SGML Functions for AthenaMuse 2

by

James C. Gentry

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND COMPUTER SCIENCE IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 1996

Signature of Author .....
Department of Electrical Engineering and Computer Science
May 28, 1996

Certified by .....
Steven Lerman
Professor of Civil and Environmental Engineering
Director, Center for Educational Computing Initiatives
Thesis Supervisor

Accepted by .....
Frederic R. Morgenthaler
Graduate Officer, Department of Electrical Engineering and Computer Science

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

List of Tables

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.
2. "Getting Started with SGML: A Guide to the Standard Generalized Markup Language and Its Role in Information Management," ArborText, 1995.
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 thee 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.

---

1. Margaret Meehan et al., *AthenaMuse 2.1 Documentation*, Cambridge, MIT AthenaMuse Software Consortium, 1995.
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 speficy 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 **`INCLUDE`** or **`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?

<![INCLUDE [Baton Rouge]]>
```

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 `Baton Rouge` is marked with **`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 **`INCLUDE`** to **`IGNORE`** in the above fragment.

A marked section looks as follows:[3]

<! [*status-keyword* [*marked-section*]]>

3. "The SGML Primer," SoftQuad, 1995. Note that this and other SGML samples are expressed as a Backus-Naur Form grammar.

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

Class Name	Description	Where Described
	DTD Classes	
`MMdtd`	Represents an entire DTD. This can contain comments (`MMsgmlComment` objects) or **DOCTYPE** definitions (`MMdtdDocumentType` objects).	Section 2.2.1.1 on page 20
`MMdtdDocumentType`	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).	Section 2.2.1.2 on page 22
`MMdtdElement`	Represents a DTD **ELEMENT** declaration. It includes a content model, represented by an `MMdtdCommentModel` object.	Section 2.2.1.3 on page 26
`MMdtdContentModel`	Represents a content model for an **ELEMENT** declaration.	Section 2.2.1.4 on page 29
`MMdtdAttribute`	Represents one attribute of an **ELEMENT** declaration. Note that each `MMdtdAttribute` object represents one item in an **ATTLIST** declaration.	Section 2.2.1.5 on page 32
`MMdtdEntity`	Represents an **ENTITY** declaration in a DTD.	Section 2.2.1.6 on page 36
`MMdtdNotation`	Represents a DTD **NOTATION** declaration.	Section 2.2.1.7 on page 38

TABLE 1. The AM2SGML Classes

Class Name	Description	Where Described
	Document Instance Classes	
`MMsgmlDocument`	Represents an entire document instance. This corresponds to a DTD (`MMdtd` object) which describes the document instance. This contains a top-level element (`MMdtdElement` object).	Section 2.2.2.1 on page 42
`MMsgmlElement`	Represents an element in a document instance. This corresponds to an **`ELEMENT`** declaration (`MMdtdElement` 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.	Section 2.2.2.2 on page 44
`MMsgmlPCData`	Represents text data of type **`#PCDATA`** in an element.	Section 2.2.2.3 on page 45
`MMsgmlEmpty`	Represents the lack of data in a document instance element. In other words, this corresponds to the content model **`EMPTY`**.	Section 2.2.2.4 on page 46
`MMsgmlAttribute`	This represents an attribute of a document instance element. This corresponds to an item in a DTD **`ATTLIST`** declaration (an `MMdtdAttribute` object), which states which possible value this attribute can take.	Section 2.2.2.5 on page 47
	Classes Used in DTDs or Document Instances	
`MMsgmlSectionMark`	Represents a section mark: **`INCLUDE`**, **`IGNORE`**, or none.	Page 18; Section 2.2.1.8 on page 40
`MMsgmlComment`	Represents a comment in a DTD or document instance.	Section 2.2.1.9 on page 41

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 `MMdtd` Class

This represents a DTD in its entirety. A DTD consists of one or more concurrent document-type (**`DOCTYPE`**) declarations and possibly comments.[4] Therefore, the `MMdtd` class consists mainly of handles to `MMdtdDocumentType` 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.[5] The specification for MMdtd is given below.

`class MMdtd`

Overview

MMdtd is a class of mutable objects which describe a group of concurrent **DOCTYPE** declarations.

Methods

`upon Construct: list els, handle hMarkArg`
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.
Effects: Creates a new MMdtd object containing the **DOCTYPE** declarations given in `els` (as well as the comments), and with the marking specified by `hMarkArg`.

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

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

`upon ConstructFromString: string s`
Effects: Creates an MMdtd object from string `s`.

`on DocTypes: return list`
Effects: Returns a list of handles to MMdtdDocumentType objects, representing the **DOCTYPE** declarations in `self`.

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

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

---

5. C. M. Sperberg-McQueen, and Lou Burnard, editors, *Guidelines for Electronic Text Encoding and Interchange: Draft Version 2*, 1993, p. 23.

`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 `MMsgmlComment`.
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 `MMsgmlComment`.
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 2. Members of `MMdtd`

Member	Type	Description
`docTypeDecls`	`list`	A list of handles to `MMdtdDocumentType` or `MMsgmlComment` 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.
`hMark`	`handle`	A handle to `MMsgmlSectionMark`. The marking of the declaration.

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:

< ! **`DOCTYPE`** *name* [*item*$^{+}$] > | < ! **`DOCTYPE`** *name* [**`system`** *"file"*] [*item**] >

Each *item* is an entity, notation, attribute list, or element declaration.[6] If *file* is absent, then the document type is defined based on the *item*s. Otherwise, it is defined based on the concatenation of the *item*s and the declaration in *file*. In this case, there may be no *item*s. 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`.

---

6. Sperberg-McQueen, pp. 23-24.

`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.[7] 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` object, 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.[8]

`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 3. Members of `MMdtdDocumentType`

Member	Type	Description
`name`	`string`	The name of the declaration.
`systemName`	`string`	The "reference name" of this `MMdtdDocumentType`. This is the name that is used if this `MMdtdDocumentType` is referenced using the **SYSTEM** clause of another **DOCTYPE** declaration.
`hSystemRef`	`handle`	A handle to an `MMdtdDocumentType`. If self contains a **SYSTEM** reference to another **DOCTYPE** declaration, then `hSystemRef` represents this declaration.
`items`	`list`	A list of handles to the elements in the **DOCTYPE** declaration. This does not include elements referenced by a **SYSTEM** clause.
`hMark`	`handle`	A handle to `MMsgmlSectionMark`. The marking of the declaration.
`hParent`	`handle`	A handle to an `MMdtd`. The DTD object in which this declaration is found.

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]

9. "The SGML Primer."

< ! **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

$$+ (inc\text{-}elt_1 \mid inc\text{-}elt_2 \mid inc\text{-}elt_3 \mid \ldots \mid inc\text{-}elt_n)$$

while an exclusion list has the form

$$- (exc\text{-}elt_1 \mid exc\text{-}elt_2 \mid exc\text{-}elt_3 \mid \ldots \mid exc\text{-}elt_m)$$

where the *inc-elt*s and *exc-elt*s 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)>
```

