CBIML
A Markup Language for Scientific Data Sets

GRADUATE PROJECT REPORT

Submitted to the Faculty of
The Department of Mathematical and Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, TX

In Partial Fulfillment of the Requirements for the Degree of
MASTER of SCIENCE in COMPUTER SCIENCE

By
Jonathan Scott Duff
Summer 2001

Committee Members

Dr. Patrick R. Michaud
Chairperson

Dr. Michelle Moore
Member

Dr. Holly Patterson-McNeill
Member
ABSTRACT

This project is the development of a markup language to describe the scientific data sets produced by the Conrad Blucher Institute for Surveying and Science (CBI). A Document Type Definition (DTD) has been created to describe the grammar of the markup language and software has been written to convert CBI data sets to the markup format. Prototype software has also been created to illustrate how others may read the resulting markup format and perform useful queries on the dataset that it describes.
In this example, the text `<para>` is the start tag and `</para>` is the end tag. The enclosed text is the paragraph described by this markup. The tags themselves are meant to be read and processed by special software that understands the markup language and how it is to be processed. This special processing software is typically called a browser if it is to be displayed for human consumption. By marking or tagging sections of text in this manner, the browser software may process similarly marked text consistently.

Also, the tag name is not the only text that may be enclosed in the markup delimiters. An optional attribute list may follow the tag name in the start tag. An attribute list consists of a series of text separated by white space characters. These attributes may or may not have values associated with them. If an attribute has a value, it appears as attribute="value", otherwise it is unadorned. For instance, in the above example, had the `<para>` tag had attributes that define the font that the paragraph should be rendered in (if possible) and whether or not the font should appear in boldface, the markup may have looked like this:

```
<para>This is a paragraph</para>
<para font="arial" boldface>This is another paragraph</para>
```

Other groups have used XML-compliant markup languages for data transfer. The Astronomical Markup Language (AML) is used to describe astronomical objects in a consistent manner so that astronomers can share information [Guillaume1999]. The Math Markup Language (MathML) allows for unambiguous syntax in describing mathematical formulas and functions [Buswell1999]. The Chemical Markup Language
(CML) provides scientists with a method to transfer complex chemical information without regard to formatting details and possible misinterpretations [Murray-Rust1995].

A major advantage of such wide spread use of XML technology is that applications and software libraries for parsing the XML format already exist. By utilizing an XML-compliant format, it is easy for other organizations to parse CBI’s data sets.
TABLE OF CONTENTS

ABSTRACT ........................................................................................................ ii
TABLE OF CONTENTS ....................................................................................... iii
LIST OF TABLES ............................................................................................... v
1. Background and Introduction .................................................................... 1
   1.1. The Conrad Blucher Institute ............................................................... 1
   1.2. Extensible Markup Language ............................................................. 3
2. Conrad Blucher Institute Markup Language ............................................. 6
   2.1. Metadata .............................................................................................. 6
       2.1.1. Station information ................................................................. 7
       2.1.2. Information about the data set ............................................... 7
       2.1.3. Data/metadata revision history .............................................. 8
       2.1.4. CBIML versions ...................................................................... 8
   2.2. Datasets ............................................................................................... 8
       2.2.1. Six-minute data ........................................................................ 8
       2.2.2. Bench mark descriptions ........................................................ 9
       2.2.3. Bench mark levels .................................................................... 10
       2.2.4. Monthly summary .................................................................. 10
       2.2.5. Tidal datum publication .......................................................... 11
   2.3. CBIML ................................................................................................ 12
       2.3.1. XML markup ........................................................................... 12
       2.3.2. DTD Markup ........................................................................... 13
3. Environment ............................................................................................... 18
4. System Design ........................................................................................... 19
   4.1. Data dictionary ................................................................................... 19
   4.2. Initial DTD .......................................................................................... 19
   4.3. CBIML DTD Description .................................................................. 19
   4.4. DTD Design Considerations ............................................................. 24
   4.5. Generating CBIML ........................................................................... 28
   4.6. Validating CBIML ............................................................................ 31
   4.7. Refining the DTD ............................................................................. 32
   4.8. Prototype CBIML Application ......................................................... 32
   4.9. Publishing the CBIML DTD ............................................................. 35
5. Results ....................................................................................................... 36
6. Future Work .............................................................................................. 37
   6.1. More serinfo detail ............................................................................ 37
   6.2. Enhanced sixmin subroutine ............................................................ 37
   6.3. Incremental output ........................................................................... 38
   6.4. Revision Histories ............................................................................ 38
   6.5. XML Schemas ................................................................................... 39
6.6. Other XML advances ............................................................................. 39
6.7. Other data types .................................................................................. 39
APPENDIX A. Data Dictionary .................................................................... 42
APPENDIX B. Preliminary DTD for CBIML ............................................... 46
APPENDIX C. Source code for CBIML.pm library ................................... 50
APPENDIX D. Source code for cbimlcheck ............................................... 60
APPENDIX E. Final DTD for CBIML ........................................................... 61
APPENDIX F. Source code for prototype CBIML Querier ....................... 65
LIST OF TABLES

