every field. These include full readings graphs, of all readings in the data set, or statistical graphs. DayDataTable only provides readings graphs for the data of its day. MonthDataTable and YearDataTable also provide graphs of all readings within their time frames, and statistical graphs, such as the average temperatures for every month. The statistics bar displays statistical information of the data in the time frame for every field. For those statistics not supported by a particular field a blank space is left.

Next we have the Field headers and the data itself. It is ordered chronologically from the top down. For every field among the data table are marked the highest and lowest values of their data sets for those fields that are supported. These are marked by color schemes, the lowest reading in the set marked as blue, and the highest as red. This makes it easily identifiable to the user when a particularly important reading occurred.

Graph Selectors & Statistics Bar									
Graphs	Line graph	Line graph	Line graph	Line graph	Line graph	Bar graph Acumulative line graph	Line graph	Line graph Wind rose chart	Line graph
High	90.7	72.6	91.3	96.6	30.01	0.0	5.4	-	1615.0
Avg	80.39	69.1	70.32	82.27	29.96	-	2.08	70.1	444.6
Low	70.1	66.32	46.05	67.2	29.89	-	-	-	-
Total	-	-	-	-	-	0.0	-	-	-
Time	Temperature (°F)	Dew Point (°F)	Relative Humidity (%)	Heat Index (°F)	Pressure (inHg)	Precipitation (in)	Wind Speed (mph)	Wind Direction (°)	Flux Density ($\mu\text{mol/s/m}^2$)
12:10 A.M.	74.1	70.3	88.2	72.8	30.0	0.0	1.37	8.87	0.0
12:20 A.M.	74.0	70.2	87.3	72.7	30.0	0.0	0.97	226.2	0.0
12:30 A.M.	74.3	70.0	86.5	73.4	30.0	0.0	0.43	15.94	0.0
12:40 A.M.	74.1	70.2	88.2	72.9	30.0	0.0	0.92	0.0	0.0
12:50 A.M.	73.6	70.0	88.6	72.0	30.0	0.0	0.16	1.7	0.0
1:00 A.M.	73.4	69.83	88.5	71.6	30.0	0.0	0.5	36.88	0.0
1:10 A.M.	73.2	69.83	89.9	71.2	29.98	0.0	0.99	43.86	0.0
1:20 A.M.	73.1	70.0	89.9	70.6	29.98	0.0	0.99	98.8	0.0
1:30 A.M.	73.1	69.76	89.1	70.9	29.98	0.0	1.44	0.47	0.0
1:40 A.M.	72.8	69.71	90.4	70.3	29.98	0.0	1.15	77.1	0.0
1:50 A.M.	72.5	69.66	90.6	69.68	29.98	0.0	1.06	93.7	0.0
2:00 A.M.	72.6	69.62	89.4	70.0	29.98	0.0	1.55	96.6	0.0
2:10 A.M.	72.7	69.36	89.8	70.4	29.97	0.0	1.22	24.43	0.0
2:20 A.M.	72.1	69.2	90.0	69.18	29.97	0.0	1.12	56.4	0.0

Figure 6-3. DayData data table.

Graphs		10 minute intervals			10 minute intervals			10 minute intervals			10 minute intervals			10 minute intervals			10 minute intervals			10 min intervals
		Daily lows			Daily lows			Daily lows			Daily lows			Daily lows			Daily highs			Daily averages
		Daily averages			Daily averages			Daily averages			Daily averages			Daily averages			Daily totals			Daily highs
		Daily highs			Daily highs			Daily highs			Daily highs			Daily highs						Daily highs
High	70.1	80.77	103.3	67.34	69.57	83.8	64.54	88.86	96.2	71.7	82.78	92.7	30.0	30.05	30.1	0.3	1.4	2.79		
Avg	66.45	75.87	87.53	62.53	67.1	71.76	49.43	76.16	93.04	63.17	75.97	89.74	29.94	29.99	30.04	-	-	1.73		
Low	63.25	72.6	82.7	57.52	63.68	67.61	32.93	65.39	77.8	60.8	70.2	87.0	29.86	29.9	29.96	-	-	-		
Total	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2.8	-		
Day	% of data loss	Temperature (°F)			Dew Point (°F)			Relative Humidity (%)			Heat Index (°F)			Pressure (inHg)			Precipitation (in)			Wind Speed (mph)
		Low	Avg	High	Low	Avg	High	Low	Avg	High	Low	Avg	High	Low	Avg	High	High	Total	Avg	
Saturday 1	0	66.73	76.04	88.1	64.26	67.37	70.5	46.85	76.5	92.6	63.03	75.93	91.1	29.86	29.9	29.96	0.0	0.0	1.55	
Sunday 2	0	67.19	74.62	85.3	64.63	68.15	71.5	52.49	81.64	95.6	63.62	73.69	88.4	29.89	29.95	30.02	0.01	0.03	1.51	
Monday 3	0	68.66	73.36	84.9	67.34	69.57	71.9	56.57	88.86	96.1	62.69	70.2	89.2	29.93	29.98	30.03	0.3	1.4	1.13	
Tuesday 4	0	68.99	74.92	82.7	67.1	68.56	71.5	64.54	81.23	96.2	62.64	74.47	87.0	29.94	29.97	30.03	0.01	0.01	2.03	
Wednesday 5	0	67.24	73.31	85.6	65.27	68.71	74.1	55.28	86.67	95.9	62.54	71.07	88.9	29.93	29.97	30.02	0.27	0.73	1.17	
Thursday 6	0	66.85	75.15	85.9	65.2	68.61	72.7	55.51	81.73	95.7	61.7	74.03	90.2	29.94	29.98	30.04	0.03	0.04	1.67	
Friday 7	0	66.35	74.12	85.1	63.96	67.34	74.1	57.07	80.8	94.8	62.24	73.37	89.4	29.94	29.98	30.03	0.05	0.1	1.49	
Saturday 8	0	64.3	72.6	85.0	62.59	67.62	72.9	56.74	85.46	95.6	60.8	71.05	88.9	29.98	30.02	30.08	0.17	0.43	1.28	
Sunday 9	0	66.89	75.33	85.5	65.55	68.6	71.6	56.31	80.95	95.9	61.2	74.55	90.3	29.97	30.01	30.06	0.0	0.0	1.6	
Monday 10	13	67.87	77.3	89.1	57.52	65.61	68.43	32.93	70.18	93.8	63.43	76.81	90.2	29.94	29.99	30.05	0.0	0.0	1.94	
Tuesday 11	100	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Wednesday 12	52	70.1	78.72	87.3	61.15	65.82	70.7	49.8	65.39	77.8	71.7	81.51	92.0	29.95	29.99	30.04	0.0	0.0	2.79	
Thursday 13	10	65.97	76.85	87.4	63.21	68.55	72.4	54.42	76.49	91.8	63.46	78.53	92.2	29.95	30.0	30.05	0.0	0.0	1.71	
Friday 14	0	67.8	76.41	85.1	60.5	67.82	71.0	45.10	77.11	95.7	61.95	75.04	97.2	29.96	29.9	29.95	0.0	0.0	1.64	

Figure 6-4. MonthData data table.

On Figure 6-4 and 6-5, we can see links are provided at the left hand column to open the corresponding DayDataTable or MonthDataTable of the particular day or month. An extra column is added in the MonthDataTable for Data Loss. Data Loss is the percentage of data marked as invalid in the WXDayData object for that particular Location relative to the number of total readings a day should have for that particular Location. This is important for a client that is viewing a MonthDataTable and may encounter a rare statistic for a low or high reading in a particular field. With the Data Loss indicator, he can ascertain whether all the data for that day was collected, which tells him/her whether that statistic is accurate or not. For example, a “low temperature reading of the day” that is unreasonably higher than any other day in the month; if the user notices that there is a 50% Data Loss for that day, it tells him/her that the weather station only recorded temperature during the daylight hours, and the hours when the coldest temperatures occur were discarded, and that lowest reading statistic was taken from the reduced set of hotter temperatures. This explains to the user the strange statistic, and avoids necessity for any further inquiry.