10.Sperberg-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 4. Members of `MMdtdElement`

Member	Type	Description
`isBeginOpt`	`boolean`	`TRUE` iff the beginning tag of an element defined by `self` is optional.
`isEndOpt`	`boolean`	`TRUE` iff the ending tag of an element defined by `self` is optional.
`hContentModel`	`handle`	A handle to an `MMdtdContentModel` object. This specifies exactly what types of data an element defined by `self` can contain.
`hMark`	`handle`	A handle to `MMsgmlSectionMark`. The marking of the declaration.
`hParent`	`handle`	A handle to an `MMdtdDocumentType`, or `NULL`. The document type in which `self` is defined, if any.
`inclusionExs`	`list`	A list of elements which can be included anywhere within the content model of `self`.
`exclusionExs`	`list`	A list of elements which can be included anywhere within the content model of `self`.
`atts`	`list`	The list of all attributes defined for `self`.

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 | 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.

`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`.

`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 5. Members of `MMdtdContentModel`

Member	Type	Description
`modelType`	`string`	This is the type of the content model - one of `'empty`, `'PCData`, `'el`, or `'complex`.

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 => PCDataModel;
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, &PCDataModel}} =>
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[11]

<!**ATTLIST** *element-type*

11."The SGML Primer."

attribute-name$_1$ *attribute-value*$_1$ *default-value*$_1$
attribute-name$_2$ *attribute-value*$_2$ *default-value*$_2$
attribute-name$_3$ *attribute-value*$_3$ *default-value*$_3$
.
.
.
attribute-name$_n$ *attribute-value*$_n$ *default-value*$_n$
>

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.[12] Each *default-value* is the default value of the attribute. This can be one of the following:[13]

- 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."

13."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 corresponding 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

Value of attType (or thisAttType)	Type of value (e.g. hDefaultVal)
'cData	string
'entity	string
'entities	list of strings
'ID	string
'IDRef	string
'IDRefs	list of strings
'name	string
'names	list of strings
'nmToken	string
'nmTokens	list of strings
'notation	handle to MMdtdNotation
'number	integer
'numbers	list of integers
'nuToken	string
'nuTokens	list of strings

`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 7. Members of `MMdtdContentModel`

Member	Type	Description
`hEl`	`handle`	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.
`name`	`string`	The name of the attribute.
`attType`	`string`	The type of the attribute; one of the values given in Table 6 on page 34.
`defaultStat`	`string`	One of `'implied`, `'required`, `'current`, or `'given`.
`enumVals`	`list`	If `self` is an enumerated type, then this is a list of the possible values of `self`. Otherwise, it should be empty.
`defaultVal`	`handle`	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`.

Activities

none

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.[14]

An entity has the form[15]

> <!**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.[16]

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*.[17] The following is an example of such a declaration:

```
<!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.

`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 8. Members of MMdtdEntity

Member	Type	Description
isParameter	boolean	TRUE iff self is a parameter entity.
isSystem	boolean	TRUE iff self is a system entity
name	string	The name of the entity.
isSubdoc	boolean	TRUE iff val represents an SGML file with its own DTD.
val	string	the value of the entity.
hNotation	handle	A handle to an MMdtdNotation (or NULL). The notation used to interpret val.

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."

< ! **NOTATION** *notation-name* **SYSTEM** *"system-identifier"* >

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 9. Members of MMdtdNotation

Member	Type	Description
name	string	The name of the notation.
val	string	The value of the notation.
hParent	handle	A handle to an MMdtdDocumentType object. The document type in which the notation is declared.

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 `MMsgmlSectionMark` 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 `MMdtdEntity` 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 section 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 `hContext`.

`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 10. Members of `MMsgmlSectionMark`

Member	Type	Description
`markType`	`string`	The type of mark this is. One of `'include`, `'ignore`, `'none`, or `'entity`.
`markEntity`	`handle`	If `self` corresponds to an entity, this is the `MMdtdEntity` object representing that entity.

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 11. Members of `MMsgmlComment`

Member	Type	Description
`text`	`string`	The text of the comment.

Activities

none

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

```
MMsgmlComment {'Construct, "This is my comment text"} =>
  myComment
```

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

```
MMsgmlComment {'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 correponding 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 12. Members of `MMsgmlDocument`

Member	Type	Description
`hDTD`	`handle`	A handle to an `MMdtd` object. The DTD describing the document instance.
`hTop`	`handle`	A handle to an `MMdtdElement`. The top-level element.