Table 5.1. CGI Applications ................................................................. 28
Table 5.2. CBIML.pm Subroutines ......................................................... 29
1. Background and Introduction

1.1. The Conrad Blucher Institute

The Conrad Blucher Institute for Surveying and Science (CBI) collects environmental parameters from over 50 meteorological stations along the Texas coast. These data are used for a variety of purposes including but not limited to hurricane response, oil spill response, littoral boundary determination, and tidal datum publication.

Much of these data are collected at six-minute intervals. The data that are not collected at six-minute intervals are normalized in some fashion to be on a six-minute boundary. The reason for this normalization is that CBI stores and disseminates its data in a line-oriented format where each line represents one record and each record represents one hour of data. Each record contains a station identifier, the date and time of the data, an identifier that represents the environmental parameter collected and ten data values, one for each six-minute time period within the hour starting at the beginning of the hour. By having a consistent format, researchers around the world are able to write software that can easily read and parse any data from CBI and perform whatever analysis they desire.

However, there are several problems with this format. Metadata, such as the name of the station that the station identifier represents, where the station is located, and the units of the data must be given in an ad hoc fashion. Currently, such information is included as comment lines within the data set. Those lines beginning with an octothorpe
(hash mark) are considered commentary and have no fixed, regular format. This makes it difficult for automated systems to parse the metadata.

Also, as CBI expands its capabilities to collect newer meteorological parameters, the six-minute format is becoming unwieldy. Newer data sets of variable data rates do not easily fit into the six-minute format. For instance, a project to study waves in a particular bay may collect water-level readings at 10 Hz for an hour. How can CBI disseminate these data in a format that is usable to other researchers?\(^1\) Currently, problems of this type are solved on a per-project basis and usually the data is disseminated in the same format in which it was collected.

CBI generates other data products in addition to the six-minute data. Some of these products are: bench mark descriptions and elevations, tidal datums\(^2\), monthly summaries of the data quality at each station, and reports concerning maintenance visits to stations. Project sponsors, researchers at CBI, and others use these data products. CBI would like to have a common format that can be processed easily by computer software, represent CBI data sets and associated metadata, and that is flexible enough to be modified to accommodate new datasets. This goal is achieved by providing CBI with its own markup language that completely describes the data and meta-data associated with it. Numerous standards have been developed for representing data and metadata,

---

\(^1\) Newer data sets that do not fit easily into the six-minute format also present storage problems, but that is beyond the scope of this proposal.

\(^2\) Datum is used here in the surveying sense as a base from which heights are measured, thus the plural is "datums" rather than "data". Example tidal datums are: mean high water, mean sea level, and mean low water.
but none have been widely adopted. One standard that is gaining popularity is the Extensible Markup Language (XML).

1.2. Extensible Markup Language

XML is a subset of the Standard Generalized Markup Language (SGML). SGML is an international standard adopted by the International Standards Organization (ISO Standard 8879 [ISO1986]) as a markup language for describing markup languages. For example, the specification for the now common Hypertext Markup Language (HTML) was formulated in SGML [Berners-Lee1995]. Thus, XML also describes a markup language for describing markup languages.

Markup languages are sometimes called tag languages or descriptive languages because they provide tags (markup) that describe the text that they mark. XML-compliant markup languages enclose tags within < and > characters to distinguish markup from non-markup text.\(^3\) A special character sequence is provided to represent the characters < and > independent of their special meaning as markup delimiters.