Graphs	10 minute intervals			10 minute intervals			10 minute intervals			10 minute intervals			10 minute intervals			10 minute intervals		10 minute intervals		m int Mc ave Y r
	Monthly lows			Monthly lows			Monthly lows			Monthly lows			Monthly lows			Monthly highs		Monthly averages		
	Monthly averages			Monthly averages			Monthly averages			Monthly averages			Monthly averages			Monthly totals		Monthly highs		
	Monthly highs			Monthly highs			Monthly highs			Monthly highs			Monthly highs							
High	67.7	77.66	108.9	64.94	70.32	99.0	49.47	80.75	96.8	62.37	78.3	99.4	29.9	29.99	32.15	0.66	8.7	1.98	157.01	
Avg	64.95	76.61	95.63	60.61	68.73	84.0	41.15	78.38	96.27	60.9	76.8	96.37	29.55	29.96	30.41	-	-	1.66	31.5	4
Low	63.25	75.13	89.0	55.63	66.88	74.4	30.88	73.91	95.8	59.82	74.52	92.7	28.11	29.93	30.02	-	-	-	-	
Total	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	22.79	-	-	
Month	Temperature (°F)			Dew Point (°F)			Relative Humidity (%)			Heat Index (°F)			Pressure (inHg)			Precipitation (in)		Wind Speed (mph)		Ge V Dir
	Low	Avg	High	Low	Avg	High	Low	Avg	High	Low	Avg	High	Low	Avg	High	High	Total	Avg	High	
January	64.46	75.13	89.4	61.2	67.86	94.5	41.9	79.85	96.8	59.82	74.52	98.3	28.11	29.98	32.15	0.17	1.2	1.51	157.01	3
February	63.25	75.87	103.3	57.52	67.1	83.8	32.93	76.16	96.2	60.8	75.97	92.7	29.86	29.99	30.1	0.3	2.8	1.73	6.28	3
March	63.48	76.62	91.9	55.63	66.88	74.4	30.88	73.91	96.0	61.67	77.12	94.7	29.81	29.94	30.06	0.12	1.12	1.98	7.81	4
April	65.41	77.22	108.9	61.55	69.97	99.0	45.68	79.81	96.5	60.45	77.53	98.3	29.79	29.93	30.07	0.66	8.7	1.69	6.28	5
May	65.41	77.66	91.3	62.81	70.32	77.1	46.05	79.78	96.3	60.31	78.3	99.4	29.82	29.97	30.08	0.32	6.51	1.55	6.62	4
June	67.7	77.13	89.0	64.94	70.23	75.2	49.47	80.75	95.8	62.37	77.37	94.8	29.9	29.96	30.02	0.24	2.47	1.49	5.0	4
July	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
August	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
September	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
October	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
November	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
December	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

Figure 6-5. YearData data table.

The DayDataTable displays a much larger set of records than the MonthData or YearData tables. For a Location with a 10-minute readings interval, the table amounts to 144 records. This causes interface problems for the user whenever the data greatly exceeds the screen size of the computer [1]. This problem has already been addressed by the W3C, and as solution they have set forth a standard of separating data in tables by header, body and footer. It should allow the user to scroll through the data portion of the table, while the rest of the page and the table header remain static. Unfortunately, mainstream browsers have not yet implemented this. To correct this, the DayDataTable page, repeats the Field headers bar right after the 12:00 N. record, and at the foot of the table.

The PrintTable pages provide the client with a display of the data that is optimal for printing. Only minimal and essential data is displayed in a colorless fashion. They display the same data as their DataTable counterparts, and their corresponding statistics. The statistics among the data table are marked with (H) and (L) for High and Low.

Mayaguez January 31, 2003									
High	87.4	71.9	93.7	90.9	29.98	0.0	4.37	-	1600.0
Avg	74.48	67.68	80.75	73.78	29.92	-	1.28	43.27	256.53
Low	66.2	64.1	51.0	62.28	29.88	-	-	-	-
Total	-	-	-	-	-	0.0	-	-	-
Time	Temperature (°F)	Dew Point (°F)	Relative Humidity (%)	Heat Index (°F)	Pressure (inHg)	Precipitation (in)	Wind Speed (mph)	Wind Direction (°)	Flux Density (μmol/s/m ²)
12:10 A.M.	68.85	66.99	93.4	63.85	29.96	0.0	1.17	23.67	0.0
12:20 A.M.	68.8	66.83	93.4	64.0	29.96	0.0	0.92	75.8	0.0
12:30 A.M.	68.73	66.75	93.3	63.97	29.96	0.0	0.72	48.76	0.0
12:40 A.M.	68.6	66.62	93.4	63.85	29.96	0.0	0.5	28.2	0.0
12:50 A.M.	68.39	66.48	(H) 93.7	63.52	29.96	0.0	0.9	53.29	0.0
1:00 A.M.	68.28	66.42	(H) 93.7	63.35	29.96	0.0	1.08	19.43	0.0
1:10 A.M.	68.16	66.25	93.6	63.33	29.94	0.0	1.17	105.5	0.0
1:20 A.M.	68.33	66.31	92.7	63.68	29.94	0.0	1.08	29.99	0.0
1:30 A.M.	68.24	66.04	92.8	63.94	29.94	0.0	0.9	85.6	0.0
1:40 A.M.	68.0	65.93	92.8	63.52	29.94	0.0	1.26	62.06	0.0
1:50 A.M.	68.1	65.86	92.2	63.89	29.94	0.0	0.74	35.28	0.0
2:00 A.M.	68.07	65.7	91.9	64.11	29.94	0.0	0.72	56.88	0.0
2:10 A.M.	68.03	65.56	91.7	64.25	29.93	0.0	1.37	54.23	0.0
2:20 A.M.	67.94	65.45	91.8	64.23	29.93	0.0	0.72	31.69	0.0
2:30 A.M.	67.78	65.38	92.2	63.95	29.93	0.0	1.4	28.96	0.0
2:40 A.M.	67.6	65.4	92.5	63.46	29.93	0.0	1.15	40.61	0.0

Figure 6-6. DayDataPrintTable.

The CompiledDataTables share the same header as the DataTables, with the exception of navigation bars. They all provide the same statistics bars, for the data they compiled. No graphs for CompiledData are implemented, so there are no graph bars. The CompiledDaySummaries and CompiledMonthSummaries each provide links to their corresponding DayDataTable and MonthDataTable pages. Each of these pages also provides links to their corresponding printable versions.

High		91.2	-	-
Avg		78.89	49.02	-
Low		67.7	-	-
Total		-	-	-
Date	Time	Temperature (°F)		Wind Direction (°)
May 27, 2003	12:10 A.M.	74.1		84.1
May 27, 2003	12:20 A.M.	74.2		50.18
May 27, 2003	12:30 A.M.	74.4		76.5
May 27, 2003	12:40 A.M.	74.5		41.69
May 27, 2003	12:50 A.M.	74.3		38.58
May 27, 2003	1:00 A.M.	73.9		33.96
May 27, 2003	1:10 A.M.	73.5		18.68
May 27, 2003	1:20 A.M.	73.2		29.24
May 27, 2003	1:30 A.M.	72.8		9.62
May 27, 2003	1:40 A.M.	72.7		43.67
May 27, 2003	1:50 A.M.	72.8		35.56
May 27, 2003	2:00 A.M.	72.7		59.89
May 27, 2003	2:10 A.M.	72.3		92.2
May 27, 2003	2:20 A.M.	72.5		60.27
May 27, 2003	2:30 A.M.	72.7		53.29
May 27, 2003	2:40 A.M.	72.5		41.03
May 27, 2003	2:50 A.M.	72.1		39.05
May 27, 2003	3:00 A.M.	71.9		172.5
May 27, 2003	3:10 A.M.	72.0		25.75

(1)