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 Table 13.

TABLE 13. Members of `MMsgmlElement`

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

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`
Requires: `hNewEntity` is a handle to an `MMdtdEntity` object.
Modifies: `self`.
Effects: Adds the entity represented by `hNewEntity` to the end of `self`.

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

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

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

Members

The members of `MMsgmlPCData` are given in Table 14.

TABLE 14. Members of `MMsgmlPCData`

Member	Type	Description
`data`	`list`	A list of handles to `strings` or `MMdtdEntity` objects. This is the data represented by `self`.

Activities

none

2.2.2.4 The `MMsgmlEmpty` Class

This class represents the **`EMPTY`** data content.

`class MMsgmlEmpty`

Overview

An MMsgmlEmpty object is an immutable representation of the SGML **`EMPTY`** data construct.

Methods

`upon Construct`
Effects: Creates an `MMsgmlEmpty` object.

`upon ConstructFromString: string s, handle hContext`
Requires: `hContext` is a handle to an `MMsgmlDocument` object.
Effects: Creates an `MMsgmlEmpty` 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 `MMsgmlAttribute` are given in Table 15 on page 48.

TABLE 15. Members of `MMsgmlPCData`

Member	Type	Description
`hSpec`	`handle`	A handle to `MMdtdAttribute`. The attribute corresponding to `self`.
`val`	`handle`	The value of self. This has type given in Table 6 on page 34.

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`.

---

1. Geoff Bartlett and Pierre Richard, *The SGML Parser Interface: Functional Specification Version 1.25*, Yorktown Heights, IBM T. J. Watson Research Center, 1992.

One of the fields of the PAC data structure is called `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. `parseSGMLFile` sets this field to the function `dispatch`. Another field is `user_p`. This is where AM2SGML stores the *application data*. Also stored there is space for the structure that AM2SGML uses to represent the parsed SGML document. This structure has the type `SGMLDocument`.

Whenever YASP calls `dispatch`, it passes to it the original PAC structure as an argument. The `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 `dispatch` function is basically a large `switch` statement. There are seperate functions to deal with each possible platform-dependent function and each possible interesting parsing event; these functions are called the *service functions*. `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, `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 `SGMLDocument` structure stored in the application data, if necessary.

After YASP is finished parsing the entire SGML file, `PRSMAIN` returns control to the function `parseSGMLFile`, which called it. By this time, the `SGMLDocument` structure stored in the user information represents a totally parsed SGML document. `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 describer 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 16. The Service Functions

Value of `func`	Name of Service Function	Description of Service Function
	Interesting Parsing Events	
`ELTST_PFC`	`eltSt`	The start of an element
	Section 3.3.5.1 on page 58	
`ELTND_PFC`	`eltNd`	The end of an element
	Section 3.3.5.2 on page 59	
`RE_PFC`	`re`	Relevant record end (this is when a record is ended unexpectedly by the end of a file)

TABLE 16. The Service Functions

Value of func	Name of Service Function	Description of Service Function
TEXT_PFC	text	Test data has been encountered
	Section 3.3.5.3 on page 59	
PI_PFC	pi	A processing instruction has been encountered
PROST_PFC	proSt	The beginning of the prolog has been encountered. The prolog consists of the SGML declaration and DTD
	Section 3.3.5.13 on page 64	
PROND_PFC	proNd	The end of the prolog has been encountered
DOCND_PFC	docNd	The end of a document has been reached
	Section 3.3.5.14 on page 65	
ETYST_PFC	etySt	The start of an entity has been encountered (an entity in this case is the same as a file)
ETYND_PFC	etyNd	The end of an entity has been encountered
SKIP_PFC	skip	For some reason, data in the SGML file has been skipped
DECST_PFC	decSt	The start of an SGML declaration has been encountered
DTDST_PFC	DTDSt	The start of a DTD has been encountered
	Section 3.3.5.15 on page 65	
DTDND_PFC	DTDNd	The end of a DTD has been encountered
	Section 3.3.5.16 on page 65	
DAPND_PFC	dapNd	The end of the document has been encountered right after the end of the prolog
	Platform-Dependent Services	
MSG_PFC	msgs	Print a message
	Section 3.3.5.4 on page 60	
SGET_PFC	sGet	Get storage
	Section 3.3.5.5 on page 61	
SFRE_PFC	sFre	Free storage
	Section 3.3.5.5 on page 61	
FIND_PFC	find	Create a file
	Section 3.3.5.6 on page 61	
XENOP_PFC	xEnOp	Open a text file
	Section 3.3.5.7 on page 62	
UTOPR_PFC	utOpR	Open a binary (utility) file for reading
	Section 3.3.5.8 on page 63	
UTOPW_PFC	urOpW	Open a binary file for writing
	Section 3.3.5.8 on page 63	
UTWRT_PFC	utWrt	Write to a binary file
	Section 3.3.5.9 on page 63	
UTRD_PFC	utRd	Read from a binary file
	Section 3.3.5.10 on page 63	

TABLE 16. The Service Functions

Value of `func`	Name of Service Function	Description of Service Function
`UTCL_PFC`	`utCl`	Close a binary file
	Section 3.3.5.11 on page 64	
`XENRD_PFC`	`xEnRd`	Read from a text file
	Section 3.3.5.12 on page 64	
`XENCL_PFC`	`xEnCl`	Close a text file
	Section 3.3.5.11 on page 64	

3.3.4 The `SGMLObject` Class and its Subclasses

The `SGMLObject` 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 `SGMLObject` is given in Section B.7 on page 77, and the implementation is given in Section 1 on page 77.