Typically, markup comes in pairs: a start tag and an end tag. The end tag looks just like the start tag except that the tag name is prefixed with a / character. As an example, a markup language might provide a tag called \texttt{para} for describing certain text as a paragraph. Here are two paragraphs written in such a markup language:

\begin{verbatim}
<para>This is a paragraph</para>
<para>This is another paragraph</para>
\end{verbatim}

\(^3\) In SGML, the markup delimiting characters may be redefined.
2. Conrad Blucher Institute Markup Language

The primary objective of this project was to create an XML-compliant markup language that describes the data sets produced by CBI. This markup language has been dubbed Conrad Blucher Institute Markup Language (CBIML). There are five data sets that are described by this markup language:

1. CBI six-minute data,
2. bench mark descriptions,
3. bench mark levels,
4. monthly summaries of tidal parameters, and
5. tidal datum publications.

Associated with each of these data sets is metadata concerning the station where the data were collected, the type of data, the data format, the date and time the data were reported, and any special processing may have been done to the data.

Because these data and metadata are explicitly defined in the markup language, they will no longer be include in an ad hoc format within data sets produced by CBI. Also, by utilizing XML, the markup language may be extended to include new data sets as needed. The next sections list the individual items that are described by CBIML for both the data sets and the metadata.
2.3.2.1. Elements

Elements are the markup (tags) surrounded by < and >. The syntax of the element declaration is <!ELEMENT tagname children>, where tagname is the name of the tag that is being defined and children is a list of tags separated by commas that must appear between the start and end tag for this markup. The order in which these tags are listed is significant. The list of children may also contain these other special characters: *+, ?, |, (, and ). Parentheses are used to group tag names, and all of the other symbols are operators that act on groups of tag names. A tag name by itself is a singleton group.

A group that is followed by a * character must occur zero or more times. Thus, <!ELEMENT station (name,alias*)> defines a tag called station that must contain a name tag followed by zero or more alias tags. A group that is followed by a + character must occur one or more times. A group that is followed by a ? character must occur exactly zero or once. The | character is a binary operator used for alternation. tagA|tagB means that either tagA or tagB may occur at this point in the list of children.

There are also two special values that may appear in addition to a list of child tags. If the value EMPTY appears by itself, it means that there must be nothing between the start and end tags of the element. If the value #PCDATA appears anywhere in the list of child tags, it means that any character data may appear between the start and end tags of the element.
Below is an example DTD that defines tags describing information about data collection stations:

```xml
<!ELEMENT stationinfo name,alias*,location>
<!ELEMENT name (#PCDATA)>
<!ELEMENT location (lat,lon)>
<!ELEMENT lat (#PCDATA)>
<!ELEMENT lon (#PCDATA)>
<!ELEMENT alias (#PCDATA)>
```

Note that a `stationinfo` tag must contain both `name` and `location` tags in that order. Changing the order or omitting either the `name` or `location` would make the markup no longer conformant to the DTD. The same applies for the `lat` and `lon` tags. Strict adherence to the DTD assures that all parties involved in processing the data are using the same language. Below is markup that conforms to the DTD given above:

```xml
<stationinfo>
  <name>Naval Air Station</name>
  <location>
    <lat>27.7056</lat>
    <lon>97.2803</lon>
  </location>
</stationinfo>
```

### 2.3.2.2. Attributes

Attributes are a way to modify the behavior of an element (tag) by providing additional information. Attributes for each entity are defined using the syntax `<!ATTLIST tagname attrdefs>`. The `tagname` must match one that has been previously declared as an element. The `attrdefs` is a whitespace separated list of attribute names, attribute types, optional attribute modifiers and optional attribute values. A simple attribute list might look like this:

```xml
<!ATTLIST stationinfo
id   CDATA          #REQUIRED
```
In this example, the `stationinfo` tag has two required attributes called `id` and `type`, and an implied attribute called `tidal`. The `id` attribute can have a value of any text. The `type` attribute can only have one of the three values: `sutron`, `vitel`, or `cube`. The `tidal` attribute may have the values `yes` or `no` with an implied value of `yes`.