High		73.3	81.17	108.9	-
Avg		69.46	77.35	88.2	49.91
Low		65.41	71.4	77.5	-
Total		-	-	-	-
Date	Temperature (°F)			General Wind Direction (°)	
	Low	Avg	High		
March 27, 2003	71.4	76.13	85.5	57.46	
March 28, 2003	68.8	75.63	87.4	75.73	
March 29, 2003	68.62	75.15	85.2	38.31	
March 30, 2003	70.0	77.66	86.6	17.3	
March 31, 2003	66.81	76.95	87.6	10.39	
April 1, 2003	69.58	74.22	85.7	62.11	
April 2, 2003	69.76	76.13	86.7	63.4	
April 3, 2003	69.43	74.84	85.5	57.3	
April 4, 2003	67.93	76.78	86.9	29.68	
April 5, 2003	70.5	78.04	89.4	62.23	
April 6, 2003	73.3	81.17	108.9	17.85	
April 7, 2003	72.0	80.24	96.9	75.9	
April 8, 2003	73.2	80.46	96.9	55.39	
April 9, 2003	72.3	79.09	86.4	67.81	
April 10, 2003	68.87	77.68	89.0	66.84	
April 11, 2003	70.1	76.5	87.0	70.72	
April 12, 2003	70.4	78.37	88.1	78.23	
April 13, 2003	69.43	76.05	89.1	62.96	

(2)

High		69.54	79.45	108.9	-
Avg		65.51	76.94	92.52	46.81
Low		60.99	74.25	87.5	-
Total		-	-	-	-
Month	Temperature (°F)			General Wind Direction (°)	
	Low	Avg	High		
January 2002	62.96	76.06	88.9	48.06	
February 2002	60.99	74.25	89.2	48.91	
March 2002	63.32	75.42	89.9	49.39	
April 2002	63.41	75.09	87.5	46.16	
May 2002	66.25	77.65	91.4	51.81	
June 2002	67.97	79.31	93.4	49.77	
July 2002	67.51	79.31	93.8	42.03	
August 2002	69.54	78.6	92.9	46.25	
September 2002	68.99	79.45	92.6	53.42	
October 2002	66.97	77.68	91.5	50.32	
November 2002	67.43	76.93	90.5	44.85	
December 2002	64.09	75.51	90.0	47.54	
January 2003	64.46	75.13	89.4	39.68	
February 2003	63.25	75.87	103.3	37.59	
March 2003	63.48	76.62	91.9	44.66	
April 2003	65.41	77.22	108.9	58.59	
May 2003	65.41	77.66	91.3	43.66	
June 2003	67.7	77.23	89.0	39.85	

(3)

Figure 6-7. CompiledDataTables. (1) CompiledReadingsTable, (2) CompiledDaySummariesTable, (3) CompiledMonthSummariesTable.

Graph windows are opened through the links at any of the graph bars from DataTables. GraphTables can be of DayGraphTables, MonthGraphTables, or YearGraphTables, each of which displaying data for their corresponding time frames, fields, and/or statistics. They provide the client with a form for selecting different dates, unit systems, Locations, and fields for which to regenerate a graph. An Options form is also available to modify the graph at the user's convenience, such as the image size, the range of the y-axis, and graph gridlines. Every GraphTable page also provides a link to

open the corresponding DataTable of the current date and Location. These graphs are displayed as regular web images, so the user can save them through the web browser's save image option.

The graph type can be modified through their corresponding WXField class. This allows future application operators, to provide graph types which best suit the type of data to be displayed. This should be done by taking into consideration the structure and type of data to be displayed; the intended use of the graph; and the information needed by the client from the data [2].

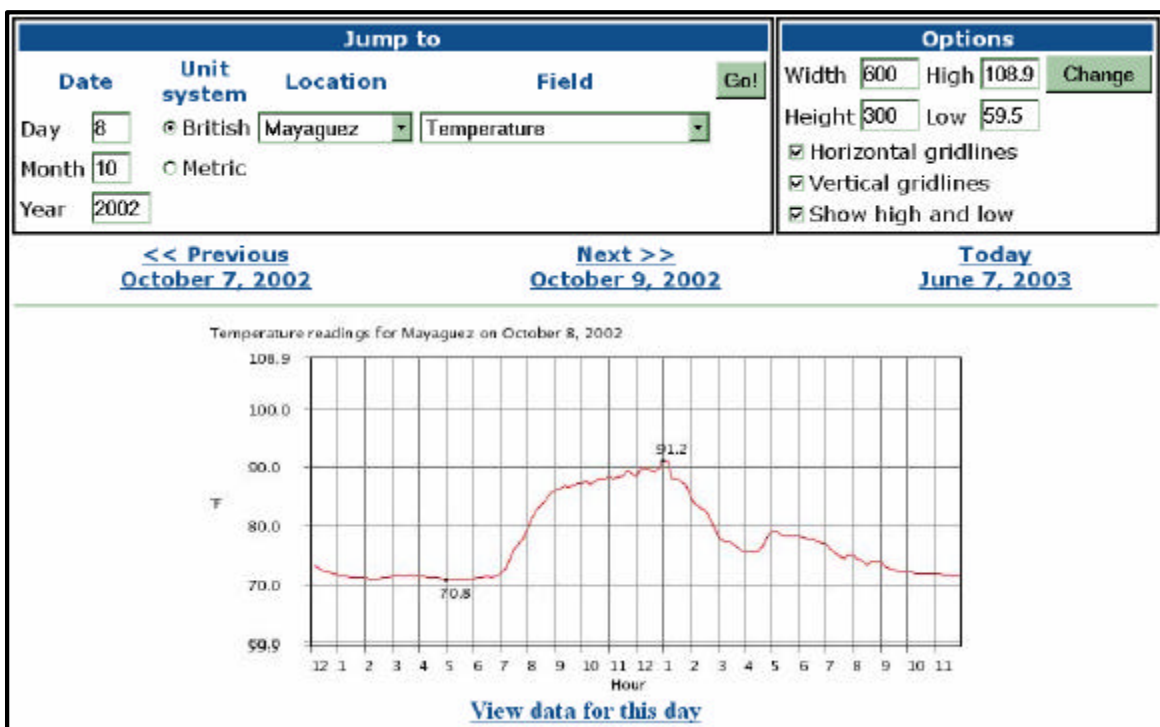


Figure 6-8. Graph table.

6.1 Performance Tests

The main test of performance for our software is the load time of a requested data page by a user. Factors that determine the load time are the speed of the server to perform the software operations of data retrieval and creation of WebPages and graph images, the

speed of the user's Internet connection to transfer the data into his computer, and the loading of the webpage on his computer browser. Usually this last factor is negligible, but when unusually large WebPages, where there are many text elements to display and arrange, a browser can delay several seconds displaying the whole set of data.

The following tests were performed requesting data for a Location with a 10-minute readings interval and on an Internet connection of a 56K modem. The results are an average of 10 tries for each test. All these tests go through the following steps:

1. Retrieval of data on the server.
2. Calculations of relevant statistics on the retrieved data set.
3. Creation of webpage or graph image.
4. Download of webpage or graph image into user's computer.
5. Display of webpage or graph image on user's browser.

Tests results:

- Day Data Page (144 records) : 3.2 sec
- Month Data Page (31 records) : 2.4 sec
- Year Data Page (12 records) : 1.7 sec
- Compiled Readings Page (10 days, 1440 records) : 6.7 sec
- Compiled Day Summaries Page (50 days, 50 records) : 3.5 sec
- Compiled Month Summaries Page (50 months, 50 records) : 3.8 sec
- Day Graph (144 readings) : 3.8 sec
- Month Stat Graph (12 readings) : 3.1 sec
- Month Full Graph (4464 readings) : 5.6 sec
- Year Stat Graph (31 readings) : 3.6 sec
- Year Full Graph (52560 readings) : 7.3 sec

Chapter 7

Conclusions

The developed application has been implemented and successfully executed over the past year in the UPR's Climate Department server. Currently it has acquired archived data from nine Locations' throughout Puerto Rico and receives real-time data from our local weather station, which is displayed in real time at our website.

