SGML Functions for AthenaMuse 2

by

James C. Gentry

Submitted to the Department of Electrical Engineering and Computer Science on May 28, 1996 in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science and Engineering

ABSTRACT

The Standard Generalized Markup Language provides a way to specify markup languages. With these markup languages, diverse types of information can be organized. SGML makes the storage and sharing of information easier, and it is being used by an increasingly large number of publishers.

AthenaMuse 2 is a tool used to create multimedia computer programs, which is currently being developed at the Center for Educational Computing Initiatives of the Massachusetts Institute of Technology. AM2 uses an object-oriented language called the Application Description Language (ADL) to script multimedia programs.

This paper describes a group of classes, written for the ADL, which can parse and dissect SGML documents. These classes, collectively called AM2SGML, will enable the AM2 application developer to use SGML in his applications, and in the future will possibly provide the base for a new set of ADL Hypertext Markup Language (HTML) classes.

Thesis Supervisor: Steven Lerman
Title: Professor of Civil and Environmental Engineering; Director, Center for Educational Computing Initiatives
Acknowledgments

There are many people I would like to acknowledge for helping me complete this paper.

First, I would like to thank everyone at the Center for Educational Computing Initiatives. I especially thank Professor Steven Lerman for his guidance through every stage in this project. I also thank Ana Beatriz Chiquito for first introducing me to CECI.

I would like to thank my Brothers and friends at Pi Lambda Phi, Massachusetts Theta, for giving me support at the times when I did not happen to be working on this or any other projects. And finally, I would like to thank my family, for reasons which go without saying.
Table Of Contents

1 Motivation .............................................................................................................................13

1.1 SGML ...................................................................................................................................13

1.2 AM2 .....................................................................................................................................14
  1.2.1 Better HTML Functions .................................................................................................14
  1.2.2 The Current State of HTML in AM2 .............................................................................15

1.3 SGML in AM2 ......................................................................................................................16

2 Specifications ........................................................................................................................17

2.1 Purpose................................................................................................................................17

2.2 Class Specifications ..............................................................................................................17
  2.2.1 The DTD Classes ..........................................................................................................20
    2.2.1.1 The MMdtd Class ..................................................................................................20
    2.2.1.2 The MMdtdDocumentType Class .........................................................................22
    2.2.1.3 The MMdtdElement Class ...................................................................................26
    2.2.1.4 The MMdtdContentModel Class .........................................................................29
    2.2.1.5 The MMdtdAttribute Class ..................................................................................32
    2.2.1.6 The MMdtdEntity Class .......................................................................................36
    2.2.1.7 The MMdtdNotation Class ....................................................................................38
    2.2.1.8 The MMsgmlSectionMark Class .........................................................................40
    2.2.1.9 The MMsgmlComment Class ...............................................................................41
  2.2.2 The Document Instance Classes ......................................................................................42
    2.2.2.1 The MMsgmlDocument Class .............................................................................42
    2.2.2.2 The MMsgmlElement Class .................................................................................44
    2.2.2.3 The MMsgmlPCData Class ...................................................................................45
    2.2.2.4 The MMsgmlEmpty Class .....................................................................................46
    2.2.2.5 The MMsgmlAttribute Class ...............................................................................47

3 The AM2SGML C++ Classes ..................................................................................................49

3.1 YASP ....................................................................................................................................49
  3.1.1 Why YASP? ...................................................................................................................49
  3.1.2 An Overview of YASP ..................................................................................................50

3.2 Overview of the AM2SGML C++ Functions .......................................................................50

3.3 AM2SGML Module Descriptions .......................................................................................52
  3.3.1 The Test Function ........................................................................................................52
  3.3.2 The parseSGMLFile Function .....................................................................................52
  3.3.3 The dispatch Function ................................................................................................54
  3.3.4 The SGMLObject Class and its Subclasses ..................................................................56
    3.3.4.1 The SGMLDocument Class ...............................................................................56
    3.3.4.2 The SGMLElement Class ....................................................................................57
  3.3.5 The Service Functions ................................................................................................58
    3.3.5.1 The eltSt Function ...............................................................................................58
    3.3.5.2 The eltNd Function .............................................................................................59
    3.3.5.3 The text Function ...............................................................................................59
<table>
<thead>
<tr>
<th></th>
<th>File</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.20</td>
<td>text.h</td>
<td>83</td>
</tr>
<tr>
<td>B.21</td>
<td>pi.h</td>
<td>83</td>
</tr>
<tr>
<td>B.22</td>
<td>msg.h</td>
<td>83</td>
</tr>
<tr>
<td>B.23</td>
<td>msg.cc</td>
<td>83</td>
</tr>
<tr>
<td>B.24</td>
<td>sGet.h</td>
<td>85</td>
</tr>
<tr>
<td>B.25</td>
<td>sGet.cc</td>
<td>85</td>
</tr>
<tr>
<td>B.26</td>
<td>sFre.h</td>
<td>85</td>
</tr>
<tr>
<td>B.27</td>
<td>find.h</td>
<td>85</td>
</tr>
<tr>
<td>B.28</td>
<td>find.cc</td>
<td>86</td>
</tr>
<tr>
<td>B.29</td>
<td>xEnOp.h</td>
<td>88</td>
</tr>
<tr>
<td>B.30</td>
<td>xEnOp.cc</td>
<td>88</td>
</tr>
<tr>
<td>B.31</td>
<td>utOpR.h</td>
<td>88</td>
</tr>
<tr>
<td>B.32</td>
<td>utOpW.h</td>
<td>89</td>
</tr>
<tr>
<td>B.33</td>
<td>utOpW.cc</td>
<td>89</td>
</tr>
<tr>
<td>B.34</td>
<td>utWrt.h</td>
<td>89</td>
</tr>
<tr>
<td>B.35</td>
<td>utWrt.cc</td>
<td>89</td>
</tr>
<tr>
<td>B.36</td>
<td>utRd.h</td>
<td>90</td>
</tr>
<tr>
<td>B.37</td>
<td>utRd.cc</td>
<td>90</td>
</tr>
<tr>
<td>B.38</td>
<td>utCl.h</td>
<td>90</td>
</tr>
<tr>
<td>B.39</td>
<td>xEnRd.h</td>
<td>91</td>
</tr>
<tr>
<td>B.40</td>
<td>xEnRd.cc</td>
<td>91</td>
</tr>
<tr>
<td>B.41</td>
<td>xEnCl.h</td>
<td>92</td>
</tr>
<tr>
<td>B.42</td>
<td>xEnCl.cc</td>
<td>92</td>
</tr>
<tr>
<td>B.43</td>
<td>proSt.h</td>
<td>92</td>
</tr>
<tr>
<td>B.44</td>
<td>proSt.cc</td>
<td>92</td>
</tr>
<tr>
<td>B.45</td>
<td>proNd.h</td>
<td>92</td>
</tr>
<tr>
<td>B.46</td>
<td>docNd.h</td>
<td>93</td>
</tr>
<tr>
<td>B.47</td>
<td>etySt.h</td>
<td>93</td>
</tr>
<tr>
<td>B.48</td>
<td>etySt.cc</td>
<td>93</td>
</tr>
<tr>
<td>B.49</td>
<td>etyNd.h</td>
<td>93</td>
</tr>
<tr>
<td>B.50</td>
<td>skip.h</td>
<td>94</td>
</tr>
<tr>
<td>B.51</td>
<td>decSt.h</td>
<td>94</td>
</tr>
</tbody>
</table>
Appendix C - Sample Run

Appendix C - Sample Run ................................................................. 109
List of Tables

TABLE 1. The AM2SGML Classes.................................................................19
TABLE 2. Members of MMdtd ..................................................................22
TABLE 3. Members of MMdtdDocumentType .............................................26
TABLE 4. Members of MMdtdElement .......................................................29
TABLE 5. Members of MMdtdContentModel ..............................................31
TABLE 6. Possible Types of hDefaultVal ...............................................34
TABLE 7. Members of MMdtdContentModel ..............................................35
TABLE 8. Members of MMdtdEntity ..........................................................38
TABLE 9. Members of MMdtdNotation .......................................................39
TABLE 10. Members of MMsgmlSectionMark ..........................................41
TABLE 11. Members of MMsgmlComment ...............................................42
TABLE 12. Members of MMsgmlDocument ..............................................43
TABLE 13. Members of MMsgmlElement ..................................................45
TABLE 14. Members of MMsgmlPCData ..................................................46
TABLE 15. Members of MMsgmlPCData ..................................................48
TABLE 16. The Service Functions ............................................................54
TABLE 17. YASP Message Severities .......................................................60
TABLE 18. YASP File Classes .................................................................62
1 Motivation

AthenaMuse 2® is an authoring tool for creating multimedia computer programs currently being developed at the Massachusetts Institute of Technology's Center for Educational Computing Initiatives.¹ This paper describes the design and partial implementation of a set of classes for AM2 which can be used to manipulate Standard Generalized Markup Language (SGML) documents. This set of classes is collectively called AM2SGML.

1.1 SGML

SGML is an international standard introduced in 1986, described by International Organization for Standardization standard ISO 8879. Because of SGML's many benefits, it is becoming increasingly popular. Today, it is used by such various organizations as the Department of Defense, the Association of American Publishers, and the American Trucking Association. It is gaining widespread acceptance both in the United States and abroad.²

SGML can be used to define the formats of certain types of data using a Document Type Definition (DTD), by specifying which types of elements the data can contain. The data can then be marked up, so that the location and arrangements of the elements in the data are made explicit. SGML allows for more efficient storage of information, it increases the ease of sharing information, and it allows for the easy storage and maintenance of information in databases.³ In addition,

¹ AthenaMuse is a registered trademark of MIT.
³ "Getting Started."
SGML is used to define the Hypertext Markup Language (HTML), which is in wide use in the World Wide Web.

1.2 AM2

AM2 is designed to make the implementation of multimedia applications simpler. One way in which it does this is through its use of the Application Description Language. The ADL has classes to display and manipulate a variety of media types, including images, sounds, movies, and HTML pages.

Currently, the ADL has no functionality for manipulating SGML documents besides classes which can display HTML pages. While AM2 would be greatly improved with the addition of classes to handle SGML in general, the original reason for creating these classes was so that AM2’s ability to manipulate HTML pages would be improved.

1.2.1 Better HTML Functions

While AM2 provides basic functions for displaying HTML pages, following links, and so forth; but it is missing more advanced functions which would be quite useful when writing applications involving HTML. A major problem with the current implementation of the HTML functions is that it is based on software written by a third party. Therefore, the types of functions dealing with HTML which are available to the application programmer are largely limited to those that are provided in the third party library. It is difficult if not impossible to add to or modify these functions. To change this, the implementation of HTML functions for AM2 needs to be redesigned and re-implemented.
1.2.2 The Current State of HTML in AM2

The display of an HTML page can be divided into two stages. First, the HTML source document is parsed into some sort of data structure. In the current HTML functions, this structure is a linked list. Once the parsing is complete, the structure is passed to a function which displays the page in a region of a window on the screen. There are drawbacks in the current implementation of both of these stages.

The linked list structure currently used is not especially convenient to manipulate; because of this, accessing information about the HTML page is not as simple and possibly not as efficient as it could be. Creating a better data structure would not only help with this problem, but would have numerous other benefits as well. Since third-party software would no longer be relied on to parse HTML into a structure, an HTML file could be parsed the same way as any other SGML document. By changing the DTD used for HTML, extra features could even be added to HTML. For example, the idea of defining AM2 classes in HTML has been considered; this would be easier to implement if we handled our own HTML parsing. Also, parts of the HTML structure could be made visible to ADL application programmers as ADL classes. This would allow programmers an easy way to dynamically manipulate and define HTML pages in applications.

There are also drawbacks in the current functions used to display the HTML data structures. Right now it is difficult or impossible to change the way that an HTML page is displayed. For example, properties such as image and text positions and default colors cannot be set by the application programmer. These problem would be solved if custom functions to display HTML were written. There would be additional benefits as well. For example, arbitrary ADL objects could be displayed within an HTML page. Also, other types of SGML documents could be displayed using
the HTML display functions. Unfortunately, due to time constraints, the display of these HTML structures is not in the scope of this project.

1.3 SGML in AM2

Instead of limiting this project to the parsing and display of HTML documents, it made sense to go one step further, and include within its scope the parsing of general SGML documents also. With basic SGML support built in to the ADL, the ADL application developer would be able to make use of SGML in his applications.

Due to the large amount of work involved with the parsing of SGML documents, a full implementation of the AM2SGML specifications has not been completed. However, a “bare bones” implementation has been done. The AM2SGML specifications and a description of the implementation so far are described in this report. Hopefully, the implementation can some day be completed, and AM2SGML will prove to be a valuable addition to AM2.
2 Specifications

This chapter describes the user interface for the ADL classes which make up AM2SGML. For a description of AM2 and the ADL, please consult the MIT AthenaMuse Software Consortium’s *AthenaMuse 2.1 Documentation*.¹

2.1 Purpose

The classes defined below represent two types of entities - DTDs, and files written in marked-up text defined by a DTD, which we will call document instances. There is a group of classes defined for each of these entities. The top-level class `MMsgmlDocument` represents an SGML document instance, while `MMdtd` represents a DTD. A sample DTD has been defined in Appendix A. This DTD represents a subset of the definition of HTML.²

2.2 Class Specifications

The specifications of the DTD classes and the SGML document instance classes are given below. There are a few things to note. First, unless a class is described as immutable, the programmer can set any members of the class at any time, unless otherwise specified. If a class is immutable, then once an object of that class is created, it cannot be changed. As such, members of immutable classes can be set only when an object of that class is created. Also, all class members are readable.

---

² Note that this DTD may contain inconsistencies with actual HTML.
Second, none of the classes have any activities. An activity is the way that the ADL programmer can handle events generated by user actions, clock ticks, incoming messages, or other such outside stimuli. For example, the ADL programmer could specify that a certain function be called whenever the user clicks the right mouse button in a certain region of the screen. Due to the nature of the AM2SGML classes, none of them need to handle events.

Note also that most of the classes have an element hMark. In SGML, a certain section of text can be marked with the status keywords \texttt{INCLUDE} or \texttt{IGNORE}. These keywords are used when a document contains sections which should sometimes be ignored, and at other times be included. For example, assume that we had a textbook expressed as an SGML document. Also assume that we want to print the student’s version and the teacher’s version of the textbook from the same document. A part of the SGML file might look like the following.

Problem 69.
What is the Capital of Louisiana?

\begin{verbatim}
<![INCLUDE [Baton Rouge]]>
\end{verbatim}

In the teacher’s version of the textbook, we want the answer to the question to be printed. Therefore, we use the SGML fragment given above, and since \texttt{Baton Rouge} is marked with \texttt{INCLUDE}, it will be printed. In the student’s version of the textbook, we do not want the answer to be there. Therefore, before printing the student’s version, we would change \texttt{INCLUDE} to \texttt{IGNORE} in the above fragment.

A marked section looks as follows:

\begin{verbatim}
<! [ status-keyword [ marked-section ] ]>
\end{verbatim}

\begin{flushright}
\end{flushright}
Here, *status-keyword* is **INCLUDE**, **IGNORE**, or a parameter entity which represents **INCLUDE** or **IGNORE**. If a class represents an item that is inside a section marked **INCLUDE**, then hMark is a handle to an MMsgmlSectionMark object representing “include.” If a class represents an item that is inside a section marked **IGNORE**, then hMark is a handle to an MMsgmlSectionMark object representing “ignore.” If a class represents an item that is inside a section marked with a parameter entity, then hMark is a handle to an MMsgmlSectionMark object representing this parameter entity. If an item is not in a marked section, then the object which represents it has hMark equal to an MMsgmlSectionMark object signifying “no mark.”

Table 1 gives an listing of all the AM2SGML classes. It also gives a brief description of each, and tells where each is described in this chapter.

**TABLE 1. The AM2SGML Classes**

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Where Described</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMdtd</td>
<td>Represents an entire DTD. This can contain comments (MMsgmlComment objects) or DOCTYPE definitions (MMdtdDocumentType objects).</td>
<td>Section 2.2.1.1 on page 20</td>
</tr>
<tr>
<td>MMdtdDocumentType</td>
<td>Represents a DOCTYPE definition in a DTD. This can contain entities (MMdtdEntity object), notations (MMdtdNotation objects), attribute lists (MMdtdAttribute objects), element declarations (MMdtdElement objects), and comments (MMsgmlComment objects).</td>
<td>Section 2.2.1.2 on page 22</td>
</tr>
<tr>
<td>MMdtdElement</td>
<td>Represents a DTD ELEMENT declaration. It includes a content model, represented by an MMdtdCommentModel object.</td>
<td>Section 2.2.1.3 on page 26</td>
</tr>
<tr>
<td>MMdtdContentModel</td>
<td>Represents a content model for an ELEMENT declaration.</td>
<td>Section 2.2.1.4 on page 29</td>
</tr>
<tr>
<td>MMdtdAttribute</td>
<td>Represents one attribute of an ELEMENT declaration. Note that each MMdtdAttribute object represents one item in an ATTLIST declaration.</td>
<td>Section 2.2.1.5 on page 32</td>
</tr>
<tr>
<td>MMdtdEntity</td>
<td>Represents an ENTITY declaration in a DTD.</td>
<td>Section 2.2.1.6 on page 36</td>
</tr>
<tr>
<td>MMdtdNotation</td>
<td>Represents a DTD NOTATION declaration.</td>
<td>Section 2.2.1.7 on page 38</td>
</tr>
</tbody>
</table>
TABLE 1. The AM2SGML Classes

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Where Described</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMsgmlDocument</td>
<td>Represents an entire document instance. This corresponds to a DTD (MMsgmlElement object) which describes the document instance. This contains a top-level element (MMsgmlElement object).</td>
<td>Section 2.2.2.1 on page 42</td>
</tr>
<tr>
<td>MMsgmlElement</td>
<td>Represents an element in a document instance. This corresponds to an ELEMENT declaration (MMsgmlElement object) in a DTD. This contains any number of sub-elements and text strings (MMsgmlPCData objects), or an MMsgmlEmpty object, which represents nothing at all.</td>
<td>Section 2.2.2.2 on page 44</td>
</tr>
<tr>
<td>MMsgmlPCData</td>
<td>Represents text data of type #PCDATA in an element.</td>
<td>Section 2.2.2.3 on page 45</td>
</tr>
<tr>
<td>MMsgmlEmpty</td>
<td>Represents the lack of data in a document instance element. In other words, this corresponds to the content model EMPTY.</td>
<td>Section 2.2.2.4 on page 46</td>
</tr>
<tr>
<td>MMsgmlAttribute</td>
<td>This represents an attribute of a document instance element. This corresponds to an item in a DTD ATTLIST declaration (an MMsgmlAttribute object), which states which possible value this attribute can take.</td>
<td>Section 2.2.2.5 on page 47</td>
</tr>
</tbody>
</table>

Classes Used in DTDs or Document Instances

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Where Described</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMsgmlSectionMark</td>
<td>Represents a section mark: INCLUDE, IGNORE, or none.</td>
<td>Page 18; Section 2.2.1.8 on page 40</td>
</tr>
<tr>
<td>MMsgmlComment</td>
<td>Represents a comment in a DTD or document instance.</td>
<td>Section 2.2.1.9 on page 41</td>
</tr>
</tbody>
</table>

2.2.1 The DTD Classes

Following are the classes used to describe various components of a DTD. Also described are the classes that can be used in both a DTD or a document instance.