The `SGMLObject` 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 `SGMLObject` 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 `text` is given in Section B.20 on page 83.

3.3.5.4 The `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 `stderr`. YASP usually sends a number of messages in a row; a group of messages is called a *suite*.

Before AM2SGML calls the YASP entry point `PRSMAIN`, it creates a `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 `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 `dad` field of the PAC
- The severity of this suite of messages. This is represented by an integer, which is given in the `errsev` field of the PAC. The possible error severities are given in Table 17.

TABLE 17. YASP Message Severities

Number	Type of Message
0	Informational message
4	Warning
8	Error
12	Severe error
16	Terminating error

- The message identifier. This is the integer that the message handler uses to find the proper message template. This is stored in the `msgcod` field of the PAC

- Whether or not this is the last message in the suite. If it is, the `msglast` 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_cls` of a structure called an `EXID`, which

is given in the dad 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 18. YASP File Classes

Name	Description
CAPACITY_XFC NOTATION_XFC SYNTAX_XFC	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 dad 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.
CHARSET_XFC	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.
UTODTW_XFC	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.
UTSYNU_XFC	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.
UTSYNL_XFC	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.
PRIMARY_XFC	This represents the SGML file to be parsed.

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 `utOpR` and `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 `utOpR` just calls `xEnOp`. `utOpW` simply gets the `File` object from the `ahl` field of the PAC, makes sure it is not already open, and if not calls its `File::openWrite` method. If the file was already opened, `utOpW` returns `4`. If the file is not successfully opened, `utOpW` returns `6`. Otherwise, it returns `0`. The code for `utOpR` and `utOpW` is given in Section B.31 on page 88 through Section B.33 on page 89.

3.3.5.9 The `utWrt` Function

This function is used by YASP to write to a binary file. The `File` object is stored in the `ahl` field of the PAC, and the data to be written is stored in the `dad` field of the PAC, in an ADLEN structure. The pointer in the ADLEN structure points to the actual data, and the integer in the ADLEN structure is set to the length of the data. AM2SGML uses the `File::put` method, which also takes a pointer and an integer, to write the data. The declaration for `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 `utRd` Function

This function is used to read from a binary file. Once again, the FileID is stored in the `ahl` field of the PAC, and the `dad` field contains an ADLEN structure. The integer part of the 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 ADLEN structure. `utRd` uses the `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 `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 ouputs 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.

4 The Future of SGML and AM2

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).[1] 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]

```
<!-- A DTD for a simple subset of HTML -->

<!-- The first DOCTYPE declaration will specify tags which we
   normally associate with HTML. The second declaration tries to
   split an HTML document into physical pages. -->

<!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 O O &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 O O TITLE>
   <!ELEMENT TITLE - - #PCDATA -(EM)>
                                    <!-- The beginning and end tags for a
                                       title are necessary.
                                       We do not want to allow emphasized
                                       text in the title -->
   <!ELEMENT BODY O O (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
```

1. Dave Raggett, "Document Type Definition for the HyperText Markup Language (HTML DTD)", 1995.

```
40      align (left | center | right | justify) left
41                                      -- The alignment of the paragraph -
42                                      by default, it is left-aligned --
43      nowrap (nowrap) #IMPLIED        -- Disable word wrap --
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>
54 ]>
```

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

```
1  // test.cc
2  // Jamie Gentry
3
4  #include <iostream.h>
5  #include "AM2YASP.h"
6  #include "parseSGMLFile.h"
7
8  int main(int argc, char** argv)
9  {
10   // The main SGML document strucure
11   SGMLDocument* mainDoc;
12
13   // This should take four arguments: The SGML
14   // file to be parsed, a DTD file (as long as the
15   // DTD is not in the main SGML file), possibly
16   // an OCS file, and possibly an ODT file.  If we
17   // have less than four arguments, we should give
18   // an error.
19   if (argc < 5) {
20     cerr << "Bad arguments." << endl;
21     return (-1);
22   }
23
24   // Go through the arguments.
25   const char* mainFile = argv[1];
26   const char* DTDFile  = strcmp(argv[2], "0") == 0
27     ? (char*) NULL : stringCopy(argv[2]);
28   const char* OCSFile  = strcmp(argv[3], "0") == 0
29     ? (char*) NULL : stringCopy(argv[3]);
30   const char* ODTFile  = strcmp(argv[4], "0") == 0
31     ? (char*) NULL : stringCopy(argv[4]);
32
33   // Now, parse the file mainFile.
34   mainDoc = parseSGMLFile(mainFile, DTDFile,
35         OCSFile, ODTFile);
36
37   // Now, unparse the object mainDoc.
38   cerr << "----------" << endl;
39   char* unparsed = mainDoc->unparse();
40   if (mainDoc != (SGMLDocument*) NULL)
41     cerr << unparsed << endl;
42   else
43     cerr << "(NULL)" << endl;
44   cerr << "----------" << endl;
45
46   return (0);
47 }
```