The database is available on the web server. Using the file structure outlined in Chapter 2, third party institutions have access to the database. This database is updated every 10 minutes, so users can retrieve the latest data and incorporate it into their databases and applications, in an automated fashion.

Through the web interface, users have at their screens an easy to use and understand interface for retrieving data they need. This interface provides them with the relevant information about the location, and the tools to select and request or compile the particular type of data they are looking for. The data pages serve to display the requested data in a design that also points out the relevant statistics most users would also be interested in. Through the use of representative icons he can also quickly retrieve other information regarding the data he is currently viewing. These data can also be downloaded and used at their convenience.

With the data compilation tool, users have a mechanism for retrieving the specific data they need. They can select the time range of data desired along with the Fields for a particular Location. These data is then provided in a continuous data table, which can be downloaded and manipulated for their interests. This saves them the task of retrieving all the individual data files and the compiling them himself together.

Users also have the option to view and navigate through the data in visual form with charts and graphs. This enhances the study of data by providing a quick view of the states of the particular weather Field. These charts and graphs can be downloaded as regular web images for other uses by researchers.

7.1 Future Work

Having implemented this system and made it available to the public on the Internet, the tasks left to are to continue expanding the software to include more locations and more data, and to improve on the features for helping researchers study weather data. Currently our locations are limited to Puerto Rico, yet the system can be adapted to include other locations throughout the world if other organizations agree to share their data.

One feature desired by users is a mechanism for Data Comparison. Users at the site should have the ability to group data tables from different locations and/or dates into one table or data file. This would greatly allow researchers to study weather phenomena and its effects on different locations. This data comparison feature should come with all its corresponding interfaces, data tables, graphs, and data files.

A mapped display of statistical data. It would be of importance to display some statistical value for fields for several locations laid over a map of a region. It provides users with a better view of weather behavior and its effects over a particular geographical area. This could be done making use of the Location's geographical coordinates already stored as part of its information file.

Bibliography

1. Burton, C. and Johnston, L. 1998. *Will World Wide Web User Interfaces be Usable?* Computer Human Interaction Conference. Proceedings 1998. 1998 Australasian , 30 Nov.-4 Dec. 1998 p: 39 –44.
2. Doumont, J. and L. Vandenbroeck. March 2002. *Choosing The Right Graph.* Professional Communication, IEEE Transactions on. 45(1): 1–6.
3. Eckel, Bruce. 2002. *Thinking in Java 3rd edition.* Prentice Hall PTR, USA.
4. Harold, Eliote Rusty. 1999. *XML Bible.* IDG Books, USA.
5. Hong Su, Kramer, D., Li Chen, Claypool, K. and Rundensteiner, E.A. 1-2 April 2001. *XEM: Managing the Evolution of XML Documents.* Research Issues in Data Engineering. 2001 Proceedings. Eleventh International Workshop. p: 103–110.
6. Jingyu Hou, Yanchun Zhang and Kambayashi, Y. 2001. *Object Oriented Representation for XML Data.* Cooperative Database Systems for Advanced Applications. CODAS 2001. The Proceedings of the Third International Symposium. p: 40 –49.
7. Jun Wen, Rui Zhang and Xianliang Lu. 4-5 Nov 2002. *The Design of Efficient XML Document Model.* Machine Learning and Cybernetics. Proceedings. 2002 International Conference. vol 2. p: 1102 -1106.
8. Mak, E.H.C., Chan, S.S.M. and Qing Li. 6-8 Nov 2002. *XML vs. Object-Oriented XML: Motivations, Applications, and Performance Evaluations.* Cyber Worlds, 2002. Proceedings. First International Symposium. p: 371 –377.
9. Mueller, A., Mundt, T. and Lindner, W. 2001. *Using XML to Semi-automatically Derive User Interfaces.* User Interfaces to Data Intensive Systems, 2001. UIDIS 2001. Proceedings. Second International Workshop. p: 91 –95.
10. Ping Zhang, Small, R.V., von Dran, G.M. and Barcellos, S. 1999. *Websites That Satisfy Users: A Theoretical Framework for Web User Interface Design and Evaluation.* System Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference. Volume: Track2, p: 8 pp.

11. Pokorny, Jaroslav. 2000. *XML Functionality*. Database Engineering and Applications Symposium, 2000 International. p: 266 – 274.
12. Renner, A. 2-6 April 2001. *XML Data and Object Databases: The Perfect Couple?* Data Engineering, 2001. Proceedings. 17th International Conference. p: 143–148.
13. Roy, J. and Ramanujan, A. March-April 2001. *XML Schema Language: Taking XML to the Next Level*. IT Professional , 3(2): 37 –40.
14. Sangho Ha and Kyoungrea Kim. 12-16 June 2001. *Mapping XML Documents to the Object-Relational Form*. Industrial Electronics, 2001. Proceedings. ISIE 2001. IEEE International Symposium. Vol: 3. p: 1757 -1761.
15. Soreide, N.N., Sun, C.L., Kilonsky, B.J., Denbo, D.W., Zhu, W.H. and Osborne, J.R. 5-8 Nov 2001. *A Climate Data Portal*. OCEANS, 2001. MTS/IEEE Conference and Exhibition. Vol: 4. p: 2315 -2317.
16. Tornqvist, A., Nelson, C. and Johnson, M. 16-18 June 1999. *XML And Objects-the Future of The e-forms on the Web*. Enabling Technologies: Infrastructure for Collaborative Enterprises, 1999. (WET ICE '99) Proceedings. IEEE 8th International Workshops. p: 303 -308.

Appendix A

Programmer's Manual

A.1 Setting up the application

The software package comes in a bundle of java classes. To run the application the installation of the Java Runtime Environment is required. It is available for download a Sun's website (java.sun.com). To modify the software or add Locations and Fields, it is necessary to download the Java Standard Development Kit (JDK) also available at Sun's website.

After installing both of these packages, and properly configured them to allow the execution of java applications, the software can be copied onto the host computer. It can be copied into any directory of the host computer, if installing on a web server, it should be installed into a directory accessible to users from the internet and given privileges to read and write into the records directory.

Next, the main operating class of the application needs to be written. This class is called **InteractiveWX**. In it are declared the Records and Compiles directories. This class can also be used to operate the automated functions of the application and executed from there. Access to these directories should be placed under get methods named below. Sample Code A-1 gives us an example of how it should look:

```
class InteractiveWX {  
  
    public static file getRecordsDirectory() {  
        return new File( "C:\\\\Records\\" );  
    }  
  
    public static file getCompilesDirectory() {  
        return new File( "C:\\\\Records\\\\Compiles" );  
    }  
}
```

Sample Code A-1. Declaration of Records and Compiles directories in InteractiveWX class

The application is now ready to be used. At this moment however there are no Locations set up in the system. The next section covers how to add these Locations into the database. There is also several Fields provided with the software that most climate data support, if there is data for fields that are not supported they should be added as well.

A.2 Adding a Location

A Location needs to have its own corresponding Location class. We should first give it a representative name for it. The class must also extend the WXLocation class and implement the methods declared by it in order to be recognized by the application as a valid Location. Working on the Location at Mayaguez we could create a class as described in Sample Code A-2.

```
class Mayaguez extends WXLocation {  
  
}
```

Sample Code A-2. Location class declaration.

A constructor method should then be written for the class. This constructor should assign a keyword for the Location, making sure that it is a unique keyword for all the implemented Locations. The indexes for the supported classes must be given a unique value different than -1. A value of -1 for a Field index indicates the Location doesn't support that Field. Each field index needs to be given a distinct value in consecutive orders, since these indexes will be used to identify data fields from data sets. These field indexes are declared in the WXLocation class and inherited into our current Location class when extending it. The constructor for our Mayaguez Location is declared in Sample Code A-3.

```

class Mayaguez extends WXLocation {

    public Mayaguez() {
        KeyWord = "mayaguez"
        PressureIndex      = 0;
        RelativeHumidityIndex = 1;
        DewPointIndex      = 2;
        WindSpeedIndex      = 3;
    }
}

```

Sample Code A-3. Location class constructor declaration.