2.3.2.3. Entities

Entities constitute a fundamental concept in SGML/XML. In the simplest form they merely provide a way to define shorthand for a common sequence of characters. There are two forms of entity: parameter and general. Parameter entities consist of some text prefixed by a percent sign (`%`) and suffixed by a semicolon (`;`). It is important to note that parameter entities are only used within the DTD itself and cannot be used within the markup described by the DTD. This is contrasted with general entities which can be used within the DTD and within the markup described by the DTD.

General entities consist of some text prefixed by an ampersand (`&`) and suffixed by a semicolon (`;`). There are two types of general entity: internal and external. An internal general entity is declared as follows: `<!ENTITY name "value">`. After the entity declaration, anywhere the text `&name;` appears, it will be substituted with the value associated with that name (even within other entities). For example, if you were manually creating a document that used the phrase "The Conrad Blucher Institute for
3. Environment

This project utilizes the existing computing infrastructure available at CBI. This infrastructure includes: various personal computers running RedHat Linux 7.0, Perl 5.6.0, MySQL 3.22.30, and Apache 1.3.14.

The programming language Perl was used to convert CBI’s existing data formats into CBIML. Perl was chosen because CBI already uses Perl extensively for data processing tasks and for disseminating data via the Common Gateway Interface (CGI) feature of the Apache Web server. The specification of XML version 1.0 available from the World Wide Web consortium has been used to verify XML compliance and compatibility [Bray 1998].

Since there appear to be few good XML or DTD authoring tools, the standard UNIX™ editor, vi, was used to construct the DTD. The freely available Perl module XML::Parser was used to validate the syntax of the XML documents against the DTD as well as check the syntax of the DTD itself.
4. System Design

This section is a design description of the the DTD for CBIML, the software to generate CBIML, and the prototype query software. Also described within this section are the ancillary data products that were used to create the CBIML DTD and CBIML software.

4.1. Data Dictionary

To create a markup language for CBI data sets, all of the data items were identified and described in a clear and concise manner. The relationship between data items was also clearly identified. A data dictionary has been created that lists each data item, its salient characteristics (for instance, data type, legal values, numeric range) and relationship to other data items. The data dictionary is given as Appendix A.

4.2. Initial DTD

An initial DTD was created that describes the data items listed in the data dictionary. This DTD was built up as a series of logical pieces. Unrelated data items each have their own section within the DTD. Appendix B contains the initial DTD created before conferring with CBI personnel.

4.3. CBIML DTD Description

The DTD for CBIML is composed of several largely independent sections that describe portions of a CBIML document. Rather than placing these portions in individual files, it was decided to use one file for the DTD in the interest of clarity. If, in the future, the DTD becomes too unwieldy as new data types are added, it can easily be broken into individual files.
Each section of the DTD is surrounded by comments denoting the purpose of that section. The following sections are present within the DTD:

1. Parameter Entities
2. CBIML Root Element
3. CBIML Meta-information
4. Station Information
5. Series Information
6. Sixmin Data
7. Bench Mark Descriptions
8. Bench Mark Levels
9. Monthly Summaries
10. Tidal Datums

The following text describes the main features of each of these sections. For a more detailed listing of the DTD, see Appendix C.

4.3.1. Parameter Entities

The first section with the DTD is for parameter entities. These entities are only valid within the DTD unlike general entities, which can be used in the XML document itself. Parameter entities were used for attributes that may occur in several varied places. For instance, those data items that represent elevations should have the datum that the elevation is relative to and the epoch for which that datum was computed.

4.3.2. CBIML Root Element

This section describes the root element of a CBIML document. The root element is the container within which all other markup should appear. For CBIML documents the root
element was called \textit{cbiml}. The \textit{cbiml} element has a required attribute, \textit{version}, which represents the version of CBIML to which the document conforms. The version created by this project is designated as “1.0”.

Contained within the \textit{cbiml} element is a required \textit{cbimlinfo} element and zero or more of any of the following elements: \textit{stationinfo}, \textit{serinfo}, \textit{sixmin}, \textit{benchmark}, \textit{level}, \textit{monsum}, and \textit{datum}.

\textbf{4.3.3. CBIML Meta-information}

A element for meta-information about the CBIML document is called \textit{cbimlinfo}. This element contains a required element called \textit{report-date} and an optional \textit{revhistory} element. The report date is when the CBIML document was generated. The revision history contains a series of comments, who made them and when. The \textit{revhistory} element is used wherever we wish to record changes within a CBIML document.