B.2 AM2YASP.h

```
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

```
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
```

```
15   // The document instance for this file
16   SGMLDocument* doc;
17
18   // The element we are currently looking at
19   SGMLElement* current;
20
21   // The file name of the primary entity
22   char* primaryEntity;
23
24   // The file name of theDTD
25   char* DTDFile;
26
27   // The OCS file name
28   char* OCSFile;
29
30   // The ODT file name
31   char* ODTFile;
32
33   // The message handler
34   MessageHandler* msgHandler;
35 };
36
37 SGMLDocument* parseSGMLFile(const char* fileName,
38   const char* DTDFileName,
39   const char* OCSFileName,
40   const char* ODTFileName);
41 // Requires: fileName is a string
42 // Modifies: dtd, doc
43 // Effects: Tries to parse the file fileName as an
44 //          SGML document. If successful, returns
45 //          an SGMLDocument representing the file
46 //          fileName.  This returned value can be
47 //          destroyed using delete.
48 //          If any of DTDFileName, OCSFileName,
49 //          or ODTFileName are not present, they
50 //          should be set to NULL.
51 //          If there is any error, then NULL is
52 //          returned.
```

B.4 parseSGMLFile.cc

```
1// parseSGMLFile.cc
2  // Jamie Gentry
3
4  #include <stdio.h>
5  #include <iostream.h>
6  #include <portable.h>
7  #include <yaspapi.h>
8
9  #include "parseSGMLFile.h"
10 #include "dispatch.h"
11 #include "random.h"
12
13 // Parser main entry point
14 extern "C" unsigned int C_FCT PRSMAIN (PAC *);
15
16 SGMLDocument* parseSGMLFile(const char* fileName,
17             const char*
18             DTDFileName,
19             const char*
20             OCSFileName,
21             const char*
22             ODTFileName)
23 {
24   // Set up the YASP PAC structure
25   PAC pac;
26
27   // ID
28   pac.id            = PAC_IDENTIFIER;
29
30   // Reserved
31   pac.resv          = short(0);
32
33   // Reserved
34   pac.argc          = short(0);
35
36   // Reserved
37   pac.argv          = (char**) 0;
```

```
38
39  // Address for app services
40  pac.ase           = dispatch;
41
42  // Private app use
43  pac.user_p        = (void*) new AppData;
44
45  // Get message handler
46  // This file name should be moved elsewhere, but
47  // for the time being it is easier just to hard
48  // code it here
49  MessageHandler* msgObj = new MessageHandler(
50  "/mit/jcgentry/SGML/YASP/parser/msg/yspus.txt");
51
52  // See if the message handler was created ok
53  if (!msgObj)
54    return (0);
55
56  // Get a pointer to the application data
57  AppData* myData   = (AppData*) pac.user_p;
58
59  // Set up application data structure
60  myData->dtd       = (DTD*) NULL;
61  myData->doc       = (SGMLDocument*) NULL;
62
63  // The element we are currently parsing
64  myData->current   = (SGMLElement*) NULL;
65
66  // The name of the SGML file we are parsing
67  myData->primaryEntity
68                    = stringCopy(fileName);
69
70  // The name of the OCS file given, or NULL is
71  // none is given
72  myData->OCSFile   = ((OCSFileName ==
73       (char*) NULL) ?
74            (char*) NULL :
75            stringCopy(OCSFileName));
76
77  // The name of the ODT file, or NULL
78  myData->ODTFile   = ((ODTFileName ==
79       (char*) NULL) ?
80            (char*) NULL :
81            stringCopy(ODTFileName));
82
83  // The name of the DTD, or NULL
84  myData->DTDFile   = ((DTDFileName ==
85       (char*) NULL) ?
86            (char*) NULL :
87            stringCopy(DTDFileName));
88
89  myData->msgHandler
90                    = msgObj;
91
92  // Call main parser entry point
93  unsigned int rc   = PRSMAIN(&pac);
94
95  // To clean up better, we need to see if
96  // anything should be deleted here.
97
98  // Check return code from parser
99  if (rc == 0)
100   return (myData->doc);
101 else
102   return ((SGMLDocument*) NULL);
103}
```

B.5 dispatch.h

```
1  // dispatch.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
6
7  int dispatch(PAC* pac);
8  // 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

```
// dispatch.cc
// Jamie Gentry

#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));

  case FIND_PFC:  // 11: Get a FileID-AHL
```

```
71    return (find(*pac));
72    break;
73
74  case XENOP_PFC: // 12: Open external entity
75    return (xEnOp(*pac));
76    break;
77
78  case UTOPR_PFC: // 13: Open utility file for
79                  // reading
80    return (utOpR(*pac));
81    break;
82
83  case UTOPW_PFC: // 14: Open utility file for
84                  // writing
85    return (utOpW(*pac));
86    break;
87
88  case UTWRT_PFC: // 15: Write utility file record
89    return (utWrt(*pac));
90    break;
91
92  case UTRD_PFC:  // 16: Read utility file record
93    return (utRd(*pac));
94    break;
95
96  case UTCL_PFC:  // 17: Close a utility file
97    return (utCl(*pac));
98    break;
99
100 case XENRD_PFC: // 18: Read data from an
101                 // external entity
102   return (xEnRd(*pac));
103   break;
104
105 case XENCL_PFC: // 19: Close an eternal entity
106   return (xEnCl(*pac));
107   break;
108
109 case PROST_PFC: // 20: Begin prolog
110   return (proSt(*pac));
111   break;
112
113 case PROND_PFC: // 21: End prolog
114   return (proNd(*pac));
115   break;
116
117 case DOCND_PFC: // 22: End of DOC found
118   return (docNd(*pac));
119   break;
120
121 case ETYST_PFC: // 23: Report starting an entity
122   return (etySt(*pac));
123   break;
124
125 case ETYND_PFC: // 24: Report end of an entity
126   return (etyNd(*pac));
127   break;
128
129 case SKIP_PFC:  // 27: Skipped data
130   return (skip(*pac));
131   break;
132
133 case DECST_PFC: // 28: Start of an SGML
134                 // declaration
135   return (decSt(*pac));;
136   break;
137
138 case DTDST_PFC: // 29: Start of DTD
139   return (DTDSt(*pac));
140   break;
141
142 case DTDND_PFC: // 30: End of DTD
143   return (DTDNd(*pac));
144   break;
145
146 case DAPND_PFC: // 31: End of DOC right after
147                 // prolog
148   return (dapNd(*pac));
```

```
149     break;
150
151   default:
152     return (unimplFn(*pac));
153   }
154}
```