Next we need to write the Location's time interval function. This function returns the time interval of readings for that Location in minutes. Given that a Location can change intervals of readings in the course of its time range, it expects a WXDay object of the data when the readings interval is sought. If it has always maintained a constant readings interval, the WXDay object can be ignored.

```

class Mayaguez extends WXLocation {

    public Mayaguez() {
        ...
    }

    public getReadingsInterval( WXDay Day ) {
        return 10;
    }
}

```

Sample Code A-4. Location class constructor declaration.

Then the most important function in the Location class is the method for retrieving data from its source. This method takes as parameter the WXDay object of the day for data requested and should return the data in a WXRecordSet object. It will access the weather station or the third party database and retrieve the data in the format they store it. It should then convert the data into the accepted format of the Data Management classes and properly organize it into its corresponding WXRecord objects and return the WXRecordSet with all the records.

```

class Mayaguez extends WXLocation {

    public Mayaguez() {
        ...
    }

    public int getReadingsInterval( WXDay Day ) {
        ...
    }

    public WXRecordSet getReadingsFromSource( WXDay Day ) {
        ...
    }

}

```

Sample Code A-5. Declaration of Location's function for retrieving data from the source.

This Location class is now ready to be included into the application. It should now be registered into the WXLocation class as one of the Location's available in the database and assigned a LocationCode value. These Location codes are declared in the WXLocation class, in a segment that should look like Sample Code A-6.

```

class WXLocation {

    public static final int MARICAO    = 0,
                        AGUADILLA = 1;

    ...

}

```

Sample Code A-6. Location codes in WXLocation class.

We then add another entry for the Location we've just created.

```

class WXLocation {

    public static final int MARICAO    = 0,
                        AGUADILLA = 1,
                        MAYAGUEZ  = 2;

    ...

}

```

Sample Code A-7. Entry of new class into the Location codes list.

We must then associate that Location Code to the Location class we've just created. This is done in the Location's `getLocation()` function. This function takes as parameter a Location Code and should return the corresponding Location object.

```
class WXLocation {

    public static final int MARICAO    = 0,
                          AGUADILLA = 1,
                          MAYAGUEZ   = 2;

    public static WXLocation getLocation( int LocationCode ) {
        switch ( LocationCode ) {
            case MARICAO    : return new Maricao();
            case AGUADILLA : return new Aguadilla();
        }
        return null;
    }
}
```

Sample Code A-8. `getLocation()` method in `WXLocation` class.

Into this method, we then add the new Location:

```
class WXLocation {

    public static final int MARICAO    = 0,
                          AGUADILLA = 1,
                          MAYAGUEZ   = 2;

    public static WXLocation getLocation( int LocationCode ) {
        switch ( LocationCode ) {
            case MARICAO    : return new Maricao();
            case AGUADILLA : return new Aguadilla();
            case MAYAGUEZ   : return new Mayaguez();
        }
        return null;
    }
}
```

Sample Code A-9. Entry of new Location into `getLocation()` method.

The Location is now ready to collect data. It is recommended that the Location's `saveLocationInfo()` method is called to create the Location's data directory within the Record directory (using its keyword) and its XML data file. The XML data file will have

all the corresponding tags for the location data, but they will all be empty. The user can now input into them the Location's information, such as its name, the coordinates, organization that operates it, etc.

Adding a Field

To implement a field that is not currently supported by the system it needs its own class, which extends `WXField` and implements the functions declared into it. We first create our Field class and make it an extension of `WXField`, as shown in Sample Code A-10.

```
class Temperature extends WXField {  
  
}
```

Sample Code A-10. Field class declaration.

Then we declare the overall relevant statistic of the Location for this field. This is the data which is to be saved into the Location's XML Data file.

```
class Temperature extends WXField {  
  
    private double HighestReading,  
                  LowestReading;  
  
    private WXDay HighestReadingDay,  
                  LowestReadingDay;  
  
}
```

Sample Code A-11. Field class relevant statistics declaration.

These statistics need to be written to and read from the Location's XML Data file. This is done through the Fields `writeInfo()` and `readInfo()` functions. Each of which take a `XMLDocumentHandler` object, and should use its interface to read and write its data into the file.

```

class Temperature extends WXField {

    ...

    public void writeInfo( XMLDocumentHandler ) {
        ...
    }

    public void readInfo( XMLDocumentHandler ) {
        ...
    }

}

```

Sample Code A-12. XML information handler methods.

The Field's title or name is retrieved through a `getTitle()` function. This is the title that will be used at all tables when naming a column of data.

```

class Temperature extends WXField {

    ...

    public String getTitle() {
        return "Temperature";
    }

}

```

Sample Code A-13. Field's Title method.

After the Title, we also need the units in which the data for this field is measured in. The Data Management classes retrieve this through the Field's `getUnit()` method, which takes as parameter the unit system desired, as one of the values declared in `WXData`. An entry for `WXData.MOST_PRECISE` should also have its entry in this method, it should return one of the Metric or British units, depending on which system is most precise for this Field.

```

class Temperature extends WXField {

    ...

    public String getUnit( int UnitSystem ) {
        switch ( UnitSystem ) {
            case WXData.METRIC      : return "°C";
            case WXData.BRITISH     : return "°F";
            case WXData.MOST_PRECISE : return getUnit( WXData.BRITISH );
        }
        return null;
    }
}

```

Sample Code A-14. Field's unit method.

Aside from providing these units, this Field must also be able to convert a reading from one of these unit systems to another. This must be done in the Field's `convert()` method, which takes as parameters a reading and a unit system to convert the reading to. The Data Management classes always work with readings in their most precise unit systems, so this method should only convert readings if a unit system is provided that is not the Field's most precise unit system. Since the given data will already be in that type of system.

```

class Temperature extends WXField {

    ...

    public double convert( double Reading, int UnitSystem ) {
        if ( UnitSystem == WXData.METRIC )
            return ( Reading - 32 ) * 5/9;
        else
            return Reading;
    }
}

```

Sample Code A-15. Field's convert method.

As we can see in the example, given that this Field's most precise unit system is `WXData.BRITISH`, it will always receive its Reading value in the British unit system, so

if a request is made to convert it into a British unit system, no conversion is necessary. This explains why a mathematical operation need only be performed in the case that the unit system to convert the reading to is of WXData.METRIC.

Finally, this Field needs to identify to the Data Management classes, for which statistics it supports. This is done through a `hasStat()` method. It requires as parameter one of the statistical indexes, declared in the WXData class, and returns true or false depending on whether this Field supports that statistic.

```
class Temperature extends WXField {

    ...

    public boolean hasStat( int StatIndex ) {
        switch ( StatIndex ) {
            case WXData.LOW :
            case WXData.AVG :
            case WXData.HIGH :
                return true;
        }
        return false;
    }
}
```

Sample Code A-16. Field's hasStat method.

In this case, the statistics implemented by temperature are low, average and high. These are the statistics that have their entry in the hasStat method, for which will return true if provided any of their statistical indexes. A StatIndex of WXData.TOTAL, would not have any entry in the hasStat method and would return false in this case.

This Field is now ready to be used by the application. It must first be registered as a possible field for any Location in the WXLocation class. Here is where all the field indexes are declared in a segment of code like the one illustrated in Sample Code 3-7.

```
class WXLocation {  
  
    public int DewPointIndex      = -1,  
           PressureIndex        = -1,  
           RelativeHumidityIndex = -1,  
           WindSpeedIndex       = -1,  
           WindDirectionIndex   = -1,  
           TemperatureIndex     = -1;  
  
}
```

Sample Code A-17. Declaration of Field indexes in WXLocation class.