2.2.1.1 The MMsgmlClass

This represents a DTD in its entirety. A DTD consists of one or more concurrent document-type (DOCTYPE) declarations and possibly comments. Therefore, the MMsgmlClass class consists mainly of handles to MMsgmlDocumentType classes, which represent DOCTYPE declarations.

4. Concurrent DOCTYPE definitions are different ways of organizing the same document. This is how one can specify overlapping tags in SGML document instances.
and MMsgmlComment classes, which represent SGML comments.\textsuperscript{5} The specification for MMdtd is given below.

\begin{verbatim}
class MMdtd

Overview

MMdtd is a class of mutable objects which describe a group of concurrent \texttt{DOCTYPE} declarations.

Methods

upon Construct: list els, handle hMarkArg
  \textbf{Requires:} The elements in els are handles to either MMdtdDocumentType objects or MMsgmlComment objects. els must contain at least one MMdtdDocumentType object. hMarkArg is a handle to an MMsgmlSectionMark object.
  \textbf{Effects:} Creates a new MMdtd object containing the \texttt{DOCTYPE} declarations given in els (as well as the comments), and with the marking specified by hMarkArg.

upon ConstructFromFile: handle hFileStream
  \textbf{Requires:} hFileStream is a handle to a readable IOfile object.
  \textbf{Effects:} Creates an MMdtd object, from the DTD specified in the file in hFileStream.

upon ConstructFromStream: handle hWebStream
  \textbf{Requires:} hWebStream is a handle to an IOwebStream subclass.
  \textbf{Effects:} Creates an MMdtd object, from the DTD specified by the stream IOwebStream.

upon ConstructFromString: string s
  \textbf{Effects:} Creates an MMdtd object from string s.

on DocTypes: return list
  \textbf{Effects:} Returns a list of handles to MMdtdDocumentType objects, representing the \texttt{DOCTYPE} declarations in self.

on Comments: return list
  \textbf{Effects:} Returns a possibly empty list of handles to MMsgmlComment objects, representing the comments in self.

on AddDocType: handle hNewDocType
  \textbf{Requires:} hNewDocType is a handle to an MMdtdDocumentType.
  \textbf{Modifies:} self.
  \textbf{Effects:} Adds the document type declaration corresponding to hNewDocType to the "end" of self.

\end{verbatim}

on RemoveDocType: handle hDoc return boolean
   Requires: hDoc is a handle to an MMdtdDocumentType.
   Modifies: self, hDoc.
   Effects: Removes the document type hDoc from the DTD self. If hDoc was not present in self, does nothing and returns FALSE. Otherwise returns TRUE.

on AddComment: handle hNewComment
   Requires: hNewComment is a handle to an MMMsgmlComment.
   Modifies: self.
   Effects: Adds the comment corresponding to hNewComment to the “end” of self.

on RemoveComment: handle hComment return boolean
   Requires: hComment is a handle to an MMMsgmlComment.
   Modifies: self, hComment.
   Effects: Removes the comment hComment from the DTD self. If hComment was not present in self, does nothing and returns FALSE. Otherwise returns TRUE.

on Unparse: return string
   Effects: Returns a string representing self.

Members
The members of the class MMdtd are given in Table 2.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>docTypeDecls</td>
<td>list</td>
<td>A list of handles to MMdtdDocumentType or MMMsgmlComment objects. Each of these represents a concurrent document-type declaration (or a comment); the order of these components in the list is the same as their order in the DTD.</td>
</tr>
<tr>
<td>hMark</td>
<td>handle</td>
<td>A handle to MMMsgmlSectionMark. The marking of the declaration.</td>
</tr>
</tbody>
</table>

Activities
none.

2.2.1.2 The MMdtdDocumentType Class

This represents the information contained within a DOCTYPE declaration in a DTD. A DOCTYPE declaration has the following form:

\[
\text{<!DOCTYPE name [ item* ] > | <!DOCTYPE name [ system "file" ] [ item* ] >}
\]
Each *item* is an entity, notation, attribute list, or element declaration. If *file* is absent, then the document type is defined based on the *items*. Otherwise, it is defined based on the concatenation of the *items* and the declaration in *file*. In this case, there may be no *items*. The specification for MMdtdDocumentType is given below.

---

**class** MMdtdDocumentType

**Overview**

An MMdtdDocumentType is a mutable object which represents a **DOCTYPE** declaration in a DTD.

**Methods**

**upon Construct:** string nameArg, handle hParentArg, list els, handle hMarkArg

*Requires:* The elements in *els* are handles to MMdtdEntity, MMdtdNotation, MMdtdAttribute, MMdtdElement, or MMsgmlComment objects. *els* must contain at least one MMdtdElement object whose name is the same as the argument nameArg. hMarkArg is a handle to an MMsgmlSectionMark object. hParentArg is a handle to an MMdtd.

*Modifies:* hParentArg.

*Effects:* Creates a new MMdtdDocumentType object containing the components given in *els* and a mark corresponding to hMarkArg; if hParentArg is not NULL, then self is treated as if it were declared in the DTD represented by hParentArg, and self is automatically added to hParentArg.

**upon ConstructFromDocType:** handle hBaseDocType, handle hParentArg, list els, handle hMarkArg

*Requires:* hBaseDocType is a handle to an MMdtdDocumentType object. hMarkArg is a handle to an MMsgmlSectionMark object. hParentArg is a handle to an MMdtd object or NULL.

*Modifies:* hThisParent hParentArg.

*Effects:* Creates an MMdtdDocument object, from hBaseDocType plus the elements in *els* and with a mark corresponding to hMarkArg. If hParentArg is not NULL, then self is treated as if it were declared in the DTD represented by hParentArg, and self is automatically added to hParentArg.

---

upon ConstructFromString: string s, handle hContext

Requires: hContext is a handle to an MMdtd object, or NULL.
Modifies: hContext.
Effects: Creates an MMdtdDocumentType object from string s. If hContext is not NULL, then self is treated as if it were declared in the context of the DTD corresponding to hContext.

on AllItems: return list
Effects: Returns a list of handles to all the items in the MMdtdDocumentType, including those declared in a file referenced by a SYSTEM reference. Note that these can include handles to MMdtdEntity, MMdtdNotation, MMdtdAttribute, MMdtdElement, and MMsgmlComment objects.

on Entities: return list
Effects: Returns a list of handles to MMdtdEntity objects, representing the ENTITY declarations in self.

on UnusedEntities: return list
Effects: Returns a list of handles to MMdtdEntity objects, representing the ENTITY declarations in self which are not used. An entity would not be used if there is another entity within the DOCTYPE declaration with the same name.

on Notations: return list
Effects: Returns a list of handles to MMdtdNotation objects, representing the NOTATION declarations in self.

on Atts: return list
Effects: Returns a list of handles to MMdtdAttribute objects, representing the ATTLIST declarations in self.

on Els: return list
Effects: Returns a list of handles to MMdtdElement objects, representing the ELEMENT declarations in self.

on RedundantElNames: return list
Effects: Returns a list of strings, consisting of the names which are used for more than one element.

on BaseEl: return handle
Effects: Returns a handle to an MMdtdElement object, representing the highest-level element in self. This element must have the same name as self.

on Comments: return list
Effects: Returns a possibly empty list of handles to MMsgmlComment objects, representing the comments in self.

7. SYSTEM references are described on page 36.
8. If there are two ENTITY declaration with the same name, the first one is used. This means that if a given name is used for an ENTITY declaration in an explicitly declared entity as well as in an entity referenced with a SYSTEM clause, then the explicitly declared entity is used.
on AddEntity: handle hNewEntity
   Requires: hNewEntity is a handle to an MMdtdEntity.
   Modifies: self.
   Effects: Adds hNewEntity to the “end” of self.

on RemoveEntity: handle hEntity return boolean
   Requires: hEntity is a handle to an MMdtdEntity, and is an entity in the document type self.
   Modifies: self, hEntity
   Effects: Removes the entity hEntity from document type self. If hEntity was not present in self, does nothing and returns FALSE. Otherwise returns TRUE.

on AddNotation: handle hNewNotation
   Requires: hNewNotation is a handle to an MMdtdNotation.
   Modifies: self.
   Effects: Adds hNewNotation to the “end” of self.

on RemoveNotation: handle hNotation return boolean
   Requires: hNotation is a handle to an MMdtdNotation, and is a notation in the document type self.
   Modifies: self, hNotation
   Effects: Removes the notation hNotation from document type self. If hNotation was not present in self, does nothing and returns FALSE. Otherwise returns TRUE.

on AddAtt: handle hNewAtt
   Requires: hNewAtt is a handle to an MMdtdAttribute.
   Modifies: self.
   Effects: Adds hNewAtt to the “end” of self.

on RemoveAtt: handle hAtt return boolean
   Requires: hAtt is a handle to an MMdtdAttribute, and is an attribute in the document type self.
   Modifies: self, hAtt
   Effects: Removes the attribute hAtt from document type self. If hAtt was not present in self, does nothing and returns FALSE. Otherwise returns TRUE.

on AddEl: handle hNewEl
   Requires: hNewEl is a handle to an MMdtdElement.
   Modifies: self.
   Effects: Adds hNewEl to the “end” of self.

on RemoveEl: handle hEl return boolean
   Requires: hEl is a handle to an MMdtdElement, and is an element in the document type self.
   Modifies: self, hEl
   Effects: Removes the element hEl from document type self. If hEl was not present in self, does nothing and returns FALSE. Otherwise returns TRUE.
on AddComment: handle hNewComment
  Requires: hNewComment is a handle to an MMsgmlComment.
  Modifies: self.
  Effects: Adds hNewComment to the "end" of self.

on RemoveComment: handle hComment return boolean
  Requires: hComment is a handle to an MMsgmlComment, and is a comment in the document type self.
  Modifies: self, hComment
  Effects: Removes the comment hComment from document type self. If hComment was not present in self, does nothing and returns FALSE. Otherwise returns TRUE.

on Unparse: return string
  Effects: Returns a string representing self.

Members

The members of the class MMdtdDocumentType are given in Table 3.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the declaration.</td>
</tr>
<tr>
<td>systemName</td>
<td>string</td>
<td>The &quot;reference name&quot; of this MMdtdDocumentType. This is the name that is used if this MMdtdDocumentType is referenced using the SYSTEM clause of another DOCTYPE declaration.</td>
</tr>
<tr>
<td>hSystemRef</td>
<td>handle</td>
<td>A handle to an MMdtdDocumentType. If self contains a SYSTEM reference to another DOCTYPE declaration, then hSystemRef represents this declaration.</td>
</tr>
<tr>
<td>items</td>
<td>list</td>
<td>A list of handles to the elements in the DOCTYPE declaration. This does not include elements referenced by a SYSTEM clause.</td>
</tr>
<tr>
<td>hMark</td>
<td>handle</td>
<td>A handle to MMsgmlSectionMark. The marking of the declaration.</td>
</tr>
<tr>
<td>hParent</td>
<td>handle</td>
<td>A handle to an MMdtd. The DTD object in which this declaration is found.</td>
</tr>
</tbody>
</table>

Activities

none.

2.2.1.3 The MMdtdElement Class

The MMdtdElement class models an ELEMENT declaration in a DTD. An element declaration has the following form: 9

< !ELEMENT  element-type beginning-tag-optional? end-tag-optional? content-model  
[inclusion-list] [exclusion-list] >

Here, element-type is the name of the element. beginning-tag-optional? and end-tag-optional? state whether the beginning and end tags respectively must be included in an element of this type in an SGML document instance. If the beginning tag is required, then beginning-tag-optional? is - (a dash); otherwise, it is O (the letter). end-tag-optional? works the same way. content-model specifies exactly what kinds of elements can be included in element-type, and in what order.

inclusion-list and exclusion-list are exceptions; an inclusion exception is an element that can be included anywhere in the current content model, and an exclusion exception is an element that cannot be included in the content model. The two lists can appear in any order. An inclusion list has the form

+ ( inc-elt₁ | inc-elt₂ | inc-elt₃ | ... | inc-eltₙ )

while an exclusion list has the form

- ( exc-elt₁ | exc-elt₂ | exc-elt₃ | ... | exc-eltₘ )

where the inc-elts and exc-elts are element names.¹⁰

For example, consider the following element declaration (taken from the sample DTD on line 19 on page 69).

< !ELEMENT HTML O O &HTMLContent; +(EM) >

¹⁰Sperberg-McQueen, pp. 13-14.
Both the beginning and end tags are optional. The content model is &HTMLContent, which is an entity representing the text (HEAD, BODY). The inclusion list consists of the element EM, and there is no exclusion list. Consider also the declaration of the the TITLE element (taken from the sample DTD on line 26 on page 69).

```
<!ELEMENT TITLE - - #PCDATA - (EM)>
```

Both the beginning and end tags are required, and the content model is #PCDATA; the exclusion list consists of the element EM. The specification for MMdtdElement is given below.

class MMdtdElement

Overview
An MMdtdElement is a mutable object representing an ELEMENT declaration in a DTD.

Methods
upon Construct: string name, handle hParentArg,
    boolean isBeginOptional, boolean isEndOptional,
    handle hContentModelArg, list includeExs, list excludeExs,
    handle hMarkArg
    
    Requires: hParentArg is a handle to an MMdtdDocumentType object or is NULL. 
    hContentModelArg is a handle to an MMdtdContentModel object, includeExs 
    and excludeExs are lists of handles to MMdtdElement objects. hMarkArg is a handle 
    to an MMsgmlSectionMark object.
    
    Modifies: hParentArg.
    
    Effects: Creates a new MMdtdElement object. The beginning and ending tags of the specified element can be omitted if isBeginOptional and isEndOptional are TRUE, respectively. The element specified can contain the data corresponding to hContentModelArg, and can include the elements in includeExs, but not this in excludeExs. The element declaration has the mark corresponding to hMarkArg. If hParentArg is not NULL, then the element declaration is defined in the context of the document type defined by hParentArg.

upon ConstructFromString: string s, handle hContext
    
    Requires: hContext is a handle to an MMdtd object or NULL.
    
    Modifies: hContext.
    
    Effects: Creates an MMdtdElement object from s. If hContext is not NULL, then self is treated as if it were declared in the context of the DTD corresponding to hContext.
on IsRedundant: return boolean
   Effects: Returns TRUE iff self is defined in a document type in which another element with
            the same name as self is defined.

on Unparse: return string
   Effects: Returns a string representing self.

Members
The members of MMdtdElement are given in Table 4.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isBeginOpt</td>
<td>boole-</td>
<td>TRUE iff the beginning tag of an element defined by self is optional.</td>
</tr>
<tr>
<td>nModel</td>
<td>boole-</td>
<td>TRUE iff the ending tag of an element defined by self is optional.</td>
</tr>
<tr>
<td>hContent</td>
<td>handle</td>
<td>A handle to an MMdtdContentModel object. This specifies exactly what types of data an element defined by self can contain.</td>
</tr>
<tr>
<td>hMark</td>
<td>handle</td>
<td>A handle to MM_MessageSectionMark. The marking of the declaration.</td>
</tr>
<tr>
<td>hParent</td>
<td>handle</td>
<td>A handle to an MMdtdDocumentType, or NULL. The document type in which self is defined, if any.</td>
</tr>
<tr>
<td>inclusionExs</td>
<td>list</td>
<td>A list of elements which can be included anywhere within the content model of self.</td>
</tr>
<tr>
<td>exclusionExs</td>
<td>list</td>
<td>A list of elements which can be included anywhere within the content model of self.</td>
</tr>
<tr>
<td>attrs</td>
<td>list</td>
<td>The list of all attributes defined for self.</td>
</tr>
</tbody>
</table>

Activities
none.

2.2.1.4 The MMdtdContentModel Class

An MMdtdContentModel element represents a content model in a DTD. A content model specifies exactly what type of data can be contained in an element.

There are a couple of "base types" of data. These are denoted by EMPTY (which signifies no data), and #PCDATA, which signifies text.
In addition to these base types, more complex types can be created by combining base types and elements with "operators." The following are the operators defined for content models (note that e can refer to a base type, an element, or a combination thereof).

- $e?$ - This means that data of type $e$ may or may not occur. In the MMdtdContent model class, this operator is called 'optional.
- $e^*$ - This means that any number of instances (including zero) of data of type $e$ may occur. This element is called 'optionalRepeatable.
- $e^+$ - This means that one or more instances of data of type $e$ must occur. This element is called 'requiredRepeatable.
- $e_1, e_2$ - This means that data of type $e_1$ followed by data of type $e_2$ must occur. This element is called 'sequential.
- $e_1 \& e_2$ - This means that data of type $e_1$ and data of type $e_2$ must occur, in any order. This element is called 'and.
- $e_1 \mid e_2$ - This means that either data of type $e_1$ or data of type $e_2$ must occur. This element is called 'or.

In addition to these operators, parentheses may be used for grouping. The specification for MMdtdContentModel is given below.

```java

class MMdtdContentModel

Overview
An MMdtdContent model is an immutable representation of an SGML content object.

Methods
upon ConstructEmpty
    Effects: Creates an MMdtdContentModel corresponding to EMPTY.

upon ConstructPCData
    Effects: Creates an MMdtdContentModel corresponding to #PCDATA.

upon ConstructEl: handle hEl
    Effects: Creates an MMdtdContentModel corresponding to the element hEl.
```

30
upon ConstructComplex: string opArg, list args

**Requires:** args is a list of handles to MMdtdContentModel objects. opArg is 'optional', 'optionalRepeatable', 'requiredRepeatable', 'sequential', 'and', or 'or.'

**Effects:** Creates an MMdtdContentModel object corresponding to the operator opArg applied to the arguments in args. If there are extra arguments in args (e.g. if opArg is 'and, and args has length greater than one) then the extra arguments are ignored.

upon ConstructFromString: string s, handle hContext

**Requires:** hContext is a handle to an MMdtd object, or NULL.

**Effects:** Creates an MMdtdContentModel object from s. If hContext is not NULL, then self is treated as if it were declared in the context of the DTD corresponding to hContext.

on IsEmpty: return boolean

**Effects:** Returns TRUE iff self refers to the `EMPTY` content model.

on IsPCData: return boolean

**Effects:** Returns TRUE iff self refers to the `#PCDATA` content model.

on IsEl: return boolean

**Effects:** Returns TRUE iff self refers to an element.

on IsComplex: return boolean

**Effects:** Returns TRUE iff self refers to a complex content model, formed by an operator and one or more other content models.

on GetEl: return handle

**Requires:** ('IsEl => self) == TRUE.

**Effects:** Returns a handle to an MMdtdElement, the element corresponding to self.

on GetOp: return string

**Requires:** ('IsComplex => self) == TRUE.

**Effects:** Returns the operator of self, as a string. The possible values are the same as those which can be used for the string argument to ConstructComplex.

on GetArgs: return list

**Requires:** ('IsComplex => self) == TRUE.

**Effects:** Returns the arguments of self, as a list of handles to MMdtdContentModel.

on Unparse: return string

**Effects:** Returns a string representing self.

**Members**

The members of MMdtdContentModel are given in Table 5.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelType</td>
<td>string</td>
<td>This is the type of the content model - one of 'empty', 'PCData', 'el, or 'complex.</td>
</tr>
</tbody>
</table>
Activities
none.