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
45 #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

```
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* unparse();
36   // Effects: Returns the string representation of
37   //          this
38
39 private:
40   SGMLElement* top;
41 };
42
43 #endif
```

B.10 SGMLDocument.cc

```
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()->unparse());
19   char* ret  = stringCat(ret1, "\n</Document>");
20
21   return (ret);
22 }
```

B.11 SGMLElement.h

```
1  // SGMLElement.h
2  // Jamie Gentry
```

```
3
4  #ifndef SGMLElementDef
5  #define SGMLElementDef
6
7  #include "SGMLObject.h"
8  #include "random.h"
9
10 class SGMLElement: public SGMLObject {
11   // An SGMLElement represents an element in a
12   // document instance.
13
14 public:
15   SGMLElement();
16   // Effects: Creates a new SGMLObject
17
18   ~SGMLElement();
19   // Effects: Destroys self
20
21   char* unparse();
22   // Effects: Returns a character representation
23   //          of self
24
25   int addContainee(SGMLElement* el);
26   // Modifies: self, el
27   // Effects: Adds el as a containee of self.
28   //          Also sets el's container to self.
29   //          Returns true if successful, false
30   //          otherwise
31
32   inline void setContainer(SGMLElement* el)
33     // Modifies: self
34     // Effects: Sets the container of self to be
35     //          el, but does not modify el
36     {
37       container = el;
38     }
39
40   inline void setName(char* n)
41     // Requires: n is a string
42     // Modifies: self
43     // Effects: Sets the name of self to n
44     {
45       name = stringCopy(n);
46       return;
47     }
48
49   inline char* getName() const
50     // Effects: Returns the name of self, or NULL
51     //          if it is not yet set
52     {
53       return (name);
54     }
55
56   inline int getNContainees() const
57     // Effects: Returns the number of containees
58     // of self
59     {
60       return (nContainees);
61     }
62
63   inline SGMLElement* getContainer() const
64     // Effects: Returns the container of self, or
65     //          NULL if none
66     {
67       return (container);
68     }
69
70 private:
71   // The name of the element
72   char* name;
73
74   // The element in which this content is
75   // included, or NULL if none
76   SGMLElement* container;
77
78   // The containees of self
79   SGMLElement** containees;
80
```

```
81   // The number of containees
82   int nContainees;
83
84   // How many containees we can hold
85   int containeeSpace;
86 };
87
88 #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 }
27
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
40     containeeSpace += 5;
41   }
42
43   containees[nContainees] = el;
44   nContainees++;
45
46   // Set el's container
47   el->setContainer(this);
48
49   return (1);
50 }
51
52 char* SGMLElement::unparse()
53 {
54   char* begin  = new char[strlen(name) + 3];
55   sprintf(begin, "<%s>\n", name);
56   char* end    = new char[strlen(name) + 4];
57   sprintf(end, "</%s>", name);
58
59   char* ret    = begin;
60
61   if (nContainees == 0) {
62     ret        = stringCat(begin, end);
63     delete [] begin;
64     delete [] end;
65     return (ret);
66   }
```

```
67
68   for (int i = 0; i < nContainees; i++) {
69     char* ret2 = ret;
70     ret        =
71       stringCat(ret2, containees[i]->unparse());
72     delete [] ret2;
73   }
74
75   return (stringCat(ret, end));
76 }
```

B.13 DTD.h

```
1  // DTD.h
2  // Jamie Gentry
3
4  #include <stdio.h>
5  #include "SGMLObject.h"
6
7  class DTD: SGMLObject {
8    // A DTD represents an SGML DTD.
9
10 public:
11   DTD();
12   // Effects: Creates a DTD
13
14   char* unparse()
15     {
16       return ((char*) NULL);
17     }
18
19 };
```

B.14 DTD.cc

```
1  // DTD.cc
2  // Jamie Gentry
3
4  #include "DTD.h"
5
6  DTD::DTD()
7  {
8  }
```

B.15 eltSt.h

```
1  // eltSt.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
6
7  int eltSt(PAC& pac);
8  // Requires: Is called by dispatch
9  // Modifies: pac
10 // Effects: Does what is needed with the creation
11 //          of a new element. If everything is
12 //          okay, returns 0. Otherwise, returns 10
```

B.16 eltSt.cc

```
1  // eltSt.cc
2  // Jamie Gentry
3
4  #include "eltSt.h"
5  #include "SGMLElement.h"
6  #include "random.h"
7  #include "parseSGMLFile.h"
8
9  int eltSt(PAC& pac)
10 {
11   // Get our data
```

```
12  AppData* myData    = (AppData*) pac.user_p;
13
14  // Get the Document Element Descriptor
15  DED* ded           = (DED*)
16    ADLENAddress(pac.dad);
17
18  // Make the element
19  SGMLElement* newEl = new SGMLElement;
20
21  // Set the new element's name
22  newEl->setName((char*) ded->elt_p->name.p);
23
24  // See if this should be the top element
25  if (myData->doc->getTopEl() ==
26      (SGMLElement*) NULL) {
27    myData->doc->setTopEl(newEl);
28    myData->current = newEl;
29  }
30  else {
31    // Add it to the current element
32    int rc = myData->current->addContainee(newEl);
33
34    if (!rc)
35      return (10);
36
37    myData->current = newEl;
38  }
39
40  return (0);
41 }
42
```