It must also be registered into the WXLocation's `initializeFields()` method. Here, the Location iterates through all the Field indexes to find those supported by the Location. It then builds the Fields array for the Location and creates the corresponding field object to populate that array.

```

class WXLocation {

    ...

    private void initializeFields() {
        int FieldsCount = 0;
        if ( DewPointIndex      != -1 ) FieldsCount += 1;
        if ( PressureIndex      != -1 ) FieldsCount += 1;
        if ( RelativeHumidityIndex != -1 ) FieldsCount += 1;
        if ( WindSpeedIndex      != -1 ) FieldsCount += 1;
        if ( WindDirectionIndex  != -1 ) FieldsCount += 1;
        if ( TemperatureIndex      != -1 ) FieldsCount += 1;

        Fields = new WXField[ FieldsCount ];

        if ( DewPointIndex      != -1 ) Fields[ DewPointIndex ] = new DewPoint();
        if ( PressureIndex      != -1 ) Fields[ PressureIndex ] = new Pressure();
        if ( RelativeHumidityIndex != -1 ) Fields[ RelativeHumidityIndex ] = new
RelativeHumidity();
        if ( WindSpeedIndex      != -1 ) Fields[ WindSpeedIndex ] = new WindSpeed();
        if ( WindDirectionIndex  != -1 ) Fields[ WindDirectionIndex ] = new
WindDirection();
        if ( TemperatureIndex      != -1 ) Fields[ TemperatureIndex ] = new Temperature();
    }
}

```

Sample Code A-18. Entry of new Field index into WXLocation class' initializeFields() method.

After this is done, the Field is ready to be used by any location. What is left is to register this field into those locations that will support it. As described in Section 2, this is done by assigning its Field index a value different than -1. In the Mayaguez class created in Section 2, the new entry would be added as shown in Sample Code A-19.

```

class Mayaguez extends WXLocation {

    public Mayaguez() {
        KeyWord = "mayaguez"
        PressureIndex      = 0;
        RelativeHumidityIndex = 1;
        DewPointIndex      = 2;
        WindSpeedIndex      = 3;
        TemperatureIndex    = 4;
    }
}

```

Sample Code A-19. Entry of new Field index into Location class.

The operator must then modify the `getReadingsFromSource()` method to now also retrieve the data for the new Field and properly organize it in the `WXRecordSet` that it returns.

Implementing a Quality Control Algorithm for a Field

Quality Control is performed by the Field's `validate()` method. This method is implemented in the `WXField` superclass. It takes a `WXRecordSet` as the data to validate and the only validation it does is to discard all those readings not within the Field's range of valid readings. This range of valid readings is written in the Location's XML Data file for those Fields that do not have a fixed range. These can be Fields such as Temperature, or Pressure, where a range of valid readings can be adequate for one Location, but not in another where the temperature or pressure is generally higher or lower. Fields such as Wind Direction however have a fixed range of readings (0 to 359.99), as it doesn't change no matter the location. Others have only one flexible side of the readings range, like Wind Speed, where wind speeds can vary from one location to the other, yet the lowest possible value is always 0.

Subclasses of the `WXField` class, the specific Field classes, can override this method. To implement a Field specific quality control algorithm, the operator needs only

to rewrite the `validate()` method in the `Field` class and implement the algorithm into the software code. This algorithm should discard erroneous readings by replacing their value in the `WXRecord` with a `WXData.INVALID_READING` value. The Data Management classes need no modification to use the newly implemented algorithm, as they always call on the `Field`'s `validate()` method, which if not implemented, uses the one inherited by `WXField`.

Overriding the validating method from `WXField` means that only the one implemented in the `Field` class is used. It is recommended that it also call on the superclass' validate method to discard the readings outside the valid reading range as shown in Sample Code A-20.

```
class Temperature extends WXField {
    ...

    public void validate( WXRecordSet Data ) {
        super.validate( Data );

        // Quality Control Algorithm.

        return Data;
    }
}
```

Sample Code A-20. Overriding the validate method of `WXField`.

Changing a Graph for a Field.

Graphs are generated at the `WXField` class. All graphs generated here are typical line graphs. It provides several methods for displaying different types of graphs. These methods are:

- *getDayGraph()*. For generating graphs for readings in one day of data.
- *getMonthFullGraph()*. For generating a graph for all the readings in one month.

- *getMonthStatGraph()*. For a graph of statistical data for each day in the month.
- *getYearFullGraph()*. A graph for all the readings in a year.
- *getYearStatGraph()*. Statistical data for each month in the year.

Each of these methods takes in different parameters depending on the type of graph. They all require a *WXDay*, a *Location* object, a *Field* index for the field they are to draw a graph from. The *Stat* graphs require a statistical index as well. They each then retrieve the data they need, through the classes in the *Data Management* component and generate the graph in a *BufferedImage* object. Which later on is converted to a *JPG* image file and stored in the *Compiles* directory before it is displayed on the *GraphTable* page.

For an operator to implement a unique style of chart or graph for a *Field* he needs to override the methods for the graphs he wishes to change. Inside the method he must retrieve the data required, create the *BufferedImage* object, and use the *JDK's Graphics* interface to draw the graph elements on the *BufferedImage*. Then return this *BufferedImage* with the graph to the calling class.

Appendix B

User's Manual

B.1 Location Page

The user arrives at this Location page through any of the links provided at the website's homepage. This page provides important information about the current Location and from here the user can retrieve any table of data or compile a data set. To retrieve these data, he/she needs to fill out the displayed forms for data request or data compilation.

At the top is displayed relevant information about the Location. First the Owner, this is the person or institution that operates the weather station and provides us with their data. Next are the geographical coordinates of the Location. The altitude of the Location over sea level in the unit provided. The Data Collection Range are the dates of the first and last record of data for this Location. The user must always request data within this time frame. Anything before or after this time frame will simply display empty pages of "not available" data. Finally, two important statistics are the Highest and Lowest temperature readings of that Location, in British units and Metric in parenthesis. Next to them, the dates of when those readings occurred and a link to open a DayDataTable of that date.

Next, the user has the "Go To Date" form. From here he can request any data table. The date is entered in the textboxes labeled "Day", "Month", and "Year" under the "Date" heading. If the user wishes to request a MonthDataTable, the "Day" entry can be ignored. For a YearDataTable only the "Year" entry is relevant. Next he chooses the unit system to view the data by selecting one of the British or Metric options. Finally, a Data Type. The first option is to view a DayDataTable of all the readings for the date entered

in the Date form. It is labeled by the readings interval of that particular Location, for the latest date in the time range. The second option are the daily statistics for the month entered, and the third are the monthly summaries for the year in the “Year” textbox.

Finally, by clicking on the “Go!” button, or by pressing the “Enter” key, a new window is opened with the requested data.

Mayaguez

Owner: University Of Puerto Rico
 Latitude: 18°13'
 Longitude: 67°11'
 Altitude:
 Data collection range: October 18, 2000 - June 4, 2003
 Lowest recorded temperature: 59.5 °F (15.3°C) on [April 27, 2001](#)
 Highest recorded temperature: 108.9 °F (42.7°C) on [June 6, 2003](#)

Go to date

Date	Unit System	Data Collection Type
Day: <input type="text" value="4"/>	<input checked="" type="radio"/> British	<input checked="" type="radio"/> 10 minutes interval for day
Month: <input type="text" value="6"/>	<input type="radio"/> Metric	<input type="radio"/> Daily summaries for month
Year: <input type="text" value="2003"/>		<input type="radio"/> Monthly summaries for year

Go!