For example, in the code in Appendix A, on line 31, the content model for the BODY element is (HR | #PCDATA)*. In order to manually create an MMdtdContentModel object representing this, the following code segment could be used.

```
1 MMdtdContentModel 'ConstructPCData => PDataModel;
2 // Create the object for #PCDATA
3 MMdtdContentModel {'ConstructEl, hHR} => HRModel;
4 // This assumes hHR is the handle to an MMdtdElement
5 // object representing the HR element
6 MMdtdContentModel {'ConstructEl, hP} => PModel;
7 // This assumes hP is the handle to an MMdtdElement
8 // object representing the P element
9 MMdtdContentModel
10 {'ConstructComplex, 'or, {&HRModel, &PDataModel}} =>
11 HROrPCDataModel;
12 // This represents (HR | #PCDATA)
13 MMdtdContentModel
14 {'ConstructComplex, 'or, {&HROrPCDataModel, &PModel}} =>
15 unrepeatedModel;
16 // This represents (HR | #PCDATA | P)
17 MMdtdContentModel {'ConstructComplex, 'optionalRepeatable,
18 {&unrepeatedModel}} => bodyContentModel;
19 // This represents (HR | #PCDATA)*
```

2.2.1.5 The MMdtdAttribute Class

An MMdtdAttribute object represents an attribute of an element in a DTD. In a DTD, a list of attributes can be specified for each element. For example, in the DTD given in Appendix A, the ATTLIST declaration on line 39 describes the attributes for the element P: align and nowrap. There would be a separate MMdtdAttribute object for each of these.

An attribute declaration has the form\(^\text{11}\)

```
<!ATTLIST element-type
```

\(^\text{11} \text{"The SGML Primer."}\)
Here, *element-type* is the element that the attribute corresponds to. Each *attribute-name* is the name of the attribute. Each *attribute-value* describes which values the attribute can take. This can be one of **CDATA, ENTITY, ENTITIES, ID, IDREF, IDREFS, NAME, NAMES, NMTOKEN, NMTOKENS, NOTATION, NUMBER, NUMBERS, NUTOKEN, NUTOKENS**, or a list of enumerated types. Each *default-value* is the default value of the attribute. This can be one of the following:

- Data of the type described by the corresponding *attribute-value*.
- **#IMPLIED**, which means that this attribute does not need to be specified when an instance of the element *element-type* is defined.
- **#REQUIRED**, which means that when an element of type *element-type* is defined, the value of this attribute must be given.
- **#CURRENT**, which means that when an element of type *element-type* is defined, the value of this attribute is the same as the value of the attribute in the most recently defined element of type *element-type*, unless otherwise specified.

For example, in the sample DTD, the attribute align of element *P* can take the values left, center, right, or justify. By default it takes the value left.

The specification for **MMdtdAttribute** is given below.

12. "The SGML Primer."
class MMdtdAttribute

Overview

An MMdtdAttribute is an immutable representation of an attribute of an SGML class.

Methods

upon ConstructType: handle hEl, string nameArg,
  string attTypeArg, string defaultStat, any hDefaultVal

Requires: hEl is a handle to an MMdtdElement. attTypeArg is one of 'cData,
  'entity, 'entities, 'ID, 'IDRef, 'IDRefs, 'name, 'names, 'nmToken,
  'nmTokens, 'notation, 'number, 'numbers, 'nuToken, or 'nuTokens.
  defaultStat is one of 'implied, 'required, 'current, or 'given. If
  defaultStat is 'given, then hDefaultVal is not NULL, and has type correspond-
  ing to the entry in Table 6.

Modifies: hEl.

Effects: Creates an MMdtdAttribute for element hEl. The attribute has name nameArg,
  and has type corresponding to attTypeArg. If defaultStat is 'given, then the
  attribute has the default value represented by hDefaultVal, otherwise its default status is
given by defaultStat.

TABLE 6. Possible Types of hDefaultVal

<table>
<thead>
<tr>
<th>Value of attType (or thisAttType)</th>
<th>Type of value (e.g. hDefaultVal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>'cData</td>
<td>string</td>
</tr>
<tr>
<td>'entity</td>
<td>string</td>
</tr>
<tr>
<td>'entities</td>
<td>list of strings</td>
</tr>
<tr>
<td>'ID</td>
<td>string</td>
</tr>
<tr>
<td>'IDRef</td>
<td>string</td>
</tr>
<tr>
<td>'IDRefs</td>
<td>list of strings</td>
</tr>
<tr>
<td>'name</td>
<td>string</td>
</tr>
<tr>
<td>'names</td>
<td>list of strings</td>
</tr>
<tr>
<td>'nmToken</td>
<td>string</td>
</tr>
<tr>
<td>'nmTokens</td>
<td>list of strings</td>
</tr>
<tr>
<td>'notation</td>
<td>handle to MMdtdNotation</td>
</tr>
<tr>
<td>'number</td>
<td>integer</td>
</tr>
<tr>
<td>'numbers</td>
<td>list of integers</td>
</tr>
<tr>
<td>'nuToken</td>
<td>string</td>
</tr>
<tr>
<td>'nuTokens</td>
<td>list of strings</td>
</tr>
</tbody>
</table>
upon ConstructEnum: handle hEl, string nameArg, list vals, string defaultStat, string defaultVal

**Requires:** hEl is a handle to an MMdtdElement. defaultStat is one of 'implied,' 'required,' 'current,' or 'given.' vals is a list of strings. If defaultStat is 'given,' then defaultVal is not empty, and is equal to an element of vals.

**Modifies:** hEl.

**Effects:** Creates an MMdtdAttribute for element hEl. The attribute has name nameArg, and can have a value of one of the elements of vals. If defaultStat is 'given,' then the attribute has the default value defaultVal, otherwise its default status is given by defaultStat.

upon ConstructFromString: string s, handle hContext

**Requires:** hContext is a handle to an MMdtd object, or NULL.

**Effects:** Creates an MMdtdAttribute object from s. If hContext is not NULL, then self is treated as if it were declared in the context of the DTD corresponding to hContext.

on IsType: return boolean

**Effects:** Returns TRUE iff self is not of an enumerated type.

on IsEnum: return boolean

**Effects:** Returns TRUE iff self is of an enumerated type.

on Unparse: return string

**Effects:** Returns a string representing self.

**Members**

The members of MMdtdContentModel are given in Table 7 on page 35.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hEl</td>
<td>handle</td>
<td>If self is not of an enumerated type and defaultStat is 'given,' then this is the default value of self. It has the type given in Table 6 on page 34.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the attribute.</td>
</tr>
<tr>
<td>attType</td>
<td>string</td>
<td>The type of the attribute; one of the values given in Table 6 on page 34.</td>
</tr>
<tr>
<td>defaultStat</td>
<td>string</td>
<td>One of 'implied,' 'required,' 'current,' or 'given.'</td>
</tr>
<tr>
<td>enumVals</td>
<td>list</td>
<td>If self is an enumerated type, then this is a list of the possible values of self. Otherwise, it should be empty.</td>
</tr>
<tr>
<td>defaultVal</td>
<td>handle</td>
<td>If self is not an enumerated type and defaultStat is 'given,' the default value of self. It has the type given in Table 6 on page 34. If self is an enumerated type, this is a handle to a string, representing the default value of self.</td>
</tr>
</tbody>
</table>
2.2.1.6 The **MMdtdEntity** Class

An **MMdtdEntity** represents an entity in SGML. An entity can be a parameter entity, which means that it can be used in SGML declarations, or not.

An entity has the form:

```xml
<!ENTITY [\%] entity-name [SYSTEM] "replacement-entity-text" [SUBDOC] [NDATA notation-name] >
```

If the percent sign (%) is included, then the entity is a parameter entity.

If **SYSTEM** is not included, then wherever "&entity-name;" appears in a document described by the DTD, it will be replaced by *replacement-entity-text*. If **SYSTEM** is included, then wherever "&entity-name;" appears in a document described by the DTD, it will be replaced by the contents of the file *replacement-entity-text*. If **SUBDOC** is included in addition to **SYSTEM**, then *replacement-entity-text* represents an SGML document instance with its own DTD.

If **NDATA** is included (along with **SYSTEM**), then *notation-name* represents a notation, which in turn represents a method for interpreting a file. This method would be used to interpret the file *replacement-entity-text*. The following is an example of such a declaration:

```xml
<!NOTATION gif SYSTEM "/usr/bin/gifViewer">
<!ENTITY mePict SYSTEM "/mit/jcgentry/me.gif" NDATA gif>
```

---

14. Sperberg-McQueen, p. 22.
15. "The SGML Primer."
16. "The SGML Primer."
17. "The SGML Primer."
Wherever `mePict` was found in a document instance, it would be taken to mean the file
/mit/jcgentry/me.gif. Furthermore, it would be understood that this file can be read by
the program `/usr/bin/gifViewer`.

Examples of entity declarations are given starting on line 8 in the sample DTD in
Appendix A. The specification of the `MMdtdEntity` class is given below.

```plaintext
class MMdtdEntity

Overview
An `MMdtdEntity` object is an immutable representation of an SGML ENTITY declaration.

Methods
upon Construct: string nameArg, handle hParentArg,
    string hValArg, boolean isParameter, boolean isSystem,
    boolean isSubdoc, handle hNotationToUse
Requires: hParentArg is NULL or a pointer to an MMdtdDocumentType object. hNotationToUse is NULL or a pointer to an MMdtdNotation object.
Modifies: hParentArg.
Effects: Creates an MMdtdEntity named nameArg, with value hValArg. If isParameter is TRUE, then it is a parameter entity; if isSystem is TRUE, then it is a system entity, with hValArg representing a file name. If hParentArg is not NULL, then the entity declaration is defined in the context of the document type defined by hParentArg. If isSystem and isSubdoc are both TRUE, then hValArg represents an SGML file with its own DTD. If hNotationToUse is not NULL, then it represents the notation which should be used to interpret the file represented by hValArg.

upon ConstructFromString: string s, handle hContext
Requires: hContext is a handle to an MMdtd object, or NULL.
Effects: Creates an MMdtdAttribute object from s. If hContext is not NULL, then self is treated as if it were declared in the context of the DTD corresponding to hContext.

on Unparse: return string
Effects: Returns a string representation of self.
```
Members

The members of MMDtdEntity are given in Table 8 on page 38.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isParameter</td>
<td>bool-true</td>
<td>TRUE iff self is a parameter entity.</td>
</tr>
<tr>
<td>isSystem</td>
<td>bool-true</td>
<td>TRUE iff self is a system entity.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the entity.</td>
</tr>
<tr>
<td>isSubdoc</td>
<td>bool-true</td>
<td>TRUE iff val represents an SGML file with its own DTD.</td>
</tr>
<tr>
<td>val</td>
<td>string</td>
<td>the value of the entity.</td>
</tr>
<tr>
<td>hNotation</td>
<td>handle</td>
<td>A handle to an MMDtdNotation (or NULL). The notation used to interpret val.</td>
</tr>
</tbody>
</table>

Activities

none

For example, the entity on line 8 on page 69 could be created with

```
MMDtdEntity
{ 'Construct, 'HTMLContent, NULL, "(HEAD, BODY)" TRUE, FALSE, FALSE, NULL} => myEntity;
```

The entity on line 13 on page 69 could be created with

```
MMDtdEntity { 'Construct, 'infoFile, NULL, "/usr/local/html.info", FALSE, TRUE, NULL} => myEntity2;
```

2.2.1.7 The MMDtdNotation Class

The MMDtdNotation class represents an SGML notation. A notation is a technique used to process a type of data which could be included in an SGML document. 18

A notation has the following form: 19

---

18.“The SGML Primer.”
19.“The SGML Primer.”
For example, on line 48 of the sample DTD, a notation called eq is defined, and associated with the file /usr/bin/eqView. This presumably means that equation data can be included in a document instance of this DTD, and that data could be processed using the program /usr/bin/eqView. The specification for MMdtdNotation is given below.

---

class MMdtdNotation

**Overview**

An MMdtdNotation object is an immutable represents a NOTATION declaration in an SGML DTD.

**Methods**

- **upon Construct**: string nameArg, handle hParentArg, string valArg
  - Requires: hParentArg is a handle to an MMdtdDocumentType object, or NULL.
  - Modifies: hParentArg.
  - Effects: Creates an MMdtdNotation with name nameArg and value valArg. If hParentArg is not NULL, then the notation declaration is defined in the context of the document type defined by hParentArg.

- **upon ConstructFromString**: string s, handle hContext
  - Requires: hContext is a handle to an MMdtd object, or NULL.
  - Effects: Creates an MMdtdNotation object from s. If hContext is not NULL, then self is treated as if it were declared in the context of the DTD corresponding to hContext.

- **on Unparse**: return string
  - Effects: Returns a string representation of self.

**Members**

The members of MMdtdNotation are given in Table 9 on page 39.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the notation.</td>
</tr>
<tr>
<td>val</td>
<td>string</td>
<td>The value of the notation.</td>
</tr>
<tr>
<td>hParent</td>
<td>handle</td>
<td>A handle to an MMdtdDocumentType object. The document type in which the notation is declared.</td>
</tr>
</tbody>
</table>
Activities
none.

2.2.1.8 The MMsgmlSectionMark Class

As described on page 18, sections of a DTD or a document instance can be marked. An
MMsgmlSectionMark object represents one of these marks. The specification of MMsgml-
SectionMark is given below.

class MMsgmlSectionMark

Overview
An MMsgmlSectionMark object is an immutable representation of an SGML section mark.

Methods
upon ConstructIgnore
   Effects: Creates a section mark corresponding to IGNORE.

upon ConstructInclude
   Effects: Creates a section mark corresponding to INCLUDE.

upon ConstructEntity: handle hEntity
   Requires: hEntity is a handle to an MMdtdEnt object.
   Effects: Creates a section mark corresponding to the entity represented by hEntity.

upon ConstructNone
   Effects: Creates a section mark corresponding to the absence of a section mark. i.e., if a sec-
tion is marked with this section mark, then it is as if the section is not marked at all. (In
effect, this is the same as marking the section with INCLUDE.)

upon ConstructFromString: string s, handle hContext
   Requires: hContext is a handle to an MMdtd object, or NULL.
   Effects: Creates an MMsgmlSectionMark object from s. If hContext is not NULL, then
   self is treated as if it were declared in the context of the DTD corresponding to hCon-
text.

on IsIgnore: return boolean
   Effects: Returns TRUE iff self corresponds to the IGNORE mark.

on IsInclude: return boolean
   Effects: Returns TRUE iff self corresponds to the INCLUDE mark.
on IsEntity: return boolean
  Effects: Returns TRUE iff self corresponds to an entity.

on Unparse: return string
  Effects: Returns a string representation of self.

Members
The members of MMsgmlSectionMark are given in Table 10.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>markType</td>
<td>string</td>
<td>The type of mark this is. One of 'include', 'ignore', 'none', or 'entity.</td>
</tr>
<tr>
<td>markEntity</td>
<td>handle</td>
<td>If self corresponds to an entity, this is the MMdtdEntity object representing that entity.</td>
</tr>
</tbody>
</table>

Activities
none

2.2.1.9 The MMsgmlComment Class

This class represents a comment in a DTD or in an SGML document instance. Its specification is given below.

class MMsgmlComment

Overview
An MMsgmlComment is an immutable object representing an SGML comment.

Methods
upon Construct: string textArg
  Effects: Creates a comment with text textArg.

upon ConstructFromString: string s
  Effects: Creates an MMsgmlComment from the string s.

on Unparse: return string
  Effects: Returns the string representation of self.
Members

The members of MMsgmlComment are given in Table 11.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>string</td>
<td>The text of the comment.</td>
</tr>
</tbody>
</table>

Activities

none

Note that MMsgmlComment has two similar constructors. Construct takes the body of a comment as its argument. For example,

\[
\text{MMsgmlComment \{\text{Construct, "This is my comment text"}\} => myComment}
\]

The constructor ConstructFromString, on the other hand, takes the comment body plus the comment delimiters, as follows:

\[
\text{MMsgmlComment \{\text{ConstructFromString, "<!-- This is my comment text -->"}\} => myComment}
\]

2.2.2 The Document Instance Classes

The following are the classes which represent an SGML document instance.

2.2.2.1 The MMsgmlDocument Class

This class represents a document instance as a whole. It consists mainly of a top-level element instance. An MMsgmlDocument object should also contain a pointer to an MMDtd which describes the object. The specification for MMsgmlDocument is given below.
class MMsgmlDocument

Overview
An MMsgmlDocument is a mutable representation of an SGML document instance.

Methods
upon Construct: handle hDTDArg, handle hTopEl
   Requires: hDTDArg is a handle to an MMdtd object. hTopEl is a handle to an MMsgml-
   Class object.
   Effects: Creates an MMsgmlDocument with data corresponding to hTopClass.

upon ConstructFromFile: handle hFileStream
   Requires: hFileStream is a handle to a readable IOfile object.
   Effects: Creates an MMsgmlDocument from the data in hFileStream.

upon ConstructFromStream: handle hWebStream
   Requires: hWebStream is a handle to an IOwebStream subclass.
   Effects: Creates an MMsgmlDocument object from the data in hWebStream.

upon ConstructFromString: string s
   Effects: Creates an MMsgmlDocument object from s.

on IsGood: return boolean
   Effects: Returns TRUE iff self is a valid document instance. (This means, in part, that the
   top-level element can be properly described by the DTD).

on Unparse: return string
   Effects: Returns a string representation of self.

Members
The members of MMsgmlDocument are given in Table 12.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDTD</td>
<td>handle</td>
<td>A handle to an MMdtd object. The DTD describing the document instance.</td>
</tr>
<tr>
<td>hTop</td>
<td>handle</td>
<td>A handle to an MMdtdElement. The top-level element.</td>
</tr>
</tbody>
</table>

Activities
none
2.2.2.2 The MMsgmlElement Class

This represents a SGML element instance. It consists of a “list” consisting of items which can include other elements, text data represented by MMsgmlPCData objects, or nothing at all, which is represented by an MMsgmlEmpty object.

An element can also contain a list of attributes. An attribute in an element instance is represented by an MMsgmlAttribute object. The specification for MMsgmlElement is given below.

class MMsgmlElement

Overview

An MMsgmlElement object is a mutable representation of an SGML element instance.

Methods

upon Construct: handle hParentArg, handle hSpec, list itemsArg, list atts, handle hMarkArg

Requires: hParentArg is a handle to an MMsgmlDocument or NULL. hSpec is a handle to an MmdtdElement. itemsArg is a list of handles to either MMsgmlElement, MMsgmlPCData, or MMsgmlEmpty objects. atts is a list of MMsgmlAttribute objects or empty. hMarkArg is a handle to an MMsgmlSectionMark object.