B.17 eltNd.h

```
1  // eltNd.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
6
7  int eltNd(PAC& pac);
8  // Requires: Is called by dispatch
9  // Modifies: pac
10 // Effects: Does what is needed with the end of a
11 //          new element. If everything is okay,
12 //          returns 0. Otherwise, returns 10
```

B.18 eltNd.cc

```
1  // eltNd.cc
2  // Jamie Gentry
3
4  #include "eltNd.h"
5  #include "parseSGMLFile.h"
6
7  int eltNd(PAC& pac)
8  {
9    // Get our data
10   AppData* myData  = (AppData*) pac.user_p;
11
12   // Change the current element to the parent of
13   // the current element
14   myData->current  =
15     myData->current->getContainer();
16
17   return (0);
18 }
19
```

B.19 re.h

```
1  // re.h
2  // Jamie Gentry
3
```

```
4  #include <portable.h>
5  #include <yaspapi.h>
6
7  inline int re(PAC& pac)
8       // Effects: Returns 0. Eventually, this will
9       //          handle a relevant record end
10 {
11   return (0);
12 }
```

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 }
13
```

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
9       // Effects: Does nothing. We do not yet
10      //          handle processing instructions.
11 {
12   return (0);
13 }
14
```

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"
7
```

```
// This is defined in AM2YASP.h. It is a string
// printed at the end of each series of messages
extern char* msgEnd;

int msg(PAC& pac)
{
  // Get key items from the PAC:

  // The list of variables to be plugged into the
  // message
  ADLEN* vars       = &(pac.dad);
  ADLEN* varList    = (ADLEN*) ADLENAddress(vars);
  int nVars         = ADLENLen(vars);

  // Make varList into an array of strings
  char** varsText   = new char*[nVars];
  for (int i = 0; i < nVars; i++)
    varsText[i] =
      (char*) ADLENAddress(varList[i]);

  // The severity for this suite of messages
  int severity      = int(pac.errsev);

  // The identifier for this message within the
  // suite
  int msgID         = pac.msgcod;

  // Whether of not this is the last message in
  // the suite
  int isLastMsg     = int(pac.msglast);

  // The application data
  AppData* myData   = (AppData*) pac.user_p;

  // Get message handler
  MessageHandler* msgObj
                    = myData->msgHandler;

  // Put the variables into the message template
  char* msgBody     =
    msgObj->getInstantiatedMsg(msgID, varsText,
              nVars);
  char* errID       = msgObj->getID(msgID);

  // Print severity
  switch (severity) {
  case 0:  // Informational message
    cerr << "INFO ";
    break;
  case 4:  // Warning
    cerr << "WARNING ";
    break;
  case 8:  // Error
    cerr << "ERROR ";
    break;
  case 12: // Severe error
    cerr << "SEVERE ERROR ";
    break;
  case 16: // Terminating error
    cerr << "TERMINATING ERROR ";
    break;
  }

  // Print message ID
  cerr << "(" << errID << "): ";

  // Print message
  cerr << msgBody << endl;

  // Print "last message" delimiter, if necessary
  if (isLastMsg)
    cerr << msgEnd;

  // Is this message too severe to continue?
  if (severity >= autoQuit)
    return (10);

  return (0);
```

```
86 }
```

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       // Requires: Called by dispatch
12       // Modifies: pac
13       // Effects: Frees the storage pointed to in
14       //          the stg field of pac, and returns
15       //          0 if successful. (At the present
16       //          time, we assume it is successful)
17 {
18   free(ADLENAddress(pac.stg));
19   return (0);
20 }
```

B.27 find.h

```
1  // find.h
```

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

B.28 find.cc

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

```
140  default:
141    char errMsg[40];
142    sprintf(errMsg,
143      "File Class Field %d is unhandled",
144      int(classField));
145    notify(errMsg);
146    return (10);
147  }
148}
```

B.29 xEnOp.h

```
1  // xEnOp.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.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,
12 //          returns 4; if the file cannot be
13 //          opened, returns 6; if the file is
14 //          invalid, returuns 8. Otherwise,
15 //          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
8  int xEnOp(PAC& pac)
9  {
10   // Get the file ID
11   File* f = (File*) AHLAddress(pac.ahl);
12
13   // See if it is already open
14   if (f->isOpen())
15     return (4);
16
17   // Otherwise, open it
18   int rc  = f->openRead();
19
20   if (!rc)
21     return (6);
22
23   return (0);
24 }
```

B.31 utOpR.h

```
1  // utOpR.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
6  #include "xEnOp.h"
7
8  inline int utOpR(PAC& pac)
9        // Requires: Is called by dispatch
10       // Modifies: pac
11       // Effects: Opens the file pointed to by
12       //          pac.ahl for reading; returns 4 if
13       //          the file is already open, 6 if
14       //          the file cannot be opened, and
15       //          0 otherwise
16 {
17   // This can be exactly the same as xEnOp
```

```
18   return (xEnOp(pac));
19 }
```

B.32 utOpW.h

```
1  // utOpW.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
6
7  int utOpW(PAC& pac);
8  // Requires: Is called by dispatch
9  // Modifies: pac
10 // Effects: Opens the utility file given in the
11 //          FileID-AHL for writing. Returns 4 if
12 //          the file is already open and 6 if it
13 //          cannot be opened. Returns 0 if
14 //          everything is ok
```