Compile data

Fields	Date range	Unit system	Data Collection Type
<input type="checkbox"/> Temperature	Day Month Year	<input checked="" type="radio"/> British	<input checked="" type="radio"/> 10 minutes interval for day
<input type="checkbox"/> Dew Point	From: <input type="text"/> / <input type="text"/> / <input type="text"/>	<input type="radio"/> Metric	<input type="radio"/> Daily summaries for month
<input type="checkbox"/> Relative Humidity	To: <input type="text"/> / <input type="text"/> / <input type="text"/>		<input type="radio"/> Monthly summaries for year
<input type="checkbox"/> Heat Index			
<input type="checkbox"/> Pressure			
<input type="checkbox"/> Precipitation			
<input type="checkbox"/> Wind Speed			
<input type="checkbox"/> Wind Direction			
<input type="checkbox"/> Flux Density			

Compile!

Figure B-1. Screenshot of Location Page

At the bottom of the page is the Compile Data form. This is the form that allows the user to compile a set of data for a given time range, and fields, in a single continuous time series. On the left is a list of Fields that the Location supports. The user must select at least one of these fields to compile data for. Next is the time range selection form. The time range is provided by entering the two dates in the “From” and “To” date forms. Next the user must select one of the two unit systems. And finally the data collection type. The first option will display all the recorded readings at the Location’s time interval for the given time range. The second option gives daily summaries of the statistics for the selected Fields on all the days in the time range. Monthly summaries are retrieved with the third option. The data is retrieved and displayed in a new window by pressing the “Compile!” button.

Given that data compilation can produce excessive amounts of data request which can overwhelm the web server, limits on the extent of time ranges have been set. This also protects the server from an erroneous input from the user in the time range. Currently there is a 50 day limit for compiled readings, a 500 day limit for daily summaries, and a 500 month limit for month summaries.

B-2. Viewing Data Pages

After entering the requested data on the “Go to date” form, clicking the “Go!” button brings up a data page. The kind of data page will correspond to the “Data page type” option selected on the “Go to date” form. All these data pages share several sections in common: A data header, a navigation bar, a stats bar, and the data table.

The data header displays information about the data requested and options to modify its display. At the top is displayed the Location’s name, and the date of the data retrieved. A selection box of available Location’s is provided to open a new data page of the current date for another location. The first element in the lower options is a link to the

Location page. This is provided in case the user has navigated into the current data page from a data page of another Location. Next there is a link to download the CSV file of the data on the page. This file can be easily imported into mainstream spreadsheet applications such as MS Excel. The following option allows the user to switch the unit system the data page is displayed in to the alternate system currently set. The next option, not available on the Year Data page, opens a new data page of a higher time level than the current page. This means, on a Day Data page it opens a Month Data page for the current month; and on a Month Data page, a Year Data page for the current year. Finally is a link to a black and white version of the data more suitable for printing.

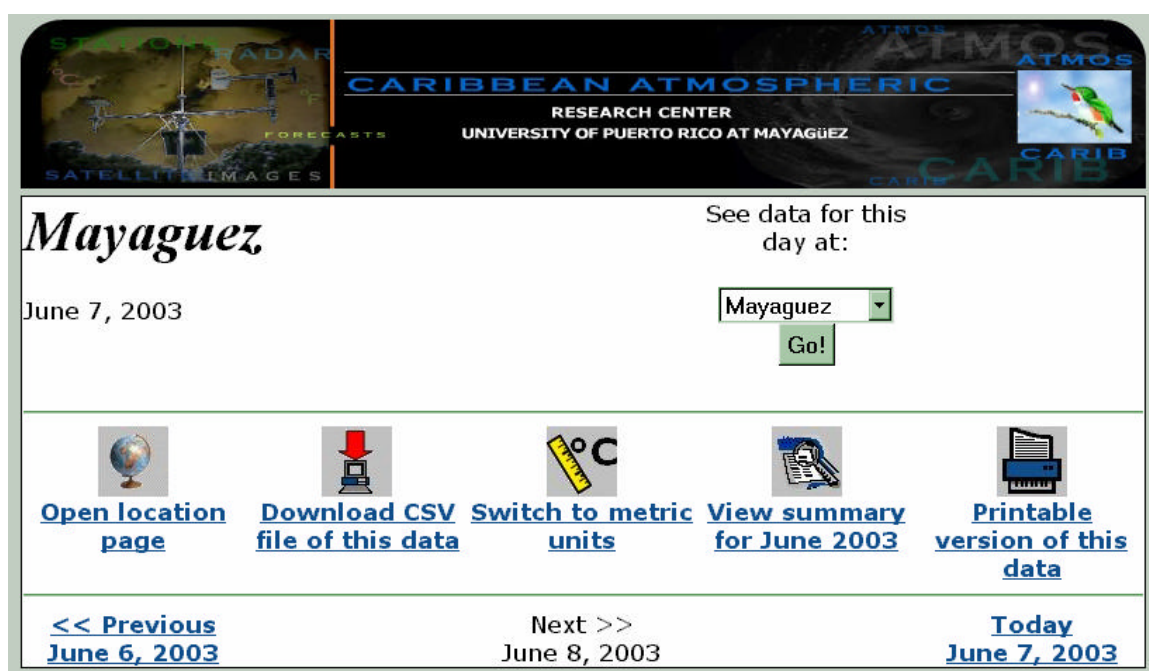


Figure B-2. Screenshot of Data Header

At the bottom of the header is a navigation bar. These links take the user to other data pages of the same type and Location at different dates. The first two are links to the immediately previous and next time units. These are days for Day Data pages, months for Month Data pages, and Years for Year Data pages. The last is a link to the actual current day, month or year. Any of these links will be disabled if the date they lead to are not within the Location's time range of data.

The data table is particular for every data page type. At the top is a graphs section. Where links to all the available graphs for every field are provided. Clicking on these links opens a new window with the requested graph image. Below is a statistics section, with all the relevant statistics for every field calculated from the data set in the data table. Invalid reading markers are placed where a particular Field does not support a statistic. Finally, there is the data. The files indexes are at the left-hand column, ordered by time, day, or month depending on the data page type. In the case of Month Data and Year Data pages, these indexes are also links to the Day Data or Month Data page of the date they display. Special statistics such as the highest and lowest reading in the data set are highlighted in the data area in red and blue accordingly.

Graphs	Line graph	Line graph	Line graph	Line graph	Line graph	Bar graph	Line graph	Line graph	Line graph
						Acumulative line graph		Wind rose chart	
High	90.7	72.6	91.3	96.6	30.01	0.0	5.4	-	1615.0
Avg	80.39	69.1	70.32	82.27	29.96	-	2.08	70.1	444.6
Low	70.1	66.32	46.05	67.2	29.89	-	-	-	-
Total	-	-	-	-	-	0.0	-	-	-
Time	Temperature (°F)	Dew Point (°F)	Relative Humidity (%)	Heat Index (°F)	Pressure (inHg)	Precipitation (in)	Wind Speed (mph)	Wind Direction (°)	Flux Density (μmol/s/m ²)
12:10 A.M.	74.1	70.3	88.2	72.8	30.0	0.0	1.37	8.87	0.0
12:20 A.M.	74.0	70.2	87.3	72.7	30.0	0.0	0.97	226.2	0.0
12:30 A.M.	74.3	70.0	86.5	73.4	30.0	0.0	0.43	15.94	0.0
12:40 A.M.	74.1	70.2	88.2	72.9	30.0	0.0	0.92	0.0	0.0
12:50 A.M.	73.6	70.0	88.6	72.0	30.0	0.0	0.16	1.7	0.0
1:00 A.M.	73.4	69.83	88.5	71.6	30.0	0.0	0.5	36.88	0.0
1:10 A.M.	73.2	69.83	89.9	71.2	29.98	0.0	0.99	43.86	0.0
1:20 A.M.	73.1	70.0	89.9	70.6	29.98	0.0	0.99	98.8	0.0
1:30 A.M.	73.1	69.76	89.1	70.9	29.98	0.0	1.44	0.47	0.0
1:40 A.M.	72.8	69.71	90.4	70.3	29.98	0.0	1.15	77.1	0.0
1:50 A.M.	72.5	69.66	90.6	69.68	29.98	0.0	1.06	93.7	0.0
2:00 A.M.	72.6	69.62	89.4	70.0	29.98	0.0	1.55	96.6	0.0
2:10 A.M.	72.7	69.36	89.8	70.4	29.97	0.0	1.22	24.43	0.0
2:20 A.M.	72.1	69.2	90.0	69.18	29.97	0.0	1.13	56.4	0.0

Figure B-3. Screenshot of Data Table

B.3 Compiling data

Data is compiled by filling the “Compile Data” form in the Location page. It is necessary to select at least one Field of those available for that Location. Then providing the time range of data to compile, by entering the start and end date of the time range. A unit system to view the data is selected and one of the Data Type options.