Modifies: hParent.

Effects: Creates an MMsgmlElement object, in the context of hParent if it is given. The element is described by hSpec, consists of items, and has mark hMarkArg. If the element has any attributes specified, they are given in atts.

upon ConstructFromString: string s, handle hContext

Requires: hContext is a handle to an MMsgmlDocument object.

Modifies: hContext.

Effects: Creates an MMsgmlElement from string s, in the context of hContext.

on IsGood: return boolean

Effects: Returns TRUE iff the items in self are properly described by the element specification corresponding to self.

on AttsNotGiven: return list

Effects: Returns a list of MmdtdAttribute objects, which are the attributes for self which are not specified but need to be.

on Unparse: return string

Effects: Returns a string representation of self.
Members

The members of MMsgmlElement are given in Table 13.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>list</td>
<td>A list of handles to MMsgmlElement, MMsgmlPCData, and MMsgmlEmpty objects. These are the data which make up the element.</td>
</tr>
<tr>
<td>allAtts</td>
<td>list</td>
<td>A list of handles to MMsgmlAttribute objects. These are the values of all the attributes for this element.</td>
</tr>
<tr>
<td>hParent</td>
<td>handle</td>
<td>A handle to an MMsgmlDocument. The document in which self is contained.</td>
</tr>
<tr>
<td>hMark</td>
<td>handle</td>
<td>The mark of this element.</td>
</tr>
</tbody>
</table>

Activities

none

2.2.2.3 The MMsgmlPCData Class

This class represents text data. More specifically, it represents text data of the SGML type #PCDATA. Note that this data can include entities. The specification for this class is given below.

class MMsgmlPCData

Overview

An MMsgmlPCData object is a mutable representation of SGML text data.

Methods

upon ConstructFromText: string text
   Effects: Creates an MMsgmlPCData object from text, which is assumed to have no entities.

upon ConstructFromString: string s, handle hContext
   Requires: hContext is a handle to an MMsgmlDocument object.
   Effects: Creates an MMsgmlPCData object from s (which may include entities). It is created in the context hContext.

on AddText: string newText
   Modifies: self.
   Effects: Adds the text newText to the end of self.
on AddEntity: handle hNewEntity
   Require: hNewEntity is a handle to an MMdtdEntity object.
   Modify: self.
   Effect: Adds the entity represented by hNewEntity to the end of self.

on AsText: return string
   Effect: Returns the contents of self, with all entities replaced by the text they represent.

on Entities: return list
   Effect: Returns a list of MMdtdEntity object, representing the entities included in self (in order).

on Unparse: return string
   Effect: Returns a string representation of self.

Members

The members of MMMsg1PCData are given in Table 14.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>list</td>
<td>A list of handles to strings or MMdtdEntity objects. This is the data represented by self.</td>
</tr>
</tbody>
</table>

Activities

none

2.2.2.4 The MMMsg1Empty Class

This class represents the EMPTY data content.

class MMMsg1Empty

Overview

An MMMsg1Empty object is an immutable representation of the SGML EMPTY data construct.

Methods

upon Construct
   Effect: Creates an MMMsg1Empty object.

upon ConstructFromString: string s, handle hContext
   Require: hContext is a handle to an MMMsg1Document object.
   Effect: Creates an MMMsg1Empty object from s (which may include entities). It is created in the context hContext.
on Unparse: return string
    Effects: Returns a string representation of self (i.e., 'EMPTY).

Members
none.

Activities
none.

2.2.2.5 The MMsgmlAttribute Class

This class represents an attribute to an SGML element, with a given value. The value is expressed as a handle. The type of the object to which the handle points depends on what SGML type the attribute has. The possibilities are given in Table 6 on page 34. The specification for MMsgmlAttribute is given below.

class MMsgmlAttribute

Overview
An MMsgmlAttribute is an immutable representation of an instance of an attribute in SGML.

Methods
upon Construct: handle hSpec, handle hValArg
    Requires: hSpec is a handle to an MMdtdAttribute; val has type specified in Table 6 on page 34.
    Effects: Creates an attribute instance with value hValArg for an attribute specified by hSpec.

upon ConstructFromString: string s, handle hValArg
    Requires: hSpec is a handle to an MMdtdAttribute.
    Effects: Creates an MMsgmlAttribute object from s. It is created as an instance of the attribute specification MMdtdAttribute.

on IsGood: return boolean
    Effects: Returns TRUE iff the value of self is an allowable value, given the possible values of the attribute.

on IsDefaultValue: return boolean
    Effects: Returns TRUE iff the value of self is also the default value of self.
on Unparse: return string

**Effects:** Returns a string representation of self.

**Members**

The members of `MMxmlAttribute` are given in Table 15 on page 48.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hSpec</td>
<td>handle</td>
<td>A handle to <code>MMdtdAttribute</code>. The attribute corresponding to self.</td>
</tr>
<tr>
<td>val</td>
<td>handle</td>
<td>The value of self. This has type given in Table 6 on page 34.</td>
</tr>
</tbody>
</table>

**Activities**

none
3 The AM2SGML C++ Classes

A set of C++ classes, which form the basis of the AM2SGML ADL classes, has been partially implemented, and is described below. Once a full set of C++ classes has been written, they can be “wrapped.” This will make them available to the ADL programmer as the ADL classes described in Chapter 2.

Much more work needs to be done on the C++ classes. However, they currently provide a basic level of functionality.

3.1 YASP

A decision was made early on to use a pre-existing SGML parser in the implementation of the AM2SGML C++ classes. There are numerous good SGML parsers available, both commercially and in the public domain, and it would have taken much redundant effort to write a new parser from scratch. For AM2SGML, the Yorktown Advanced SGML Parser (YASP) was used.

3.1.1 Why YASP?

YASP was chosen because it meets two criteria. First, it is in the public domain. Second, it has a well-documented application programming interface (API). YASP was the only parser that could be found which meets both of these criteria. The parsers YAO and SP were also considered; both are in the public domain, but the APIs for both are seriously lacking in documentation. While the YASP documentation is in many ways far from ideal, the document *The SGML Parser Interface: Functional Specification Version 1.25* is much more thorough than the documentation for the other two parsers.¹
3.1.2 An Overview of YASP

YASP is designed to set up an interactive relation between the YASP parser itself, henceforth called the “Parser,” and the client program which is using the parser, henceforth called the “Application.” YASP was designed to be as platform-independent as possible. To accomplish this, the Application performs any platform-specific functions that are necessary. Examples of such functions are allocating memory and accessing files.

The flow of control is passed back and forth between the parser and the Application. The Application calls the Parser main entry point (the PRSMAIN function) when it is ready to begin. The Parser then starts parsing an SGML document. Whenever the Parser needs the Application to perform a platform-dependent service, or whenever the Parser comes to an interesting parsing event, it turns control over to the Application. Examples of interesting parsing events would be the start or end of an element, or the start or end of a DTD. The Application performs the platform-dependent service or acts on the interesting parsing event, and then passes control back to the Parser. Information is passed back and forth between the Parser and the Application in a Parser/Application Communication Structure, or PAC.

3.2 Overview of the AM2SGML C++ Functions

Currently, AM2SGML consists of C++ functions which will read an SGML file, parse the file, and return a structure which represents the overall element structure of the SGML file. The top-level C++ function is parseSGMLFile. Among other parameters, this function takes the name of a text SGML file. parseSGMLFile sets up a PAC structure to be used with the YASP parser, and then calls the main YASP parser entry point, PRSMAIN.

One of the fields of the PAC data structure is called \texttt{ase}. This field contains a pointer to the function that YASP calls whenever it needs a platform-dependent service or whenever YASP reaches an interesting parsing event. \texttt{parseSGMLFile} sets this field to the function \texttt{dispatch}. Another field is \texttt{user}\_\texttt{p}. This is where AM2SGML stores the \textit{application data}. Also stored there is space for the structure that AM2SGML uses to represent the parsed SGML document. This structure has the type \texttt{SGMLDocument}.

Whenever YASP calls \texttt{dispatch}, it passes to it the original PAC structure as an argument. The \texttt{func} field of this structure is set to a code which represents the platform-dependent service that YASP needs performed or the interesting parsing event that it has found. The \texttt{dispatch} function is basically a large \texttt{switch} statement. There are separate functions to deal with each possible platform-dependent function and each possible interesting parsing event; these functions are called the \textit{service functions}. \texttt{dispatch} calls the appropriate service function. Each of these returns 0 if it encounters no error, or an integer between 1 and 10 otherwise. When a service function returns, \texttt{dispatch} takes its return value, and returns it to YASP. Whenever YASP comes to an interesting parsing event and calls the appropriate service function, this service function modifies the \texttt{SGMLDocument} structure stored in the application data, if necessary.

After YASP is finished parsing the entire SGML file, \texttt{PRSMAIN} returns control to the function \texttt{parseSGMLFile}, which called it. By this time, the \texttt{SGMLDocument} structure stored in the user information represents a totally parsed SGML document. \texttt{parseSGMLFile} returns a pointer to this structure. This entire process is described in more detail below.
3.3 AM2SGML Module Descriptions

Each C++ file and class of AM2SGML is described below. The function used to test AM2SGML is described also. All the code for AM2SGML is given in Appendix B.

3.3.1 The Test Function

The test function, given in Section B.1, takes four command line arguments. The first of these is the name of an SGML file to be parsed. The second is the name of a DTD file, which is optional if a DTD declaration is already in the SGML file. The third is the name of an OCS file (which is a parsed SGML declaration), and the fourth is the name of an ODT file (which is a parsed DTD). Only the first argument is necessary. If the second, third, or fourth arguments are not present, they should be replaced by the integer 0.

The test function calls parseSGMLFile, and passes it the names of the files given on the command line. Once parseSGMLFile returns the SGMLDocument object, the test function calls the method SGMLDocument::unparse, which return a text representation of the SGMLObject. This representation is then output to the screen. This representation can be checked against the original SGML file to see if the file was parsed correctly.

3.3.2 The parseSGMLFile Function

This is the top-level AM2SGML function. It sets up the PAC, creates a message handler, creates the structure used for the user information, and calls PRSMAIN. These steps are explained in more detail below. The declaration for this function is given in Section B.3, and it is defined in Section B.4.

First, a new PAC structure called pac is created. The following fields of pac are set:

- id - This is the version of the current pac structure. Its value is the constant PAC_IDENTIFIER, which is stored in one of the included YASP header files.
- **ase** - The function that YASP calls when it needs a platform-dependent service or it encounters an interesting parsing event. This function must take a pointer to a PAC structure as a parameter, and return an int. This field is set to the function dispatch, described in Section 3.3.3.

- **user_p** - This is where application data is kept. The application data is stored in a structure of type AppData. This structure is described below.

After pac is set up, parseSGMLFile creates a message handler. Whenever YASP encounters an error, it informs the Application by referring to an error number. The error numbers and their corresponding error messages are all stored in a file. The message handler is in charge of reading this file. It has a member function which will take an error number and return an error message. The message handler object is described in Section 3.3.6.

After creating the message handler, parseSGMLFile creates a structure for application data of type AppData. This structure is called myData. The important fields of myData are as follows:

- **primaryEntity** - The name of the SGML file being parsed.
- **DTDFile** - The name of the DTD file being used, or NULL if none is given on the command line.
- **dtd** - The name of a DTD object. This is the object that is returned after a DTD is parsed. At the current time, however, AM2SGML does not actually parse DTDs.
- **doc** - The SGMLDocument object that is created during parsing.
- **current** - The current element pointer. This is a pointer to the SGML element that is currently being parsed. This is initially set to NULL.
- **msgHandler** - A pointer to the message handler being used.

After setting up myData, parseSGMLFile is ready to call PRSMAIN. PRSMAIN returns 0 if it encounters no errors. If this is the case, then parseSGMLFile returns a pointer to the SGMLDocument structure created during parsing, which is stored in the doc field of myData. If PRSMAIN returns anything but 0, then parseSGMLFile returns NULL.
3.3.3 The dispatch Function

This is the function that is called by YASP whenever it needs a platform-dependent service or it has an interesting parsing event to report. Its declaration is given in Section B.5 on page 74, and its implementation in Section B.6 on page 75. dispatch takes as an argument pac, which is a pointer to a PAC structure, and it returns an int. YASP expects dispatch to return 0 if it encounters no error, or an integer between 1 and 10 if it encounters an error. Exactly which integer is returned depends on why YASP called dispatch, and on what type of error dispatch encountered.

YASP indicates what platform-dependent service it needs or what kind of interesting parsing event it has encountered by setting the func field of pac. This field has type PFC, which is an enumerated type whose values represent the different service functions. There should be a separate service function for each possible value of the func field. At the moment, the only functions which exist are those which correspond to the more frequently requested platform-dependent services and the more common parsing events.

dispatch looks at the func field of pac, and calls the corresponding service function. It does this by using a switch statement. The service functions which are currently implemented are given in Table 16. Some of the service functions do not actually do anything yet; they simply return 0. The func values for these functions are given in italics.

<table>
<thead>
<tr>
<th>Value of func</th>
<th>Name of Service Function</th>
<th>Description of Service Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELTST_PFC</td>
<td>eltSt</td>
<td>The start of an element</td>
</tr>
<tr>
<td></td>
<td>Section 3.3.5.1 on page 58</td>
<td></td>
</tr>
<tr>
<td>ELTND_PFC</td>
<td>eltNd</td>
<td>The end of an element</td>
</tr>
<tr>
<td></td>
<td>Section 3.3.5.2 on page 59</td>
<td></td>
</tr>
<tr>
<td>RE_PFC</td>
<td>re</td>
<td>Relevant record end (this is when a record is ended unexpectedly by the end of a file)</td>
</tr>
<tr>
<td>Value of func</td>
<td>Name of Service Function</td>
<td>Description of Service Function</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>TEXT_PFC</td>
<td>text</td>
<td>Test data has been encountered</td>
</tr>
<tr>
<td>PI_PFC</td>
<td>pi</td>
<td>A processing instruction has been encountered</td>
</tr>
<tr>
<td>PROST_PFC</td>
<td>proSt</td>
<td>The beginning of the prolog has been encountered. The prolog consists of the SGML declaration and DTD</td>
</tr>
<tr>
<td>PROND_PFC</td>
<td>proNd</td>
<td>The end of the prolog has been encountered</td>
</tr>
<tr>
<td>DOCND_PFC</td>
<td>docNd</td>
<td>The end of a document has been reached</td>
</tr>
<tr>
<td>ETYST_PFC</td>
<td>etySt</td>
<td>The start of an entity has been encountered (an entity in this case is the same as a file)</td>
</tr>
<tr>
<td>ETYND_PFC</td>
<td>etyNd</td>
<td>The end of an entity has been encountered</td>
</tr>
<tr>
<td>SKIP_PFC</td>
<td>skip</td>
<td>For some reason, data in the SGML file has been skipped</td>
</tr>
<tr>
<td>DE CST_PFC</td>
<td>decSt</td>
<td>The start of an SGML declaration has been encountered</td>
</tr>
<tr>
<td>DTDST_PFC</td>
<td>DTDSt</td>
<td>The start of a DTD has been encountered</td>
</tr>
<tr>
<td>DTDND_PFC</td>
<td>DTDNd</td>
<td>The end of a DTD has been encountered</td>
</tr>
<tr>
<td>DAPND_PFC</td>
<td>dapNd</td>
<td>The end of the document has been encountered right after the end of the prolog</td>
</tr>
</tbody>
</table>

**Platform-Dependent Services**

<table>
<thead>
<tr>
<th>Value of func</th>
<th>Name of Service Function</th>
<th>Description of Service Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_PFC</td>
<td>msgS</td>
<td>Print a message</td>
</tr>
<tr>
<td>SGET_PFC</td>
<td>sGet</td>
<td>Get storage</td>
</tr>
<tr>
<td>SFRE_PFC</td>
<td>sFre</td>
<td>Free storage</td>
</tr>
<tr>
<td>FIND_PFC</td>
<td>find</td>
<td>Create a file</td>
</tr>
<tr>
<td>XENOP_PFC</td>
<td>xEnOp</td>
<td>Open a text file</td>
</tr>
<tr>
<td>UTOPR_PFC</td>
<td>utOpR</td>
<td>Open a binary (utility) file for reading</td>
</tr>
<tr>
<td>UTOPW_PFC</td>
<td>urOpW</td>
<td>Open a binary file for writing</td>
</tr>
<tr>
<td>UTWRT_PFC</td>
<td>utWrt</td>
<td>Write to a binary file</td>
</tr>
<tr>
<td>UTRD_PFC</td>
<td>utRd</td>
<td>Read from a binary file</td>
</tr>
</tbody>
</table>

Section 3.3.5.3 on page 59
Section 3.3.5.13 on page 64
Section 3.3.5.14 on page 65
Section 3.3.5.15 on page 65
Section 3.3.5.16 on page 65
Section 3.3.5.10 on page 63
### TABLE 16. The Service Functions

<table>
<thead>
<tr>
<th>Value of func</th>
<th>Name of Service Function</th>
<th>Description of Service Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTCL_PFC</td>
<td>utCl</td>
<td>Close a binary file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 3.3.5.11 on page 64</td>
</tr>
<tr>
<td>XENRD_PFC</td>
<td>xEnRd</td>
<td>Read from a text file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 3.3.5.12 on page 64</td>
</tr>
<tr>
<td>XENCL_PFC</td>
<td>xEnCl</td>
<td>Close a text file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 3.3.5.11 on page 64</td>
</tr>
</tbody>
</table>

#### 3.3.4 The SGMLObjec t Class and its Subclasses

The SGMLObjec t class is intended to be a virtual base class for classes representing SGML objects. At the present time, it is a virtual base class for the classes SGMLDocument, SGMLElement, and DTD. (Currently, DTD does not do anything; its declaration is given in Section B.13 on page 81, and its trivial implementation is given in Section B.14 on page 81.) The declaration of SGMLObjec t is given in Section B.7 on page 77, and the implementation is given in Section 1 on page 77.

The SGMLObjec t class handles error functions for its child classes. Any type of SGML object can have an error state. If any sort of error occurs during the parsing of an SGML object, then the error state for that object is set to something other than noError. At the present time, AM2SGML is not extremely robust, and it does not make much use of error states. The child classes of SGMLObjec t are described below.

#### 3.3.4.1 The SGMLDocument Class

An SGMLDocument object represents an entire SGML document instance. An SGMLDocument has a top-level element, which is the element that shares the name of the document instance. The member function setTopEl sets the top element of the SGMLDocument, and the member function getTopEl returns a pointer to the top element. The unparse member function returns a string representation of the document instance. At present
time, this consists of the string <Document>, a string representation of the top-level element, and the string </Document>. The declaration of SGMLDocument is given in Section B.9 on page 78, and its implementation in Section B.10 on page 78.

### 3.3.4.2 The SGMLElement Class

The SGMLElement class represents an element. Like SGMLDocument, it inherits from SGMLObject. In SGML, an element can consist of other elements, which in AM2SGML are called containees; text data, of type #PCDATA; or nothing at all., which in SGML has type EMPTY. In the future, objects representing #PCDATA and EMPTY should also be considered containees. But at the present time, #PCDATA and EMPTY within an element are ignored.