\textbf{4.3.4. Station Information}

The \textit{stationinfo} element contains meta-information about a data collection station. This element consists of an optional \textit{revhistory} element, followed by a required \textit{stnname} and \textit{field-id}, followed by optional \textit{latlon} and \textit{bay} elements, followed by zero or more aliases for the station.

Additionally, the \textit{stationinfo} element contains a required attribute called \textit{stnid}. This attribute is a unique identifier used to represent this station throughout the CBIML document.

\textbf{4.3.5. Series Information}

The \textit{serinfo} element contains meta-information about data series. Currently the only information within this element is the series name. The \textit{serinfo} element also has a
required attribute, \textit{serid}, which is a unique identifier representing this data series throughout the CBIML document.

4.3.6. Sixmin Data

The \textit{sixmin} element describes data from the six-minute database used within CBI. This element is a container for two different data formats: \textit{smtex} and \textit{sm}. The \textit{smtex} element holds data lines in the format currently used by CBI for disseminating its six minute data. The \textit{sm} element holds a structured version of the six-minute data consisting of a series of timestamps and data values for those times.

The \textit{sixmin} element also has two required attributes: \textit{stnid} and \textit{serid}. The \textit{stnid} attribute is a unique identifier used to reference the six-minute data to the station that the data were collected for. The \textit{serid} attribute similarly references which data series these data are for.

There are also optional attributes for data series that represent elevations (\textit{relto} and \textit{epoch}) and a \textit{units} attribute describing the data units. These attributes are optional on the \textit{sixmin} element because they may also appear as attributes of the \textit{smvalue} element that represents an individual six-minute value.

4.3.7. Bench Mark Descriptions

Bench mark descriptions are contained within the \textit{benchmark} element. The benchmark element has two attributes: \textit{stnid} and \textit{bmid}. The \textit{stnid} attribute refers to the station that this bench mark is a part of and the \textit{bmid} attribute is a unique identifier representing the bench mark we are describing. Child elements of \textit{benchmark} are an optional \textit{revhistory}, \textit{spsn}, \textit{pid}, \textit{estdate}, \textit{estby}, \textit{stamping}, an optional \textit{latlon}, \textit{location}, \textit{monumentation}, \textit{marktype}, \textit{setting}, \textit{county}, \textit{state}, \textit{description}, and \textit{accepted-elevation}.
4.3.8. Bench Mark Levels

To describe bench mark leveling information, a level element was created. This element contains the following elements: an optional revhistory, levdate, chief, levelinfo, rodinfo, observer, levorder, levclass, and elevations. The level element also has two attributes: stnid and levid. The stnid attribute relates this leveling information to a particular station and the levid uniquely identifies this leveling information.

The elevations element contains a set of elevation elements. Each of the elevation elements contains an attribute called bmid that relates this elevation to a particular benchmark. Each elevation element also has the relevant attributes to relate the elevation to a particular datum and specify the units that the elevation data are in.

4.3.9. Monthly Summaries

Monthly summaries are described with CBIML documents by a monsum element.

Within the monsum element are an optional revhistory element, data-quality, monthly-datums, highs and lows. The monsum element also has attributes stnid and month. The stnid attribute relates this monthly summary to a particular station and the month attribute tells which month the summary is for.

The data-quality element has two attributes that describe the subjective quality of the data called computed-quality and assigned-quality. Also within data-quality are elements that describe the number of good, missing, and removed data values for that month.

The monthly-datums element contains one element for each of the following computed datums: monthly mean higher high water, monthly mean high water, monthly mean tide level, monthly mean sea level, monthly mean low water, and monthly mean
lower low water. The elements for each of these datums are: \textit{mmhhw, mmhw, mmlt, mmsl, mmlw, and mmlw} respectively. Finally, the high\textit{s} and low\textit{s} elements each contain a listing of high all of the high and low water elevations for the summarized month.