The Data Type options correspond to the Day Data, Month Data and Year Data pages available through the “Go To Date” form. The first option, will retrieve all readings for the selected fields at the Location’s readings interval for all dates in the provided time range. The “Daily summaries” option brings up the daily summaries, the statistical data for every day and the selected fields, for all the day in the time range. “Monthly summaries” retrieves the statistical data for the selected fields for every month in the time range.

All Compiled Data Pages display a header similar to the Data Pages. They display the Location name, and an option to compile data for the same time range at another Location. They also provide links to open the Location page, download the compiled data in a CSV file, switch to the alternate unit system, display the data in a printer friendly version, and to compile the data at a higher time level than the current one. This means in the Compiled Readings Page, an option to view the Daily Summaries for all the days in the current time range. There is no navigational bar in Compiled Data Pages.

Below the header is the data table. At its top are statistics for the selected fields calculated from the data set in the requested time range. Statistics not supported by a particular field are marked with an invalid reading marker.

High	73.3	81.17	108.9	-
Avg	69.46	77.35	88.2	49.91
Low	65.41	71.4	77.5	-
Total	-	-	-	-
Date	Temperature (°F)			General Wind Direction (°)
	Low	Avg	High	
March 27, 2003	71.4	76.13	85.5	57.46
March 28, 2003	68.8	75.63	87.4	75.73
March 29, 2003	68.62	75.15	85.2	38.31
March 30, 2003	70.0	77.66	86.6	17.3
March 31, 2003	66.81	76.95	87.6	10.39
April 1, 2003	69.58	74.22	85.7	62.11
April 2, 2003	69.76	76.13	86.7	63.4
April 3, 2003	69.43	74.84	85.5	57.3
April 4, 2003	67.93	76.78	86.9	29.68
April 5, 2003	70.5	78.04	89.4	62.23
April 6, 2003	73.3	81.17	108.9	17.85
April 7, 2003	72.0	80.24	96.9	75.9
April 8, 2003	73.2	80.46	96.9	55.39
April 9, 2003	72.3	79.09	86.4	67.81
April 10, 2003	68.87	77.68	89.0	66.84
April 11, 2003	70.1	76.5	87.0	70.72
April 12, 2003	70.4	78.37	88.1	78.23
April 13, 2003	69.43	76.05	89.1	62.96

Figure B-4. Screenshot of Compiled Data Table

The compiled data is then displayed ordered by date and time. The left hand column is the date index. In the case of Compiled Readings page, there is a column for the date and one for the time of day within that date. For daily summaries there is only a date column, which also serves as links to Day Data Pages for the date they represent. And for monthly summaries, only the month is displayed with a link to its corresponding Month Data Page. Statistics highlighted in the data table correspond to the highest and lowest value in their data set, marked as red and blue respectively.

B.4 Viewing graphs

A graph is displayed by clicking on any of the corresponding graph links for a Field on a Data Page. All graphs have a distinct interface, which allows the user to navigate between Locations, Fields and Dates and quickly retrieve more desired graphs. This is achieved using the “Jump to” form at the top of the Graph Page. This form allows the user to input a different date, location, or field and retrieving the new graph by pressing the “Go!” button.

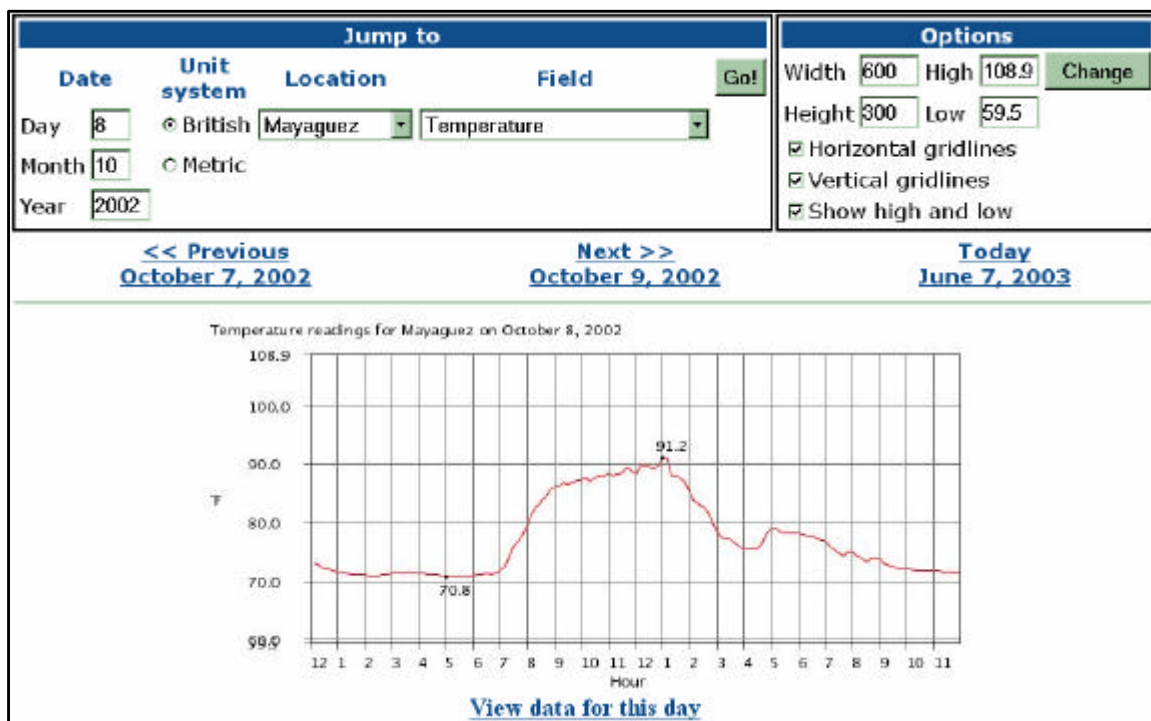


Figure B-5. Screenshot of Graph interface

The “Options” form allows the user to change some visual elements of the graph image. The size of the image can be modified by entering new dimensions (in pixels). The Y-Axis can also be modified by changing the graph’s High and Low values. And other visualization options can be set or unset, such as gridlines, and displaying important statistics on the graph image. These settings don’t apply to all types of graphs. The graph is redrawn with new changes by pressing the “Change” button.

It should be noticed that the “Jump to” and “Options” forms are independent of each other. A change in the date field will not be applied if the “Change” button is

pressed. Likewise a change in the Y-Axis field for example will not be reflected on the graph image by clicking “Go!”. If a user wishes to change the graphs date and visualization, he must first enter the desired options, click “Change”, and when the new graph is loaded, enter the new date he needs, and click “Go!”.

Below these forms is a navigational bar. The links correspond to the graph’s time level. If viewing a graph for a day’s readings, the links will lead to the next and previous day, and they actual current day. For a graph of daily statistics for a month, the links lead to previous, next and current months. These links are disabled if the date they represent are not in the Location’s time range.

Finally, there is the graph image. This is the JPG image produced by the Field’s corresponding `getGraph()` method. They will vary from Field to Field, and not all options setting are applicable. In the case of a Wind Rose for instance, the gridlines options are irrelevant to the graph’s visualization. The graph images can be downloaded into the user’s computer easily by using the browser’s save image mechanism. In MS Explorer and Netscape Navigator, it can be done by right-clicking on the image itself, and selecting the “Save picture as...” option.