An SGMLElement has a name. Pointers to the containees of an SGMLElement are stored in the member array containees. The member function addContainee adds a containee to this list, and the member function getNContainees returns the number of containees in this list. SGMLElement also has a member function unparse.

If the name of an SGMLElement is elt, then unparse returns a string consisting of the following:

- The string <elt>
- Strings representing any elements contained in elt. Note that this does not include text data or EMPTY.
- The string </elt>

For example, if elt consisted of the elements elt2 and elt3, and if elt2 in turn consisted of the element elt4, than the string obtained by unparsing elt would be similar to the following:

<elt> <elt2> <elt4> </elt4> </elt2> <elt3> </elt3> </elt>
The declaration of SGMLElement is given in Section B.11 on page 78, and its implementation is given in Section B.12 on page 80.

### 3.3.5 The Service Functions

There are 28 service functions currently implemented. The declarations and implementations of all of them are given in Section B.15 on page 81 through Section B.55 on page 95. The more important ones are described below.

#### 3.3.5.1 The eltSt Function

The eltSt function, declared in Section B.15 on page 81 and defined in Section B.16 on page 81, is called whenever YASP encounters the beginning of an element in a document instance. YASP sets the dad field of the PAC structure to a pointer to a structure representing an element description called a document element descriptor (DED). At the moment, AM2SGML only looks at the part of the DED which represents the name of the element.

The first thing that eltSt does is to get the application data from the PAC; it sets myData to point to it. Second, eltSt gets the DED structure from the PAC, and points ded to it. It then creates a new SGMLElement object, pointed to by newEl, which represents this new element. It figures out the name of the element from the DED (it is stored in ded->elt_p->name.p), and calls the function newEl->setName to set the name of the new element. (In the future, it would be useful to have a constructor for SGMLElement which takes the element’s name as an argument.)

At this point, there are two possible situations. The first is that the new element that YASP encountered is the top-level element. If this is the case, then YASP calls myData->doc->setTopEl(newEl), which sets the top-level element of the object
representing the document we are parsing to this new element. It then sets the current element
pointer, myData->current, to the new element.

The second situation is that the new element newEl is not the top-level element. In this
case, newEl must be a containee of the current element. eltSt therefore calls
myData->current->addContainee(newEl), which adds this new element to the list of
containees of the current element. (addContainee also sets the container of newEl to be the
current element.) eltSt then sets the current element pointer to the new element. As soon as the
end of the new element is encountered (that is, when YASP calls the eltNd service function),
then the current element pointer will be set back to its previous value. The eltNd function is
described in Section 3.3.5.2. The eltSt function returns 0 unless there is a problem calling
SGMLElement::addContainee. In this case it returns the value 10.

3.3.5.2 The eltNd Function

YASP calls this service function when it encounters the end of an element. eltNd sets the
current element pointer to the container of itself, which is found by calling
myData->current->getContainer(). eltNd always returns 0, as there is no way for it
to fail. The declaration of eltNd is given in Section B.17 on page 82, and its implementation is
given in Section B.18 on page 82.

3.3.5.3 The text Function

This function currently does nothing. YASP calls it whenever it encounters text data within
an element. A new class should be implemented to represent SGML text data. This new class
could inherit from SGMLElement. Then, when text was called, an instance of this class would
be created and added to the current element much in the same fashion that a containee element would be. The trivial current implementation of \texttt{text} is given in Section B.20 on page 83.

3.3.5.4 The \texttt{msg} Function

This service function is called whenever YASP wants to send a message. This happens when it encounters some sort of error, or when it has some kind of warning to give. Ideally, AM2SGML would look at the content of these messages, and act accordingly. In the current implementation, however, AM2SGML simply prints these messages to \texttt{stderr}. YASP usually sends a number of messages in a row; a group of messages is called a \textit{suite}.

Before AM2SGML calls the YASP entry point \texttt{PRSMAIN}, it creates a \texttt{MessageHandler} object. This object reads and stores all the possible messages and their numbers from a file. A pointer to the message handler is stored in the application data, in the field \texttt{msgHandler}.

The YASP message file is actually a series of templates. These templates contain strings such as \%1 and \%2, which are supposed to be replaced by actual data when a message is sent.

When YASP wants to send a message, it gives AM2SGML the following:

- A list of data, to be "plugged in" to the message template. The first element in this list replaces \%0, the second replaces \%1, and so forth. This list is stored in the \texttt{dad} field of the PAC.
- The severity of this suite of messages. This is represented by an integer, which is given in the \texttt{errsev} field of the PAC. The possible error severities are given in Table 17.

<table>
<thead>
<tr>
<th>Number</th>
<th>Type of Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Informational message</td>
</tr>
<tr>
<td>4</td>
<td>Warning</td>
</tr>
<tr>
<td>8</td>
<td>Error</td>
</tr>
<tr>
<td>12</td>
<td>Severe error</td>
</tr>
<tr>
<td>16</td>
<td>Terminating error</td>
</tr>
</tbody>
</table>

- The message identifier. This is the integer that the message handler uses to find the proper message template. This is stored in the \texttt{msgcod} field of the PAC.
• Whether or not this is the last message in the suite. If it is, the msg_last field of the PAC is set to true; otherwise, it is set to false.

msg calls the getInstantiatedMessage method of the message handler, giving it the list of data and the message identifier. The method returns a string, which is the message with the data plugged in. msg then prints this string. If the severity of the message is above a certain value called autoQuit (which is defined in msg.h), then msg returns 10. Otherwise, it returns 0. The declaration for msg is given in Section B.22 on page 83, and its implementation in Section B.23 on page 83.

3.3.5.5 The sGet and sFre Functions

This service function is called by YASP when it needs storage. The amount of storage that YASP needs if given in a structure called an ADLEN, found in the stg field of the PAC. An ADLEN contains a pointer and an integer. The integer in this case is the amount of storage (in bytes) requested. Once storage is allocated, it is returned in the pointer of the same ADLEN structure. If the memory was successfully allocated, sGet returns 0. Otherwise, it returns 10. sFre frees the storage in the stg field of the PAC. It always returns 0. The code for sGet and sFre is given in Section B.24 on page 85 through Section B.26 on page 85.

3.3.5.6 The find Function

The find function is called by YASP whenever it needs to access a file. The Application is supposed to create a FileID and return it to YASP. Whenever YASP needs to read from or write to the file, or when it wants to open or close the file, it gives the Application the FileID. AM2SGML has a class called File, which it uses for a FileID. The File class is described in Section 3.3.7 on page 66. When it calls find, YASP sets the asgn_c1s of a structure called an EXID, which
is given in the clad field of the PAC, to a code which represents which type of file YASP needs to access. The different types of files are called file classes (not to be confused with C++ classes).

The file classes which are currently handled by AM2SGML are given below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPACITY_XFC</td>
<td>These represent capacity sets, notation data, and concrete syntaxes, respectively. These are all parts of an SGML declaration. Here, YASP gives the Application a code which represents a specific file. The code is stored in the clad field of the PAC. AM2SGML has a function pubIDFileName (given in Section B.64 and Section B.65) which takes this code and returns the name of a file.</td>
</tr>
<tr>
<td>NOTATION_XFC</td>
<td></td>
</tr>
<tr>
<td>SYNTAX_XFC</td>
<td></td>
</tr>
<tr>
<td>CHARSET_XFC</td>
<td>This represents character data. Here, YASP gives the Application a string representing an SGML character set, and the Application returns a code representing that character set. AM2SGML uses the function charsetID (given in Section B.66 and Section B.67) to determine these codes.</td>
</tr>
<tr>
<td>UTODTW_XFC</td>
<td>As YASP parses a DTD, it outputs a compiled form of the DTD. Here, YASP is requesting a binary file to which it can write the compiled DTD. It is up to the Application to determine the name of this file. AM2SGML takes the name of the SGML file being parsed, and replaces any suffix with odt.</td>
</tr>
<tr>
<td>UTSYNU_XFC</td>
<td>YASP similarly parses SMGL declarations. Here, YASP is requesting a binary file to which it can write the compiled declaration. AM2SGML uses the name of the SGML file, replacing its suffix with ocs.</td>
</tr>
<tr>
<td>UTSYNL_XFC</td>
<td>This means that YASP is parsing an SGML file without a declaration, so it is requesting a file which contains a parsed declaration. AM2SGML assumes that if a parsed declaration is needed, then the name of a file containing one was given on the command line.</td>
</tr>
<tr>
<td>PRIMARY_XFC</td>
<td>This represents the SGML file to be parsed.</td>
</tr>
</tbody>
</table>

The declaration for find is given in Section B.27 on page 85. The implementation is in Section B.28 on page 86.

3.3.5.7 The xEnOp Function

This is the function that YASP calls when it needs to open a text file for reading. It sets the ahl field of the PAC to a FileID that it has previously requested. Since in AM2SGML FileIDs are simply File objects, xEnOp only needs to call the File: : openRead method. If the file is successfully opened, xEnOp returns 0. Otherwise, it returns 6. The declaration for xEnOp is given in Section B.29 on page 88. The implementation is in Section B.30 on page 88.
3.3.5.8 The \texttt{utOpR} and \texttt{utOpW} Functions

YASP calls these service functions when it needs to open a binary file for reading or writing, respectively. The AM2SGML File class does not differentiate between binary and text files (at least where opening and closing them is concerned), so \texttt{utOpR} just calls \texttt{xEnOp}. \texttt{utOpW} simply gets the \texttt{File} object from the \texttt{ahl} field of the PAC, makes sure it is not already open, and if not calls its \texttt{File::openWrite} method. If the file was already opened, \texttt{utOpW} returns 4. If the file is not successfully opened, \texttt{utOpW} returns 6. Otherwise, it returns 0. The code for \texttt{utOpR} and \texttt{utOpW} is given in Section B.31 on page 88 through Section B.33 on page 89.

3.3.5.9 The \texttt{utWrt} Function

This function is used by YASP to write to a binary file. The \texttt{File} object is stored in the \texttt{ahl} field of the PAC, and the data to be written is stored in the \texttt{dad} field of the PAC, in an \texttt{ADLEN} structure. The pointer in the \texttt{ADLEN} structure points to the actual data, and the integer in the \texttt{ADLEN} structure is set to the length of the data. AM2SGML uses the \texttt{File::put} method, which also takes a pointer and an integer, to write the data. The declaration for \texttt{utWrt} is given in Section B.34 on page 89, and its implementation in Section B.35 on page 89.

3.3.5.10 The \texttt{utRd} Function

This function is used to read from a binary file. Once again, the FileID is stored in the \texttt{ahl} field of the PAC, and the \texttt{dad} field contains an \texttt{ADLEN} structure. The integer part of the \texttt{ADLEN} structure is the amount of data that is to be read. Once the data is read, it is put into the pointer part of the \texttt{ADLEN} structure. \texttt{utRd} uses the \texttt{File::readLen} method to read from the file. It returns 0 if successful, and 8 if the file was not opened for reading, or if there was a problem reading from the file. The declaration for \texttt{utRd} is given in Section B.36 on page 90, and its implementation in Section B.37 on page 90.
3.3.5.11 The utCl and xEnCl Functions

These functions do exactly the same thing - close files. As the File object opens and closes text and binary files the same way, there is no difference between these functions. In fact, all utCl does is place a call to xEnCl. xEnCl simply gets the File object from the ahl field of the PAC, and calls its close method. These functions always return 0. The code for utCl is given in Section B.38 on page 90. The code for xEnCl is given in Section B.41 on page 92 and Section B.42 on page 92.

3.3.5.12 The xEnRd Function

This function is used by YASP to read data from a text file, given in the ahl field of the PAC. YASP does not care how much data is read. xEnRd simply calls the File::get method, which reads one line of data, and puts the line that was read in the dad field of the PAC. xEnRd returns 8 if the file is not open for reading, 4 if the file reaches the end, and 0 if everything goes okay. The code for xEnRd is given in Section B.39 on page 91 and Section B.40 on page 91.

3.3.5.13 The proSt Function

YASP calls this function when it reaches the start of the SGML prolog. The SGML prolog consists of the SGML declaration and/or the DTD. It is at this time that AM2SGML creates a new SGMLDocument object, which represents the document instance. This object is stored in the doc field of the application data. (Since an SGMLDocument represents only a document instance, and a document instance does not include DTDs or declarations, the new SGMLDocument object should probably be created once the end of the prolog is reached, when YASP calls proNd, as opposed to when the start of the prolog is reached.) The declaration for proSt is given in Section B.43 on page 92, and its implementation in Section B.44 on page 92.
3.3.5.14 The docNd Function

docNd is called when the end of the document is reached. Currently, this function does nothing, but in future versions of AM2SGML more “clean-up” work might be performed here. The current implementation is given in Section B.46 on page 93.

3.3.5.15 The DTDSt Function

This service function is called by YASP when it reaches the start of a DTD. When this happens, AM2SGML creates a new DTD object to represent the DTD. As explained before, the DTD object currently does not do anything. DTDSt does cause one to be created, however. The DTD object is stored in the dtd field of the application data structure. The declaration for DTDSt is given in Section B.52 on page 94 and its implementation in Section B.53 on page 94.

3.3.5.16 The unimplFn Function

This function is called by dispatch when it receives a request from YASP to perform a service function that is not yet implemented. In the final version of AM2SGML, all the service functions should obviously be implemented, and this function will be unnecessary.

unimplFn simply outputs an error message to the screen, telling the number of the service function that was requested. unimplFn always returns 10, since after it has been called, the parsing of the SGML file cannot go on. unimplFn uses the notify function to send the error message to the screen. Code for unimplFn is given in Section B.56 on page 95.

3.3.6 The MessageHandler Object

The MessageHandler object is responsible for reading the YASP message template file, remembering the templates, and then instantiating a template with a variable list when necessary. The specification of MessageHandler is given in Section B.58 on page 95, and its
implementation is given in Section B.59 on page 96. A MessageHandler can be “good” or “bad” - if there is a problem during creation (for example, when trying to read the template file) then a “bad” message handler is created. A MessageHandler has the unary operator ! which returns true if it is bad. The MessageHandler class has methods getMsg and getID which return the message template and a message ID string given a message number, as well as the method getInstantiatedMsg which takes an array of strings and a message number, and plugs the strings into the template corresponding to the message number.

3.3.7 The File Class

The File class is used for the YASP FileID. YASP needs a way to create a file object with a name before that file is opened; neither the C FILE type nor the C++ file stream classes seemed to be able to provide this functionality.

A File can be opened in three modes - read, write, and append (which is not yet implemented). There are simple methods for doing various types of reads and writes to a file. The File class is implemented using the C FILE type. The declaration for the File class is given in Section B.60 on page 99; its implementation is in Section B.61 on page 101.
Quite obviously, there remains much work to be done on the implementation of AM2SGML. Many more C++ functions and classes need to be written. Roughly, there should be a C++ class for every ADL class described in specifications in Chapter 2. Also, the ADL classes themselves need to be written. This is accomplished by a process called wrapping, which makes certain parts of C++ classes visible to the ADL programmer as ADL classes.

Classes which can parse SGML files will prove to be a useful addition to AM2. They will provide an easy, standardized way to store text information. Almost any AM2 application which needs to store or retrieve text data will be able to take advantage of AM2SGML.

There are currently a few projects being developed at CECI which could use SGML. One of these is called Operación Futuro. One part of this application is an “electronic workbook.” This is supposed to be similar to a regular textbook, except for the fact that it is entirely on line. The workbook is currently implemented as a group of HTML pages. However, it has become clear that HTML does not allow for much flexibility in the way that these workbook pages are designed. If SGML were used, a more specialized DTD for the workbook could be created. However, there still would need to be a way to display the workbook pages once they are parsed.

The ability to parse SGML becomes much more useful when there is a way to use the resultant structures to display the content of the SGML. There currently exist a few standards for specifying the way SGML documents should be displayed; two of the more common ones are the Output Specification (OS), developed by the United States Department of Defense, and the Docu-
ment Style Semantics and Specification Language (DSSSL). It would be useful to examine these standards, to see if they could be used with AM2.

AM2 is already able to manipulate and display an impressive array of media types, including images, movies, and sounds. With the development of AM2's SGML capabilities, text data would be added to this list. The ability to process different types of marked-up coupled with the ability to display them in a customized manner would prove to be one of AM2's more powerful features.