4.3.10. Tidal Datums

A \textit{datum} element was created to describe the tidal datum publications. This element contains the following elements: an optional \textit{revhistory}, \textit{datum-status}, \textit{date-computed}, \textit{observation-length}, \textit{observation-period}, and \textit{mean-datums}. The \textit{datum} element also has two attributes: \textit{stnid} and \textit{control}. The \textit{stnid} attribute uniquely identifies the station for which these datums have been computed and the \textit{control} attribute uniquely identifies the station used as a control station for the datum computation. The \textit{mean-datums} element is where the actual computed datums appear.

4.4. DTD Design Considerations

The creation of CBIML involved many design decisions. Below is a list of these considerations:

1. Should the attribute type for stations, bench marks, and leveling runs be \textit{ID} or \textit{NMTOKEN}?

2. Should the series name in \textit{serinfo} be an attribute or an element?

3. Where should the revision history information go?

4. Are the \textit{levels} and \textit{benchmarks} container elements really necessary?

5. Should the units be attributes or elements?

6. Other design considerations.

The remainder of this section describes the design issues and the resulting decisions.
4.4.1. Should the attribute type stations, bench marks, and leveling runs be ID or NMTOKEN?

Within a DTD, attributes may be typed in several ways. An ID attribute can be referred to via an IDREF attribute elsewhere in the document. The advantage of this is that the XML parser will help you keep your IDs and IDREF's straight so that it is readily apparent when you have an ID that is never referred to (most XML parsers output an informational message to this effect) or, more importantly, when your use an IDREF that has no corresponding ID.

Another type of attribute is the NMTOKEN. This is just an identifier. It consists of a sequence of characters. There is no corresponding “NMTOKENREF” type to refer back to an NMTOKEN, thus there is no checking in the XML parser for NMTOKEN correspondence when they are used to represent a relationship between data items.

A limitation for this project is that IDs have very specific syntactic constraints. For example, attributes of type ID can not start with a number. This is problematic because most of the identifiers used within the CBI database start with numeric characters. NMTOKENs on the other hand may start with any character.

It was decided to use ID and IDREF rather than NMTOKEN for the “matching” ability it provides and to let the XML parser help debug the DTD and CBIML document. To encode CBI database identifiers for use as ID attributes each is prefixed with a short alphabetic string of characters. For example, station identifiers are prefixed with “stn”, bench mark description identifiers are prefixed with “bm” and leveling run identifiers are prefixed with “lvl”.

25
4.4.2. Should the series name in serinfo be an attribute or an element?

It seemed a waste to have a tag for series information that contained only one tag for the series name. This would make the series name a good candidate for becoming an attribute of serinfo rather than a separate tag enclosed within the serinfo tag. It was decided to keep sername as a separate tag because it is expected that in the future the serinfo tag will contain more information such as how the data were collected or computed. Thus sername would make sense as markup that is part of a larger structure.

4.4.3. Where should the revision history information go?

The answer is "everywhere." The individual items within a CBIML document such as level or benchmark may have a modification history that is important and so should be recorded. This becomes more important in the future when programs may generate data series (for instance, to fill gaps or produce predictions).

Also, the entire CBIML document may have a modification history, especially if incremental document generation is used extensively. The entire CBIML document may have been pieced together as the output of several different programs, each adding a small piece to the whole and updating the revision history.

4.4.4. Are the levels and benchmarks container elements really necessary?

The original design of the CBIML DTD had container elements called levels and benchmarks that were for holding series of benchmark levels and benchmark descriptions respectively. Because a CBIML document can contain multiple instances of the benchmark and level tags, these container elements were not necessary. Thus, they were removed from the DTD.
4.4.5. Should the unit information be attributes or elements?

Each of CBI’s datasets needs to have dimensional information associated with it so that the user may make sense of the data. For instance, water level data can be represented as a distance expressed in meters or feet, temperature can be expressed as some value on the Celsius or Fahrenheit scale; wind speed can be expressed in meters per second or miles per hour. Without the unit information, researchers can not apply CBI datasets in a useful manner.

In the original DTD design (see Appendix B) the units of CBI’s data were not even considered. After realizing this serious omission, unit attributes were added to a few places within the DTD. It became clear that there were different situations in which units could be applied and that the whole idea of dimensional data needed rethinking.

For some tags, such as accepted-elevation within the bench mark descriptions, the unit is needed for just that one tag. It is also desirable to designate the units for an entire group of data rather than repeat the same unit information for each individual data item. These two situations roughly correspond to treating the unit information as either an attribute or a separate tag respectively. After careful consideration, it was decided to keep the units as an attribute of the various data items.