1. "Getting Started."
Appendix A  A Sample DTD

The following DTD specifies a (quite small) subset of HTML. Note that in this DTD, SGML “keywords” are in **bold**, and comments are in *italics*. Parts of this DTD are based on Dave Raggett’s HTML DTD.¹

```
<!DOCTYPE HTML [            <!-- Typical HTML tags -->
  <!ENTITY % HTMLContent "(HEAD, BODY)">
  <!-- Wherever "%HTMLContent;" appears in this DTD (or in a document instance described by it), it will be replaced with "(HEAD, BODY)" -->
  <!ENTITY infoFile SYSTEM "/usr/local/html.info"  <!-- Wherever "&infoFile;" appears in a document instance described by this DTD, it will be replaced with the contents of the file "/usr/local/html.info" -->
  <!ELEMENT HTML 0 0 &HTMLContent; +(EM)>  <!-- The beginning and end tags for an "HTML" section are optional. We want emphasized text to be able to appear anywhere (except for in a title) -->
  <!ELEMENT HEAD 0 0 TITLE>  <!-- The beginning and end tags for a title are necessary. We do not want to allow emphasized text in the title -->
  <!ELEMENT BODY 0 0 (HR | P | #PCDATA)>  <!-- The body of an HTML document can consist of any number of horizontal lines (HR), paragraphs (P), or units of plain text (#PCDATA) -->
  <!ELEMENT EM - - #PCDATA>  <!-- Emphasized text -->
  <!ELEMENT HR - O EMPTY>  <!-- A horizontal line -->
  <!ELEMENT P - O #PCDATA>  <!-- A paragraph -->
  <!ATTLIST P >
```

40    align (left | center | right | justify) left
41    -- The alignment of the paragraph --
42    nowrap (nowrap) #IMPLIED -- Disable word wrap --
43 >
44 ]>
45
46
47 <!DOCTYPE p.HTML [<!-- An (unusual) page structure for HTML -->
48   <!NOTATION eq SYSTEM "/usr/bin/eqView">
49   <!-- Equations might appear in the
50       document... these can be processed
51       with "/usr/bin/eqView" -->
52   <!ELEMENT page - O #PCDATA> <!-- This specifies a "page" -->
53   <!ATTLIST page pageNumber NUMBER #REQUIRED>
Appendix B  Code for AM2SGML

On the following pages is all of the C++ code for AM2SGML, including the program used to test the AM2SGML functions. The makefile that was used is given in Section B.70 on page 107. It assumes that all the files are in the same directory, and that the YASP library yasp.a is in the directory stored in the variable libDir. A sample run of the overall program is given in Appendix C.
B.1 test.cc

```cpp
1 #include <iostream.h>
2 #include "AM2YASP.h"
3 #include "parseSGMLFile.h"
4
5 int main(int argc, char** argv)
6 {
7     // The main SGML document structure
8     SGMLDocument* mainDoc;
9
10    // This should take four arguments: The SGML
11    // file to be parsed, a DTD file (as long as the
12    // DTD is not in the main SGML file), possibly
13    // an OCS file, and possibly an ODT file. If we
14    // have less than four arguments, we should give
15    // an error.
16    if (argc < 5) {
17        cerr << "Bad arguments." << endl;
18        return (-1);
19    }
20
21    // Go through the arguments.
22    const char* mainFile = argv[1];
23    const char* DTDFile = stricmp(argv[2], "0") == 0
24        ? (char*) NULL : stringCopy(argv[2]);
25    const char* OCSFile = stricmp(argv[3], "0") == 0
26        ? (char*) NULL : stringCopy(argv[3]);
27    const char* ODTFile = stricmp(argv[4], "0") == 0
28        ? (char*) NULL : stringCopy(argv[4]);
29
30    // Now, parse the file mainFile.
31    mainDoc = parseSGMLFile(mainFile, DTDFile, OCSFile, ODTFile);
32
33    // Now, unparsing the object mainDoc.
34    cerr << "----------" << endl;
35    char* unparsed = mainDoc->unparse();
36    if (mainDoc != (SGMLDocument*) NULL)
37        cerr << unparsed << endl;
38    else
39        cerr << "(NULL)" << endl;
40    cerr << "----------" << endl;
41    return (0);
```

B.2 AM2YASP.h

```cpp
1 // AM2YASP.h
2 // Jamie Gentry
3
4 // This is printed at the end of each series of
5 // YASP error messages
6 const char* msgEnd = "----------\n";
```

B.3 parseSGMLFile.h

```cpp
1 // parseSGMLFile.h
2 // Jamie Gentry
3
4 #include "DTD.h"
5 #include "SGMLDocument.h"
6 #include "SGMLElement.h"
7 #include "MessageHandler.h"
8
9 // Data structure for application data, passed
10 // back and forth in the user_p field of the PAC
11 struct AppData {
12    // The DTD for this file
13    DTD* dtd;
14```
// The document instance for this file
SGMLDocument* doc;

// The element we are currently looking at
SGMLElement* current;

// The file name of the primary entity
char* primaryEntity;

// The file name of the DTD
char* DTDFile;

// The OCS file name
char* OCSFile;

// The ODT file name
char* ODTFile;

// The message handler
MessageHandler* msgHandler;

SGMLDocument* parseSGMLFile(const char* fileName,
const char* DTDFileName, const char* OCSFileName,
const char* ODTFileName);

// Requires: fileName is a string
// Modifies: dtd, doc

// Effects: Tries to parse the file fileName as an
// SGML document. If successful, returns
// an SGMLDocument representing the file
// fileName. This returned value can be
// destroyed using delete.
// If any of DTDFileName, OCSFileName,
// or ODTFileName are not present, they
// should be set to NULL.
// If there is any error, then NULL is
// returned.

B.4 parseSGMLFile.cc

// parseSGMLFile.cc
// Jamie Gentry

#include <stdio.h>
#include <iostream.h>
#include <portable.h>
#include <yasppi.h>

#include "parseSGMLFile.h"
#include "dispatch.h"
#include "random.h"

extern "C" unsigned int C_FCT PRSMAIN (PAC *);

SGMLDocument* parseSGMLFile(const char* fileName,
const char* DTDFileName, const char* OCSFileName,
const char* ODTFileName);

{ // Set up the YASP PAC structure
  PAC pac;

  // ID
  pac.id = PAC_IDENTIFIER;

  // Reserved
  pac.resv = short(0);

  // Reserved
  pac.argc = short(0);

  // Reserved
  pac.argv = (char**)0;
// Address for app services
pac.ase = dispatch;

// Private app use
pac.user_p = (void*) new AppData;

// Get message handler
// This file name should be moved elsewhere, but for the time being it is easier just to hard code it here
MessageHandler* msgObj = new MessageHandler("/mit/jcgentry/SGML/YASP/parser/msg/yspus.txt");

// See if the message handler was created ok
if (!msgObj)
    return (0);

// Get a pointer to the application data
AppData* myData = (AppData*) pac.user_p;

// Set up application data structure
myData->dt = (DTD*) NULL;
myData->doc = (SGMLDocument*) NULL;

// The element we are currently parsing
myData->current = (SGMLElement*) NULL;

// The name of the SGML file we are parsing
myData->primaryEntity = stringCopy(fileName);

// The name of the OCS file given, or NULL is none is given
myData->OCSFile = ((OCSFileName == (char*) NULL) ?
    (char*) NULL :
    stringCopy(OCSFileName));

// The name of the ODT file, or NULL
myData->ODTFile = ((ODTFileName == (char*) NULL) ?
    (char*) NULL :
    stringCopy(ODTFileName));

// The name of the DTD, or NULL
myData->DTDFile = ((DTDFileName == (char*) NULL) ?
    (char*) NULL :
    stringCopy(DTDFileName));

myData->msgHandler = msgObj;

// Call main parser entry point
unsigned int rc = PRSMAIN(pac);

// To clean up better, we need to see if anything should be deleted here.

// Check return code from parser
if (rc == 0)
    return (myData->doc);
else
    return ((SGMLDocument*) NULL);

B.5 dispatch.h

#include <portable.h>
#include <yaspapi.h>

int dispatch(PAC* pac);

// Requires: Is called from the PRSMAIN function.
// Modifies: pac
// Effects: Performs the function specified (by the Parser) in the pfc field of pac, and returns the required value.

B.6 dispatch.cc

#include <iostream.h>
#include "dispatch.h"
#include "unimplFn.h"
#include "sGet.h"
#include "find.h"
#include "msg.h"
#include "sFre.h"
#include "xEnOp.h"
#include "etySt.h"
#include "xEnRd.h"
#include "decSt.h"
#include "etyNd.h"
#include "xEnCl.h"
#include "utOpW.h"
#include "utWrt.h"
#include "utCl.h"
#include "utOpR.h"
#include "utRd.h"
#include "proSt.h"
#include "DTDSt.h"
#include "DTDNd.h"
#include "proNd.h"
#include "dapNd.h"
#include "eltSt.h"
#include "eltNd.h"
#include "docNd.h"
#include "pi.h"
#include "text.h"

#include "re.h"
#include "skip.h"

int dispatch(PAC* pac)
{
    // Call the proper function
    switch (pac->func) {
    case ELTST_PFC: // 1: Start element
        return (eltSt(*pac));
        break;
    case ELTND_PFC: // 2: End element
        return (eltNd(*pac));
        break;
    case RE_PFC: // 3: Relevant record end
        return (re(*pac));
        break;
    case TEXT_PFC: // 4: Text data
        return (text(*pac));
        break;
    case PI_PFC: // 5: Processing instruction
        return (pi(*pac));
        break;
    case MSG_PFC: // 8: Message
        return (msg(*pac));
        break;
    case SGET_PFC: // 9: Get storage
        return (sGet(*pac));
        break;
    case SFRE_PFC: // 10: Free storage
        return (sFre(*pac));
        break;
    case FIND_PFC: // 11: Get a FileID-AHL
        break;
    }
return (find(*pac));
break;

case XENOP_PFC: // 12: Open external entity
return (xEnOp(*pac));
break;

case UTOPR_PFC: // 13: Open utility file for
    // reading
return (utOpR(*pac));
break;

case UTOPW_PFC: // 14: Open utility file for
    // writing
return (utOpW(*pac));
break;

case UTWRT_PFC: // 15: Write utility file record
return (utWrt(*pac));
break;

case UTRD_PFC: // 16: Read utility file record
return (utRd(*pac));
break;

case UTCL_PFC: // 17: Close a utility file
return (utCl(*pac));
break;

case XENRD_PFC: // 18: Read data from an
    // external entity
return (xEnRd(*pac));
break;

case XENCLO_PFC: // 19: Close an eternal entity
return (xEnCl(*pac));
break;

case PROST_PFC: // 20: Begin prolog
return (proSt(*pac));
break;

case PROND_PFC: // 21: End prolog
return (proNd(*pac));
break;

case DOCND_PFC: // 22: End of DOC found
return (docNd(*pac));
break;

case ETYST_PFC: // 23: Report starting an entity
return (etySt(*pac));
break;

case ETYND_PFC: // 24: Report end of an entity
return (etyNd(*pac));
break;

case SKIP_PFC: // 27: Skipped data
return (skip(*pac));
break;

case DECST_PFC: // 28: Start of an SGML
    // declaration
return (decSt(*pac));
break;

case DTDST_PFC: // 29: Start of DTD
return (DTDS(*pac));
break;

case DTDND_PFC: // 30: End of DTD
return (DTDNd(*pac));
break;

case DAPND_PFC: // 31: End of DOC right after
    // prolog
return (dapNd(*pac));
B.7 SGMLObject.h

1 // SGMLObject.h
2 // Jamie Gentry
3
4 #ifndef SGMLObjectIncl
5 #define SGMLObjectIncl
6
7 const int noError = 0;
8 const int mem = 1;
9 const int syntax = 2;
10
11 class SGMLObject {
12 // An SGMLObject represents any sort of SGML
13 // object. It is mutable
14
15 public:
16 SGMLObject();
17 // Effects: Creates a new SGMLObject
18
19 virtual ~SGMLObject();
20 // Modifies: this
21 // Effects: Destroys this (although does not
22 // destroy any "containees" of this
23
24 inline int isGood() const
25 // Effects: Returns true iff this object is
26 // not in an error state
27 {
28     return (errorCode == noError);
29 }
30
31 void setError(int code);
32 // Modifies: this
33 // Effects: Sets the error state of this to the
34 // error represented by code
35
36 virtual char* unparse() = 0;
37 // Effects: Returns the string representation of
38 // self
39
40 protected:
41 // What type of error state are we in?
42 int errorCode;
43
44 #endif

B.8 SGMLObject.cc

1 // SGMLObject.cc
2 // Jamie Gentry
3
4 #include "SGMLObject.h"
5
6 SGMLObject::SGMLObject(): errorCode(noError)
7 {
8 }
9
10 virtual SGMLObject::~SGMLObject()
11 {
12 }
13
14 void SGMLObject::setError(int code)
15 {
16     errorCode = code;
17     return;
18 }
B.9 SGMLDocument.h

```cpp
1  // SGMLDocument.h
2  // Jamie Gentry
3
4 #ifndef SGMLDocumentDef
5 #define SGMLDocumentDef
6
7 #include "SGMLElement.h"
8
9 class SGMLDocument:SGMLObject {
10   // An SGMLDocument is a mutable representation
11   // of an SGML document instance.
12
13 public:
14   SGMLDocument();
15   // Effects: Creates a new SGMLDocument with no
16   //       top-level element
17
18   inline void setTopEl(SGMLElement* el)
19   // Requires: el is not NULL
20   //       Modifies: this
21   // Effects: Sets the top element of self to be
22   //       el
23   {
24     top = el;
25     return;
26   }
27
28   inline SGMLElement* getTopEl()
29   // Effects: Returns the top element, or NULL
30   //       if there is none
31   {
32     return (top);
33   }
34
35 char* unparsen();
36   // Effects: Returns the string representation of
37   //       this
38
39 private:
40   SGMLElement* top;
41   }
42
43 #endif
```

B.10 SGMLDocument.cc

```cpp
1  // SGMLDocument.cc
2  // Jamie Gentry
3
4 #include "SGMLDocument.h"
5
6 SGMLDocument::SGMLDocument():
7   top((SGMLElement*) NULL)
8   {
9   }
10
11 char* SGMLDocument::unparse()
12 {
13   // See if this is in an error state first
14   if (!isGood())
15     return ("<ERROR Document>");
16
17 char* ret1 = stringCat("<Document>\n",
18   getTopEl() -> unparsen());
19 char* ret = stringCat(ret1, "\n</Document>");
20
21   return (ret);
22 }
```

B.11 SGMLElement.h

```cpp
1  // SGMLElement.h
2  // Jamie Gentry
```
class SGMLElement: public SGMLObject {
  // An SGMLElement represents an element in a
  // document instance.
  
public:
  SGMLElement();
  // Effects: Creates a new SGMLObject
  ~SGMLElement();
  // Effects: Destroys self
  char* unparse();
  // Effects: Returns a character representation
  // of self
  int addContainer(SGMEMElement* el);
  // Effects: Adds el as a containe of self.
  // Also sets el's container to self.
  // Returns true if successful, false
  // otherwise
  inline void setContainer(SGMEMElement* el)
  // Effects: Sets the container of self to be
  // el, but does not modify el
  {
    container = el;
  }
  inline void setName(char* n)
  // Requires: n is a string

  // Modifies: self
  // Effects: Sets the name of self to n
  {
    name = stringCopy(n);
    return;
  }

  inline char* getName() const
  // Effects: Returns the name of self, or NULL
  // if it is not yet set
  {
    return (name);
  }

  inline int getNContains() const
  // Effects: Returns the number of containers
  // of self
  {
    return (nContains);
  }

  inline SGMLElement* getContainer() const
  // Effects: Returns the container of self, or
  // NULL if none
  {
    return (container);
  }

private:
  // The name of the element
  char* name;
  // The element in which this content is
  // included, or NULL if none
  SGMLElement* container;
  // The containees of self
  SGMLElement** contains;
81 // The number of containees
82 int nContainees;
83
84 // How many containees we can hold
85 int containeeSpace;
86
87 #endif

B.12 SGMLElement.cc

1 // SGMLElement.cc
2 // Jamie Gentry
3
4 #include <stdlib.h>
5 #include <stdio.h>
6 #include "SGMLElement.h"
7
8 SGMLElement::SGMLElement(): name((char*) NULL),
9 container((SGMLElement*) NULL), nContainees(0),
10 containeeSpace(0)
11 {
12    containees = (SGMLElement**)  
13    calloc(size_t(1), sizeof (SGMLElement**));
14    if (containees == (SGMLElement**) NULL)
15        setError(mem);
16 }
17
18 SGMLElement::~SGMLElement()
19 {
20 if (containeeSpace > 0) {
21     for (int i = 0; i < nContainees; i++) {
22         delete containees[i];
23     }
24     free(containees);
25 }
26 }

28 int SGMLElement::addContainee(SGMLElement* el)
29 {
30   // See if we have enough space
31   if (nContainees == containeeSpace) {
32     // Need to add more space
33     containees = (SGMLElement**)  
34     realloc((void*) containees,
35             (containeeSpace + 5) *
36             sizeof (SGMLElement**));
37     if (containees == (SGMLElement**) NULL)
38         return (0);
39     containeeSpace += 5;
40   }
41   containees[nContainees] = el;
42   nContainees++;
43   // Set el's container
44   el->setContainer(this);
45   return (1);
46 }
47
52 char* SGMLElement::unparse()
53 {
54    char* begin = new char[strlen(name) + 3];
55    sprintf(begin, "<%s>
56    char* end = new char[strlen(name) + 4];
57    sprintf(end, "</%s>", name);
58    char* ret = begin;
59 if (nContainees == 0) {
60     ret = stringCat(begin, end);
61     delete [] begin;
62     delete [] end;
63     return (ret);
64   }
65   return (ret);
66 }

248 14
for (int i = 0; i < nContainers; i++) {
    char* ret2 = ret;
    ret = stringCat(ret2, containees[i]->unparse());
    delete [] ret2;
}
return (stringCat(ret, end));

B.13 DTD.h

#include <stdio.h>
#include "SGMLObject.h"

class DTD: SGMLObject {
    // A DTD represents an SGML DTD.

    public:
    DTD();
    // Effects: Creates a DTD

    char* unparse()
    {
        return ((char*) NULL);
    }
};

B.14 DTD.cc

int eltSt(PAC& pac)
{
    // Get our data

AppData* myData = (AppData*) pac.user_p;

// Get the Document Element Descriptor
DED* ded = (DED*) ADLENAddress(pac.dad);

// Make the element
SGMLElement* newEl = new SGMLElement;

// Set the new element's name
newEl->setName((char*) ded->elt_p->name.p);

// See if this should be the top element
if (myData->doc->getTopEl() == (SGMLElement*) NULL) {
  myData->doc->setTopEl(newEl);
  myData->current = newEl;
} else {
  // Add it to the current element
  int rc = myData->current->addContaineer(newEl);
  if (!rc)
    return (10);
  myData->current = newEl;
}

return (0);

B.17 eltNd.h

// eltNd.h
// Jamie Gentry

#include <portable.h>

B.18 eltNd.cc

// eltNd.cc
// Jamie Gentry

#include "eltNd.h"

#include "parseSGMLFile.h"

int eltNd(PAC& pac) {
  // Get our data
  AppData* myData = (AppData*) pac.user_p;

  // Change the current element to the parent of
  // the current element
  myData->current = myData->current->getContainer();
  return (0);
}

B.19 re.h

// re.h
// Jamie Gentry

#include <portable.h>

#include <yaspapi.h>

int eltNd(PAC& pac); // Requires: Is called by dispatch

// Modifies: pac
// Effects: Does what is needed with the end of a
// new element. If everything is okay,
// returns 0. Otherwise, returns 10
B.20 text.h

1 // text.h
2 // Jamie Gentry
3
4 #include <portable.h>
5 #include <yaspapi.h>
6
7 inline int text(PAC& pac)
8     // Effects: Returns 0. Eventually this will
9     // handle text (#PCDATA) data
10 {
11     return (0);
12 }

B.21 pi.h

1 // pi.h
2 // Jamie Gentry
3
4 #include <portable.h>
5 #include <yaspapi.h>
6
7 inline int pi(PAC& pac)
8     // Requires: Is called by dispatch

B.22 msg.h

1 // msg.h
2 // Jamie Gentry
3
4 #include <portable.h>
5 #include <yaspapi.h>
6
7 // If the message error severity is greater than
8     // this, we give up
9 const int autoQuit = 16;
10
11 int msg(PAC& pac);
12     // Requires: Called by dispatch
13     // Modifies: pac
14     // Effects: Displays the message indicated in the
15     // PAC. Returns 10 if the error severity
16     // is greater than or equal to the
17     // "autoquit" severity, and 0 otherwise

B.23 msg.cc

1 // msg.cc
2 // Jamie Gentry
3
4 #include <iostream.h>
5 #include "msg.h"
6 #include "parseSGMLFile.h"
8 // This is defined in AM2YASP.h. It is a string
9 // printed at the end of each series of messages
10 extern char* msgEnd;
11
12 int msg(PAC& pac)
13 {
14    // Get key items from the PAC:
15
16    // The list of variables to be plugged into the
17    // message
18    ADLEN* vars = &(pac.dad);
19    ADLEN* varList = (ADLEN*) ADLENAddress(vars);
20    int nVars = ADLENLen(vars);
21
22    // Make varList into an array of strings
23    char** varText = new char*[nVars];
24    for (int i = 0; i < nVars; i++)
25       varText[i] = (char*) ADLENAddress(varList[i]);
26
27    // The severity for this suite of messages
28    int severity = int(pac.errsev);
29
30    // The identifier for this message within the
31    // suite
32    int msgID = pac.msgcod;
33
34    // Whether of not this is the last message in
35    // the suite
36    int isLastMsg = int(pac.msglast);
37
38    // The application data
39    AppData* myData = (AppData*) pac.user_p;
40
41    // Get message handler
42    MessageHandler* msgObj
43       = myData->msgHandler;
44
45    // Put the variables into the message template
46    //
47    char* msgBody =
48       msgObj->getInstantiatedMsg(msgID, varText, nVars);
49    char* errID = msgObj->getID(msgID);
50
51    // Print severity
52    switch (severity) {
53       case 0: // Informational message
54          cerr << "INFO ";
55          break;
56       case 4: // Warning
57          cerr << "WARNING ";
58          break;
59       case 8: // Error
60          cerr << "ERROR ";
61          break;
62       case 12: // Severe error
63          cerr << "SEVERE ERROR ";
64          break;
65       case 16: // Terminating error
66          cerr << "TERMINATING ERROR ";
67          break;
68       
69    }
70
71    // Print message ID
72    cerr << "(" << errID << "); ";
73
74    // Print message
75    cerr << msgBody << endl;
76
77    // Print "last message" delimiter, if necessary
78    if (isLastMsg)
79       cerr << msgEnd;
80
81    // Is this message too severe to continue?
82    if (severity >= autoQuit)
83       return (10);
84
85    return (0);
B.24 sGet.h

1 // sGet.h
2 // Jamie Gentry
3
4 #include <portable.h>
5 #include <yaspapi.h>
6
7 int sGet(PAC& pac);
8 // Requires: Called by dispatch
9 // Modifies: pac
10 // Effects: Tries to allocate memory and put it in
11 // pac.stg.p. It tries to allocate amount
12 // pac.stg.len. If successful, returns 0.
13 // Otherwise, returns 10

B.25 sGet.cc

1 // sGet.cc
2 // Jamie Gentry
3
4 #include <stdlib.h>
5 #include "sGet.h"
6 #include "random.h"
7 #include "parseSGMLFile.h"
8
9 int sGet(PAC& pac)
10 {
11   ADLEN* storage = &(pac.stg);
12
13   ADLENAddress(storage)
14     = malloc(ADLENLen(storage));
15
16   // If we do not have enough storage, set the
17   // current element to be an error element
18   AppData* myData = (AppData*) pac.user_p;
19   if (ADLENAddress(storage) == NULL) {
20     myData->current->setError(mem);
21     return (10);
22   }
23
24   return (0);
25 }

B.26 sFre.h

1 // sFre.h
2 // Jamie Gentry
3
4 #include <portable.h>
5 #include <yaspapi.h>
6 #include <stdlib.h>
7
8 #include "random.h"
9
10 inline int sFre(PAC& pac)
11 {
12   // Requires: Called by dispatch
13   // Modifies: pac
14   // Effects: Frees the storage pointed to in
15   // the stg field of pac, and returns
16   // 0 if successful. (At the present
17   // time, we assume it is successful)
18   free(ADLENAddress(pac.stg));
19   return (0);
20 }

B.27 find.h

1 // find.h
```c
2 // Jamie Gentry
3 #include <portable.h>
4 #include <yaspapi.h>
5
6 int find(PAC& pac);
7 // Requires: Called by dispatch
8 // Modifies: pac
9 // Effects: Tries to get a file handle
10 // corresponding to the file described in
11 // pac.dad.p->EXID. Returns 0 if
12 // successful, an integer greater than 0
13 // otherwise
14

B.28 find.cc
1 // find.cc
2 // Jamie Gentry
3 #include <stdio.h>
4 #include <string.h>
5 #include "notify.h"
6 #include "find.h"
7 #include "random.h"
8 #include "parseSGMLFile.h"
9 #include "File.h"
10 #include "charSetID.h"
11 #include "pubIDFileName.h"
12
13 int find(PAC& pac)
14 {
15    // Get the application data
16    AppData* myData = (AppData*) pac.user_p;
17
18    // Get the file info from pac - this is in an
19    // EXID structure.
20    EXID* exid = (EXID*) ADLENAddress(pac.dad);
21
22    // Get the AHL structure - where the file is
23    // returned
24    AHL* ahl = (AHL*) &pac.ahl;
25
26    // Get sys ID and pub IDs
27    OFLEN* sysID = &(exid->sid);
28    OFLEN* pubID = &(exid->pid);
29
30    // Get the class. Unfortunately, each different
31    // file class is handled differently, and the
32    // YASP documentation is very vague about what
33    // exactly the file classes mean
34    unsigned char classField
35        = exid->asgn_cls;
36
37    // This is used in a couple of cases below
38    char* dot;
39
40    // Get the primary entity name
41    char* primEnt = myData->primaryEntity;
42
43    switch (classField) {
44    case CAPACITY_XFC: // 1: Capacity set
45    case NOTATION_XFC: // 9: Character data
46    case SYNTAX_XFC: // 12: Concrete syntax
47       // See if a system ID is given
48       if (OFLENLen(sysID) > 0) {
49          notify("System IDs currently unhandled");
50          return (10);
51       }
52    }
53
54    // See if a public ID is given
55    if (OFLENLen(pubID) > 0) {
56        char* fileName =
57            pubIDFileName(((char*)
58            ADLENAddress(exid->val))
59            + OFLENOffset(pubID));
60        if (fileName == (char*) NULL)
61            return (4);
```
File* f = new File(fileName);
AHLAddress(ahl) = (void*) f;
return (0);
}

// We shouldn't really get here...
return (4);
break;

case CHARSET_XFC: // 2: Character data
char* name = (char*) ALENAddress(exid->val);
AHLIdent(ahl)
  = charsetID(name +
    OLENOffset(pubID));
return (0);
break;

case UTODTW_XFC: // 15: Utility DTD to write
    // For now, the ODT file will have the same
    // name as the principal entity, but its
    // suffix will be replaced with ".odt"
    // Get this new file name
    strcpy(ODTName, primEnt);
    dot = strrchr(ODTName, '\0');
    if (dot == (char*) NULL)
      dot = &ODTName[strlen(primEnt) - 1];
    strcpy(dot, ".odt\0");

    // Create the new file
    File* ODTf = new File(ODTName);
    AHLAddress(ahl) = (void*) ODTf;
    delete [] ODTName;
    return (0);
    break;

    case UTSYNU_XFC: // 16: Util user-modified ocs
    // For now, the OCS file will have the same
    // name as the principal entity, but its
    // suffix will be replaced with ".ocs"
    // Get this new file name
    char* OCSName = new char[strlen(primEnt) + 4];
    strcpy(OCSName, primEnt);
    dot = strrchr(OCSName, '\0');
    if (dot == (char*) NULL)
      dot = &OCSName[strlen(primEnt) - 1];
    strcpy(dot, ".ocs\0");

    // Create the new file
    File* OCSf = new File(OCSName);
    AHLAddress(ahl) = (void*) OCSf;
    delete [] OCSName;
    return (0);
    break;

    case UTODTWL_XFC: // 17: Util local-system OCS
    if (myData->OCSFile == (char*) NULL)
      return (4);

    File* OCS = new File(myData->OCSFile);
    AHLAddress(ahl) = (void*) OCS;
    return (0);
    break;

    case PRIMARY_XFC: // 19: Primary Entity
    // Create new file
    File* primaryFile = new
    File(myData->primaryEntity);
    AHLAddress(ahl) = (void*) primaryFile;
    return (0);
    break;
B.29 xEnOp.h

1 // xEnOp.h
2 // Jamie Gentry
3
4 #include <portable.h>
5 #include <yasapi.h>
6
7 int xEnOp(PAC& pac);
8 // Requires: Is called by dispatch
9 // Modifies: pac
10 // Effects: Opens the file pointed to by AHL field
11 // of pac. If the file is already opened, returns 4; if the file cannot be
12 // opened, returns 6; if the file is
13 // invalid, returns 8. Otherwise,
14 // returns 0

B.30 xEnOp.cc

1 // xEnOp.cc
2 // Jamie GEntry
3
4 #include "xEnOp.h"
5 #include "File.h"
6 #include "random.h"

7 int xEnOp(PAC& pac)
8 {
9  // Get the file ID
10  File* f = (File*) AHLAddress(pac.ahl);
11  // See if it is already open
12  if (f->isOpen())
13    return (4);
14  
15  // Otherwise, open it
16  int rc = f->openRead();
17  if (!rc)
18    return (6);
19  return (0);
20 }

// This can be exactly the same as xEnOp
```c
15    return (4);
16
17    int rc = f->openWrite();
18
19    if (!rc)
20        return (6);
21
22    return (0);
23 }

B.34 utWrt.h
1 // utWrt.h
2 // Jamie Gentry
3
4 #include <portable.h>
5 #include <yaspapi.h>
6
7 int utWrt(PAC& pac);
8 // Requires: Is called by dispatch
9 // Modifies: pac
10 // Effects: Writes the data stored in pac.dad to
11 // the access-AHL. Returns 0 if
12 // successful and something greater than
13 // 0 otherwise

B.35 utWrt.cc
1 // utWrt.cc
2 // Jamie Gentry
3
4 #include <iostream>
5 #include "utWrt.h"
6 #include "File.h"
7 #include "random.h"
8
9 int utWrt(PAC& pac)
10 {
11    // Get the file ID
12    File* f = (File*) AHLAddress(pac.ahl);
13    // See if it is already open
14    if (f->isOpen())
```
9  int utWrt(PAC& pac)
10  {
11     // Get the file
12     File* f = (File*) AHLAddress(pac.ahl);
13     // Get the data
14     int dataLen = ADLENLen(pac.dad);
15     char* data = (char*) ADLENAddress(pac.dad);
16     int rc = f->put(data, dataLen);
17     if (rc)
18         return (0);
19     return (10);
20 }

B.36 utRd.h

1  // utRd.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
6
7  int utRd(PAC& pac);
8  // Requires: Is called by dispatch
9  // Modifies: pac
10  // Effects: Reads a record from the AccessID-AHL,
11     // and stores it in pac.dad. Returns 4 if
12     // the record read is of an invalid size,
13     // and 8 if the AHL is bad. Returns 0
14     // otherwise

B.37 utRd.cc

1  // utRd.cc

// Jamie Gentry
4  #include <string.h>
5  #include "utRd.h"
6  #include "random.h"
7  #include "File.h"
8
9  int utRd(PAC& pac)
10  {
11     // Get data buffer
12     void* data = ADLENAddress(pac.dad);
13     unsigned int dataLen
14         = ADLENLen(pac.dad);
15     // Get file
16     File* f = (File*) AHLAddress(pac.ahl);
17     if (!f->isOpenRead())
18         return (8);
19     // Read from file
20     size_t rc = f->readLen(data, size_t(dataLen));
21     if (dataLen != 0 && rc != dataLen)
22         return (8);
23     return (0);
24 }

B.38 utCl.h

1  // utCl.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
6  #include "xEnCl.h"
7
8  inline int utCl(PAC& pac)
// Requires: Called by dispatch
// Modifies: pac
// Effects: Closes the utility file pointed
to by the Access-AHL, and returns
// 0 if successful
{
  // This is the same as xEnCl
  return (xEnCl(pac));
}

B.39 xEnRd.h

// xEnRd.h
// Jamie Gentry

#include <portable.h>
#include <yaspapi.h>

int xEnRd(PAC& pac);

// Requires: Called by dispatch
// Modifies: pac
// Effects: Reads data from the file represented
// result in the dad field of pac.
// Returns 1 if okay; returns 8 if the file is not open for reading; returns
// 4 if we are at the end of the file.
// (Why this needs to return 1 instead of 0 is unclear, but it works better this way; it prevents YASP from sending weird messages)

B.40 xEnRd.cc

// xEnRd.cc
// Jamie Gentry
B.41 xEnCl.h

```c
1 #include <portable.h>
2 #include <yaspapi.h>
3 int xEnCl(PAC& pac);
4 // Requires: Is called by dispatch
5 // Effects: Closes the entity pointed to by pac.ahl. If successful, returns 0. If
6 // the access-AHL pointed to by pac.ahl is invalid, returns 8
```

B.42 xEnCl.cc

```c
1 #include "xEnCl.h"
2 #include "File.h"
3 int xEnCl(PAC& pac)
4 { // Get file access AHL
5   File* f = (File*) AHLAddress(pac.ahl);
6   f->close();
7   return 0;
8 }
```

B.43 proSt.h

```c
1 #include <portable.h>
2 #include <yaspapi.h>
```

B.44 proSt.cc

```c
1 // proSt.cc
2 // Jamie Gentry
3 #include <iostream.h>
4 #include "proSt.h"
5 #include "parseSGMLFile.h"
6 int proSt(PAC& pac)
7 { // Get the application data
8   AppData* myData = (AppData*) pac.user_p;
9   myData->doc = new SGMLDocument;
10  return 0;
11 }
```

B.45 proNd.h

```c
1 #include <portable.h>
2 #include <yaspapi.h>
```
inline int proNd(PAC& pac) {
  // Requires: Is called by dispatch
  // Effects: Returns 0
  return (0);
}

B.46 docNd.h

// docNd.h
// Jamie Gentry

#include <portable.h>
#include <yaspapi.h>

inline int docNd(PAC& pac) {
  // Requires: Is called by dispatch
  // Modifies: pac
  // Effects: Does what is needed with the end
  // of a document. Currently, this
  // consists of nothing. If
  // everything is okay, returns 0.
  // Otherwise, returns 10. (But since
  // this function currently does
  // nothing, there is no reason why
  // everything wouldn’t be okay)
  return (0);
}

B.47 etySt.h

// etySt.h
// Jamie Gentry

#include <portable.h>
#include <yaspapi.h>

inline int etyNd(PAC& pac) {
  // Requires: Called by dispatch
  // Effects: Reports the end of parsing of an
  // entity. Currently, this means
  // nothing
  return (0);
}

B.48 etySt.cc

// etySt.cc
// Jamie Gentry

#include "etyl.h"

int etySt(PAC& pac) {
  return (0);
}

B.49 etyNd.h

// etyNd.h
// Jamie Gentry

#include <portable.h>
#include <yaspapi.h>

inline int etyNd(PAC& pac) {
  // Requires: Called by dispatch
  // Effects: Reports the end of parsing of an
  // entity. Currently, this means
  // nothing
  return (0);
}
B.50 skip.h
1: // skip.h
2: // Jamie Gentry
3
4: #include <portable.h>
5: #include <yaspapi.h>
6
7: inline int skip(PAC& pac)
8:   // Effects: Returns 0. Eventually, this will
9:   //     handle skipped data
10: { 
11:   return (0);
12: }

B.51 decSt.h
1: // decSt.h
2: // Jamie Gentry
3
4: inline int decSt(const PAC& pac)
5:   // Requires: Is called by dispatch
6:   // Effects: Returns 0. Eventually this will
7:   //     handle the start of a declaration
8:   //     (if anything needs to be done)
9: { 
10:   return (0);
11: }

B.52 DTDSt.h
1: // DTDSt.h
2: // Jamie Gentry
3
4: #include <portable.h>
5: #include <yaspapi.h>
6
7: int DTDSt(PAC& pac)
8: { 
9:   // Get my data
10:   AppData* myData = (AppData*) pac.user_p;
11:   myData->dt = new DTD;
12:   return (0);
13: }

B.53 DTDSt.cc
1: // DTDSt.cc
2: // Jamie Gentry
3
4: #include "DTDSt.h"
5: #include "parseSGMLFile.h"
6
7: int DTDSt(PAC& pac)
8: { 
9:   // Get my data
10:   AppData* myData = (AppData*) pac.user_p;
11:   myData->dt = new DTD;
12:   return (0);
13: }

B.54 DTDNd.h
1: // DTDNd.h
2: // Jamie Gentry
3
4: #include <portable.h>
5: #include <yaspapi.h>
B.55 dapNd.h

```c
// dapNd.h

#include <portable.h>
#include <yaspapi.h>

inline int dapNd(PAC& pac)
{
    // Requires: Called by dispatch
    // Effects: Does what is necessary when the end of an SGML document is encountered right after the prolog ends. At the present time, nothing special needs to be done here, so this just returns 0
    return (0);
}
```

B.56 unimplFn.h

```c
// unimplFn.h

#include <portable.h>
```

B.57 unimplFn.cc

```c
// unimplFn.cc

#include <stdio.h>
#include "unimplFn.h"
#include "notify.h"

int unimplFn(PAC& pac)
{
    char errMsg[40];
    sprintf(errMsg, "Dispatch function %d is unimplemented", pac.func);
    notify(errMsg);
    return (10);
}
```

B.58 MessageHandler.h

```c
// MessageHandler.h

#include <portable.h>
```
```cpp
#include "random.h"

struct Message {
    char* id;
    char* content;
};

class MessageHandler {
    // A MessageHandler is a mutable object which
    // can display a message given a message ID

public:
    MessageHandler(const char* fileName);
    // Requires: fileName is a string
    // Effects: Returns a new message handler. If
    // fileName is not a valid file or
    // cannot for some reason be read,
    // returns a "bad" message handler

    ~MessageHandler();
    // Effects: Destroys this

    int operator()() const;
    // Effects: Returns true iff this is bad

    char* getMsg(int n) const
    // Requires: n is a valid message number
    // Effects: Returns the text of message number
    // n. The result can be destroyed
    // using delete []
    {
        return (stringCopy(messages[n].content));
    }

    inline char* getId(int n) const
    // Requires: n is a valid message number
    // Effects: Returns the ID string of message
    // number n. The result can be
    // destroyed using delete []

private:
    // 0 if bad, 1 if good
    int status;

    // The number of messages we know of
    int nMsgs;

    // The message themselves
    Message* messages;
};