There is no apparent benefit to `<units>0.1 millibars</units>` over units="0.1 millibars". For aggregate types the units tag appears once somewhere near the start of the aggregate and this is largely the same as if the units were an attribute. It appears just as easy to retrieve attributes of an element, as it is to retrieve elements themselves using one of the freely available XML parsers. And finally, “units”
in the sense that CBI uses them have a very simple structure that is not likely to change over time, thus a simple string attribute is sufficient.

4.4.6. Other design considerations

During the design of CBIML it became clear that several new markup tags and attributes were needed. A tag to hold CBI data in the existing six-minute format was added to the sixmin tag. Within the monsum tag, the count of good data, removed data and missing data were encapsulated within a new tag called data-quality. All tags that represented some sort of elevation needed an attribute representing the datum to which the elevations are relative to and the epoch for which that datum has been computed.

4.5. Generating CBIML

Most of the software that CBI uses to process and disseminate its data is implemented as Perl programs that follow the Common Gateway Interface (CGI) protocol. These programs are executed by an Apache Hypertext Transfer Protocol (HTTP) server that input from either standard input or as arguments passed as part of a browser request. Some example output formats are Hypertext Markup Language (HTML), GIF images, or just plain text. The data used to generate output may come from a variety of data sources such as a MySQL database, a flat file database, or may even be generated on-demand.

All of the data sources used by CBI are collectively referred to as the “CBI database”.

The following table shows existing CGI applications that output the various data that CBI disseminates:

Table 4.1. CGI Applications

<table>
<thead>
<tr>
<th>CGI Application</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmdata.cgi</td>
<td>Outputs CBI data as text or graphs.</td>
</tr>
<tr>
<td>bmarks.cgi</td>
<td>Displays and allows editing of benchmark descriptions and leveling information.</td>
</tr>
</tbody>
</table>
Surveying and Science" many times but did not want to type it out each time, an entity could be defined and used as follows:

```xml
<!ENTITY cbi "The Conrad Blucher Institute for Surveying and Science">
I work for &cbi;.
```

Which would be rendered as:

```
I work for The Conrad Blucher Institute for Surveying and Science
```

External general entities are declared similarly except that the keyword SYSTEM appears before the entity value. The entity value is then taken as a filename containing data that is to be inserted at that point in the document. For example, in order to break a document into three physical pieces, each in its own separate file, the following entity declarations might be used:

```xml
<!ENTITY intro SYSTEM "introduction.xml">
<!ENTITY body SYSTEM "body.xml">
<!ENTITY concl SYSTEM "conclusion.xml">
```

The main document that included these three pieces would look like this:

```
&intro;
&body;
&concl;
```
mktab.cgi  Makes tabulations in preparation for datum computation.
datum.cgi  Outputs tidal datum publications as HTML.

Each of these programs has a parameter that determines what action the program is to take with the given input. All actions that result in record changes (for instance, editing or verifying values) require that the user be authenticated through the basic password authentication mechanism built into the Apache Web Server.

As a practical matter, rather than modifying each individual program to produce CBIML directly, a Perl library module called CBIML.pm was created. This module contains subroutines that transform each data type from the representation used by the CBI database into CBIML. There is one subroutine for each of the main elements of a CBIML document.

This approach was taken to make it easy to relate the DTD for CBIML to the software that generates CBIML and to make code maintenance easier when CBIML changes. Also, this allows for the individual pieces of a CBIML document to be generated separately and assembled in whatever order is necessary for the DTD. Thus this library can be considered a set of primitives with which to build complex CBIML documents. The following Perl subroutines were created within the CBIML.pm module and correspond directly to the CBIML tags of the same name:

<table>
<thead>
<tr>
<th>Table 4.2. CBIML.pm Subroutines</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbiml</td>
</tr>
<tr>
<td>cbimlinfo</td>
</tr>
<tr>
<td>stationinfo</td>
</tr>
<tr>
<td>serinfo</td>
</tr>
<tr>
<td>benchmark</td>
</tr>
<tr>
<td>level</td>
</tr>
<tr>
<td>sixmin</td>
</tr>
<tr>
<td>monsum</td>
</tr>
</tbody>
</table>