B.59 MessageHandler.cc

// MessageHandler.cc
// Jamie Gentry

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "MessageHandler.h"
```
MessageHandler::MessageHandler(const char* fileName):
    char* fileSymbolName = new char[100],
    status(1),
    nMsgs(0),
    messages((Message*)
    NULL)
{
    // Open file
    FILE* msgFile = fopen(fileName, "r");
    if (msgFile == (FILE*) NULL) {
        status = 0;
        return;
    }
    // The number of messages we have room for so far
    int messageCapacity =
        20;
    messages = (Message*)
    calloc(size_t(messageCapacity),
    sizeof(Message));
    if (messages == (Message*) NULL) {
        fclose(msgFile);
        return;
    }
    // Get "header" line
    char* line = new char[100];
    fgets(line, 100, msgFile);
    delete [] line;
    // Get rest of lines
    do {
        char* line = new char[100];
        if (fgets(line, 100, msgFile) == NULL)
            break;
    } while (1);
    // Resize messages...
    messages = (Message*)
    realloc((void*) messages,
    (size_t) nMsgs * sizeof (Message));
    if (messages != (Message*) NULL)
        status = 1;
    fclose(msgFile);
}

// the line
if (line[strlen(line) - 1] == '\n')
    line[strlen(line) - 1] = '\0';
messages[nMsgs].id = stringCopyUntil(line, 10);
messages[nMsgs].content
    = stringCopy(line, 10);
nMsgs++;
// See if we’ve ran out of room for messages
if (nMsgs == messageCapacity) {
    messages = (Message*)
    realloc((void*) messages,
    size_t(messageCapacity + 20) *
    sizeof (Message));
    if (messages == (Message*) NULL) {
        fclose(msgFile);
        return;
    }
    else
        messageCapacity += 20;
}
// Get rid of carriage return at the end of

int MessageHandler::operator()() const
{
    return (status == 0);
}

char* MessageHandler::getInstantiatedMsg(int n,
    char** vars, int nVars) const
{
    // Get a copy of the template
    char* templ = getMsg(n);

    // The return string
    char* ret = (char*)
        calloc(size_t(100), sizeof (char));
    if (ret == (char*) NULL) {
        delete [] templ;
        return (ret);
    }

    // The size of the return string
    int retSize = 100;

    int nextVar = 0;

    // Where we are in the source string
    char* pSrc = templ;

    // Where we are in the return string
    char* pRet = ret;

    do {
        if (*pSrc != '\' ) {
            // This is a non-control character
            *pRet = pSrc;
            pSrc++;
            pRet++;
        }
    } while (pRet - ret >= retSize);

    return (ret);

B.60 File.h

1 // File.h
2 // Jamie Gentry
3
4 #include <stdio.h>
5 #include "random.h"
6
7 class File {
8     // A File represents a regular file.
9
10 public:
11    enum openMode {read, write, append};
12
13    File(const char* fileName);
14    // Requires: fileName is a string
15    // Effects: Creates a new (closed) File
16
17    File();
18    // Effects: Creates a new temporary file
19
20 ~File();
21    // Modifies: this
22    // Effects: Destroys this
23
24    inline char* getName() const
25    // Effects: Returns a copy of the name of this
26    {
27        return (stringCopy(name));
28    }
29
30    inline int isOpen() const
31    // Modifies: this
32    // Effects: return true iff this is open
33    {
34        return (isFileOpen);
35    }
36
37    inline int isOpenRead() const
38    // Modifies: this
39    // Effects: Returns true iff this is open
40    // with mode read
41    {
42        return (isOpen() && mode == read);
43    }
44
45    int openRead();
46    // Requires: this is not already open
47    // Modifies: this
48    // Effects: Opens this for reading. If
49    // successful, returns true; otherwise,
50 // returns false
51 int openWrite();
52 // Requires: this is not already open
53 // Modifies: this
54 // Effects: Opens this for writing. If
55 // successful, returns true; otherwise,
56 // returns false
57
58 void close();
59 // Modifies: this
60 // Effects: Closes this
61
62 inline int atEOF() const
63 // Effects: Returns true iff we are at the end
64 // of the file
65 {
66   return (isEOF);
67 }
68
69 char* get();
70 // Modifies: this
71 // Effects: Returns a line from this, including
72 // the carriage return. If a line
73 // cannot be read returns NULL. The
74 // result can be destroyed using delete
75 // []
76
77 char* get(int len);
78 // Modifies: this
79 // Effects: Returns at most len characters from
80 // this. If already at end of file,
81 // returns NULL. The result can be
82 // destroyed using delete []
83
84 inline size_t readLen(void* dest, size_t len)
85 // Requires: File is open for reading
86 // Modifies: dest
87 // Effects: Tries to read len characters from
88
89 // the file into dest. Returns the
90 // number of characters read.
91 {
92   return (int(fread(dest, size_t(1), len,
93     readFile)));
94 }
95 int put(char* data, int len);
96 // Requires: data contains at least len
97 // Modifies: this
98 // Effects: Writes len characters from data to
99 // the file. Returns true if
100 // successful, false otherwise
101
102 private:
103
104 // The filename
105 char* name;
106
107 // true iff file is open
108 int isFileOpen;
109 // The mode the file is open in
110 openMode mode;
111
112 // The input file
113 FILE* readFile;
114
115 // The output file
116 FILE* writeFile;
117
118 // A buffer for our reads and writes
119 char* buffer;
120
121 // true iff we are at the end of the file
122
123 // The size of buffer
124 int bufferSize;
B.61 File.cc

1 // File.cc
2 // Jamie Gentry
3
4 #include <string.h>
5 #include <iostream.h>
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include "File.h"
9
10 File::File(const char* fileName):
11   name(stringCopy(fileName)), isFileOpen(0),
12   readFile((FILE*) NULL), writeFile((FILE*) NULL),
13   bufferSize(0), isEOF(0)
14 {
15 }
16
17 File::File(): isFileOpen(0),
18   readFile((FILE*) NULL), writeFile((FILE*) NULL),
19   bufferSize(0), isEOF(0)
20 {
21   char tempName[L_tmpnam];
22   tmpnam(tempName);
23   name = stringCopy(tempName);
24 }
25
26
27 File::~File()
28 {
29   close();
30   delete [] name;
31 }
32
33 int File::openRead()
34 {
35   isEOF = 0;
36
37   // Set up buffer
38   buffer = (char*)
39     calloc(size_t(100), sizeof (char));
40   if (buffer == (char*) NULL)
41     return (0);
42   bufferSize = 100;
43
44   readFile = fopen(name, "r");
45   if (readFile == (FILE*) NULL)
46     return (0);
47   else {
48     isFileOpen = 1;
49     mode = read;
50     return (1);
51   }
52 }
53
54 int File::openWrite()
55 {
56   isEOF = 0;
57
58   writeFile = fopen(name, "w");
59   if (writeFile == (FILE*) NULL)
60     return (0);
61   else {
62     isFileOpen = 1;
63     mode = write;
64     return (1);
65   }
66 }
67
68 void File::close()
69 {
70   isEOF = 0;
71
72   if (!isFileOpen)
return;
switch (mode) {
case read:
    fclose(readFile);
    readFile = (FILE*) NULL;
    break;
case write:
    fclose(writeFile);
    writeFile = (FILE*) NULL;
    break;
case append:
    // This is not yet handled...
    break;
}
isFileOpen = 0;
return;

char* File::get()
{
    // If we're at the end of the file, don't even
    // try...
    if (isEOF)
        return ((char*) NULL);
    // Make sure file is properly opened...
    if (!isFileOpen || mode != read)
        return ((char*) NULL);
    for (int i = 0; i < len; i++)
    {
        int rc = fgetc(readFile);
        if (rc != EOF)
            buffer[i] = char(rc);
    }
    return

buffer = (char*)
realloc((void*) buffer,
size_t(bufferSize + 100) *
sizeof (char));
if (buffer == (char*) NULL)
    return (buffer);
bufferSize += 100;
if (buffer[i] == '\n')
    buffer[i + 1] = '\n';
return (buffer);
else {
    isEOF = 1;
    if (i == 0)
        return ((char*) NULL);
    else {
        buffer[i] = '\n';
        return (buffer);
    }

char* File::get(int len)
{
    char* ret = new char[len + 1];
    if (fgets(ret, len + 1, readFile) ==
(char*) NULL)
        return ((char*) NULL);
    return (ret);

int File::put(char* data, int len)
{
    if (mode != write)
        return (0);
151 for (int i = 0; i < len; i++) {
152    int rc = fputc(int(data[i]), writeFile);
153    if (rc == EOF)
154        return 0;
155 }
156
157 return (1);
158 }
159
160 // Does not really return...
161 return;
162 }

B.64 pubIDfileName.h
1 // pubIDfileName.h
2 // Jamie Gentry
3
4 const int pubIDsLen = 2;
5
6 const struct pubIDNamePair {
7    char* name;
8    char* fileName;
9 } pubIDs[pubIDsLen] = {
10    {"ISO 8879:1986//CAPACITY Reference//EN",
11     "/mit/jcgentry/SGML/YASP/smp00/standard.cap"},
12    {"ISO 8879:1986//SYNTAX Reference//EN",
13     "/mit/jcgentry/SGML/YASP/smp00/standard.csd"}
14    };
15
16 char* pubIDfileName(const char* name);
17 // Requires: name is a string
18 // Effects: Returns the file name corresponding to
19 // the ID in string name. If name is not
20 // recognized, returns NULL. Otherwise,
21 // the result can be destroyed using
22 // delete []

B.65 pubIDfileName.cc
1 // pubIDfileName.cc
2 // Jamie Gentry
3
4 #include <stdio.h>
5 #include "pubIDfileName.h"
B.66 charsetID.h

1 // charsetID.h
2 // Jamie Gentry
3
4 #include <portable.h>
5 #include <yaspapi.h>
6
7 // These are integers which are the AHL IDs for
8 // different character sets.
9
10 // This must be the number of elements in IDsLen
11 const int IDsLen = 1;
12
13 const struct charsetIDPair {
14     char* charsetName;
15     int ID;
16 } IDs[IDsLen] = {
17     {"ISO 646-1983//CHARSET International Reference
18         Version (IRV)//ESC 2/5 4/0",
19         ISO_646_1983_AHLID}
20    );
21
22 int charsetID(const char* name);
23 // Requires: name is a string
24 // Effects: Returns the ID corresponding to the
25 // character set name. If name is
26 // not recognized, returns UNKNOWN_AHLID

B.67 charsetID.cc

1 // charsetID.cc
2 // Jamie Gentry
3
4 #include <iostream.h>
5 #include "charsetID.h"
6
7 int charsetID(const char* name)
8 {
9     for (int i = 0; i < IDLen; i++)
10         if (strcmp(IDs[i].charsetName, name) == 0)
11             return (IDs[i].ID);
12     return (UNKNOWN_AHLID);
13 }

B.68 random.h

1 // random.h
2 // Jamie Gentry
3
4 ifndef randomDef
5 define randomDef
6
7 #include <portable.h>
8 #include <yaspapi.h>
9 #include <string.h>
10 #include <iostream.h>
11
12 ifndef YASP_PARSER_CODE
13
14 inline void* & ADLENAddress(ADLEN & a)
15 {
16     return (a.p);
17 }
18
19 inline void* & ADLENAddress(ADLEN* a)
20 {
21   return (a->p);
22 }
23
24 #else
25
26 inline void*& ADLENAAddress(ADLEN& a)
27 {
28   return (a._p);
29 }
30
31 inline void*& ADLENAAddress(ADLEN* a)
32 {
33   return (a->._p);
34 }
35
36 #endif
37
38 inline int& ADLENLen(ADLEN& a)
39 {
40   return (a.len);
41 }
42
43 inline int& ADLENLen(ADLEN* a)
44 {
45   return (a->len);
46 }
47
48 inline void*& AHLAddress(AHL* a)
49 {
50   return (a->p);
51 }
52
53 inline void*& AHLAddress(AHL& a)
54 {
55   return (a.p);
56 }
57
58 inline int& AHLIdent(AHL& a)
59 {
60   return (a.i);
61 }
62
63 inline int& AHLIdent(AHL* a)
64 {
65   return (a->i);
66 }
67
68 inline char* ETYName(ETY& e)
69 {
70   return ((char*) ADLENAAddress(e.name));
71 }
72
73 inline char* ETYName(ETY* e)
74 {
75   return ((char*) ADLENAAddress(e->name));
76 }
77
78 inline unsigned int OFLENOffset(OFLEN& o)
79 {
80   return (o.rel);
81 }
82
83 inline unsigned int OFLENOffset(OFLEN* o)
84 {
85   return (o->rel);
86 }
87
88 inline unsigned int OFLNLen(OFLEN& o)
89 {
90   return (o.len);
91 }
92
93 inline unsigned int OFLNLen(OFLEN* o)
94 {
95   return (o->len);
96 }
97
98 char* stringCopy(const char* s);
99 // Requires: s is a (non-null) string
100 // Effects: Returns a copy of s. The result can be
101 // destroyed using delete
102
103 inline char* stringCopy(const char* s, int n)
104   // Requires: cc is a string, n >= 0.
105   // Effects: Returns a copy of cc up to character n on. If n is greater
106   // or equal to the size of cc,
107   // returns NULL. The result can be
108   // destroyed with delete []
109 {
110   return (stringCopy(s + n));
111 }
112
114 char* stringCopyUntil(const char* s, int n);
115 // Requires: s is a string, n >= 0.
116 // Effects: Returns a string consisting of the
117 // first n characters of cc. If n is
118 // greater or equal to the size of s,
119 // returns NULL. The result can be
120 // destroyed using delete []
121
122 inline int stringCmpPrefix(const char* pre,
123   const char* cc)
124   // Requires: pre and cc are strings
125   // Effects: Returns true iff cc begins with
126   // the characters in pre
127 {
128   return (strncmp(pre, cc, strlen(pre)));
129 }
130
131 int stringCmpPostfix(const char* post,
132   const char* cc);
133 // Requires: pre and cc are strings
134 // Effects: Returns true iff cc ends with the
135 // characters in post
136
137 char* stringStripWhiteSpace(const char* cc);
138 // Requires: cc is a string
139 // Effects: Returns the string obtained by
140 // stripping the whitespace from cc. The
141 // result can be destroyed using delete
142 // [].
143
144 int stringIsProperDelims(const char* cc,
145   const char* front,
146   const char* back);
147 // Requires: All arguments are strings
148 // Effects: Returns true iff cc is delimited by
149 // front and back
150
151 char* stringCat(const char* s1, const char* s2);
152 // Requires: s1, s2 are strings
153 // Effects: Concatenates string s1 to s2, and
154 // returns the result. The result
155 // can be destroyed with delete []
156
157
157#endif

B.69 random.cc

1 // random.cc
2 // Jamie Gentry
3
4 #include "random.h"
5
6 char* stringCopy(const char* s)
7 {
8   char* ret = new char[strlen(s) + 10];
9   if (ret == (char*) NULL)
10      return (ret);
11   strcpy(ret, s);
12   return (ret);
13 }
14
char* stringCopyUntil(const char* s, int n)
{
    if (n >= strlen(s))
        return ((char*) NULL);
    char* ret = stringCopy(s);
    ret[n] = '\0';
    return (ret);
}

char* stringCat(const char* s1, const char* s2)
{
    int ls1 = strlen(s1);
    int ls2 = strlen(s2);
    char* ret = new char[ls1 + ls2 + 2];
    for (int j = 0; j < ls1 + ls2 + 2; j++)
        ret[j] = '\0';
    for (int i = 0; i < ls1; i++)
        ret[i] = s1[i];
    for (i = 0; i < ls2; i++)
        ret[ls1 + i] = s2[i];
    ret[ls1 + ls2] = '\0';
    return (ret);
}

B.70 Makefile

# Makefile
# Jamie Gentry
# Directories
#includeDir = $(HOME)/include/YASP
libDir = $(HOME)/lib

# Trash
rubbish = .* ~ core test out /*

# Application object files
appObjs = parseSGMLFile.o test.o dispatch.o 
        unimplFn.o notify.o SGMLObject.o 
        sGet.o find.o random.o File.o msg.o 
        MessageHandler.o xEnOp.o etySt.o 
        xEnRd.o charsetID.o pubIDFileName.o 
        xEnCl.o utOpW.o utWrt.o utRd.o 
        DTDSt.o DTD.o eltSt.o SGMLElement.o 
        eltNd.o proSt.o SGMLDocument.o

# Compiler Flags
CPPFLAGS = -I$(includeDir) -DAIX_SYS -DGNUCC
LDFLAGS = -L$(libDir)

.cc.o:
    g++ $(CPPFLAGS) $(CFLAGS) -Wall -c $< 
    -o $@

####### Application #########

test: $(appObjs)
g++ $(LDFLAGS) -o $0 $(appObjs) -lyasp

test.o: test.cc parseSGMLFile.h DTD.h 
        SGMLDocument.h SGMLElement.h

parseSGMLFile.o: parseSGMLFile.cc 
    parseSGMLFile.h DTD.h SGMLDocument.h 
    dispatch.h SGMLElement.h 
    MessageHandler.h

dispatch.o: dispatch.cc dispatch.h unimplFn.h 
    notify.h sGet.h random.h find.h msg.h 
    sFre.h xEnOp.h etySt.h xEnRd.h decSt.h 
    etyNd.h xEnCl.h utOpW.h utWrt.h utCl.h 
    utOpR.h utRd.h proSt.h DTDSt.h DTDNd.h
proNd.h dapNd.h eltSt.h eltNd.h \
docNd.h pi.h parseSGMLFile.h text.h \
re.h skip.h

unimplFn.o: unimplFn.cc unimplFn.h notify.h  
notify.o: notify.cc notify.h  
SGMLObject.o: SGMLObject.cc SGMLObject.h  
sGet.o: sGet.cc sGet.h random.h

find.o: find.cc find.h notify.h parseSGMLFile.h \
       File.h charsetID.h pubIDFileName.h

random.o: random.cc random.h  
File.o: File.cc File.h random.h

msg.o: msg.cc msg.h

MessageHandler.o: MessageHandler.cc \
       MessageHandler.h random.h File.h

xEnOp.o: xEnOp.cc xEnOp.h File.h File.h random.h

etySt.o: etySt.cc etySt.h random.h

xEnRd.o: xEnRd.cc xEnRd.h random.h

charsetID.o: charsetID.cc charsetID.h

pubIDFileName.o: pubIDFileName.cc pubIDFileName.h

xEnCl.o: xEnCl.cc xEnCl.h File.h

utOpW.o: utOpW.cc utOpW.h File.h random.h

utWrt.o: utWrt.cc utWrt.h File.h random.h

utRd.o: utRd.cc utRd.h random.h File.h

DTDSt.o: DTDSt.cc DTDSt.h DTD.h parseSGMLFile.h

DTD.o: DTD.cc DTD.h

eltSt.o: eltSt.cc eltSt.h SGMLElement.h random.h \
       parseSGMLFile.h

SGMLElement.o: SGMLElement.cc SGMLElement.h \
       SGMLObject.h random.h

eltNd.o: eltNd.h eltNd.cc parseSGMLFile.h

proSt.o: proSt.cc proSt.h parseSGMLFile.h

SGMLDocument.o: SGMLDocument.cc SGMLDocument.h \
       SGMLElement.h

Clean

clean:

$(RM) $(rubbish)
Appendix C  Sample Run

The following is a sample run of the AM2SGML program. The SGML document instance and DTD used is the following, taken from the "Guidelines for Electronic Text Encoding and Interchange."¹

```
<!DOCTYPE anth [ 
<!ELEMENT anth  - - (poem+)>
<!ELEMENT poem  - - (title?, stanza+)>
<!ELEMENT title   - 0 (#PCDATA)>
<!ELEMENT stanza - 0 (line+)>
<!ELEMENT line    0 0 (#PCDATA)>
]>

<anth>
  <poem>
    <title>The SICK ROSE
    <stanza>
      <line>O Rose thou art sick. 
      <line>The invisible worm,
      <line>That flies in the night
      <line>In the howling storm:
    <stanza>
      <line>Has found out thy bed
      <line>Of crimson joy:
      <line>And his dark secret love
    </stanza>
      <line>Does thy life destroy
  </poem>
</anth>
```

The following is the command line used to parse the file:

```
test testDir/myTest7.sgml 0 init.ocs 0 0
```

where testDir/myTest7.sgml is the name of the file given above, and init.ocs is a compiled SGML declaration produced from a previous execution of AM2SGML. The results are as follows:

---

1. Sperberg-McQueen, pp. 16-17.
These are the expected results.