B.33 utOpW.cc

```
1  // utOpW.cc
2  // Jamie Gentry
3
4  #include "utOpW.h"
5  #include "File.h"
6  #include "random.h"
7
8  int utOpW(PAC& pac)
9  {
10   // Get the file ID
11   File* f = (File*) AHLAddress(pac.ahl);
12
13   // See if it is already open
14   if (f->isOpen())
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.h>
5  #include "utWrt.h"
6  #include "File.h"
7  #include "random.h"
8
```

```
9  int utWrt(PAC& pac)
10 {
11   // Get the file
12   File* f     = (File*) AHLAddress(pac.ahl);
13
14   // Get the data
15   int dataLen = ADLENLen(pac.dad);
16   char* data  = (char*) ADLENAddress(pac.dad);
17
18   int rc = f->put(data, dataLen);
19   if (rc)
20     return (0);
21
22   return (10);
23 }
```

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
2  // Jamie Gentry
3
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
14   unsigned int dataLen
15               = ADLENLen(pac.dad);
16
17   // Get file
18   File* f     = (File*) AHLAddress(pac.ahl);
19
20   if (!(f->isOpenRead()))
21     return (8);
22
23   // Read from file
24   size_t rc   = f->readLen(data, size_t(dataLen));
25   if (dataLen != 0 && rc != dataLen)
26     return (8);
27   return (0);
28 }
```

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)
```

```
9       // Requires: Called by dispatch
10      // Modifies: pac
11      // Effects: Closes the utility file pointed
12      //          to by the Access-AHL, and returns
13      //          0 if successful
14 {
15   // This is the same as xEnCl
16   return (xEnCl(pac));
17 }
```

B.39 xEnRd.h

```
1  // xEnRd.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
6
7  int xEnRd(PAC& pac);
8  // Requires: Called by dispatch
9  // Modifies: pac
10 // Effects: Reads data from the file represented
11 //          by the ahl field of pac, and puts the
12 //          result in the dad field of pac.
13 //          Returns 1 if okay; returns 8 if the
14 //          file is not open for reading; returns
15 //          4 if we are at the end of the file.
16 //          (Why this needs to return 1 instead of
17 //          0 is unclear, but it works better this
18 //          way; it prevents YASP from sending
19 //          weird messages)
```

B.40 xEnRd.cc

```
1  // xEnRd.cc
2  // Jamie Gentry
3
4  #include <string.h>
5  #include "xEnRd.h"
6  #include "File.h"
7  #include "random.h"
8
9  int xEnRd(PAC& pac)
10 {
11   // Get access-AHL
12   File* f    = (File*) AHLAddress(pac.ahl);
13
14   if (!f->isOpenRead())
15     return (8);
16
17   if (f->atEOF())
18     return (4);
19
20   char* line = f->get();
21
22   // Put the data in the dad, if it is valid;
23   if (f->atEOF() && line == (char*) NULL)
24     return (4);
25
26   ADLENAddress(pac.dad)
27             = (void*) line;
28   ADLENLen(pac.dad)
29             = strlen(line);
30
31   // I don't know why this should be returning 1
32   // instead of 0 - the documentation is not
33   // clear. I do know, however, that it prevents a
34   // lot of problems!!!
35
36   return (1);
37 }
```

B.41 xEnCl.h

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

B.42 xEnCl.cc

```
1  // xEnCl.cc
2  // Jamie Gentry
3
4  #include "xEnCl.h"
5  #include "File.h"
6
7  int xEnCl(PAC& pac)
8  {
9    // Get file access AHL
10   File* f = (File*) AHLAddress(pac.ahl);
11   f->close();
12
13   return (0);
14 }
```

B.43 proSt.h

```
1  // proSt.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
6
7  int proSt(PAC& pac);
8  // Requires: Called by dispatch
9  // Modifies: pac
10 // Effects: Creates a new SGMLDocument instance.
11 //          Returns 0
```

B.44 proSt.cc

```
1  // proSt.cc
2  // Jamie Gentry
3
4  #include <iostream.h>
5  #include "proSt.h"
6  #include "parseSGMLFile.h"
7
8  int proSt(PAC& pac)
9  {
10   // Get the application data
11   AppData* myData = (AppData*) pac.user_p;
12
13   myData->doc     = new SGMLDocument;
14
15   return (0);
16 }
```

B.45 proNd.h

```
1  // proNd.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
```

```
6
7  inline int proNd(PAC& pac)
8        // Requires: Is called by dispatch
9        // Effects: Returns 0
10 {
11   return (0);
12 }
```

B.46 docNd.h

```
1  // docNd.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
6
7  inline int docNd(PAC& pac)
8        // Requires: Is called by dispatch
9        // Modifies: pac
10       // Effects: Does what is needed with the end
11       //          of a document. Currently, this
12       //          consists of nothing. If
13       //          everything is okay, returns 0.
14       //          Otherwise, returns 10. (But since
15       //          this function currently does
16       //          nothing, there is no reason why
17       //          everything wouldn't be okay)
18 {
19   return (0);
20 }
```

B.47 etySt.h

```
1  // etySt.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
6
7  int etySt(PAC& pac);
8  // Reqiroes: Is called by dispatch
9  // Modifies: pac
10 // Effects: Currently does nothing except return 0
```

B.48 etySt.cc

```
1  // etySt.cc
2  // Jamie Gentry
3
4  #include "etySt.h"
5
6  int etySt(PAC& pac)
7  {
8    return (0);
9  }
```

B.49 etyNd.h

```
1  // etyNd.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
6
7  inline int etyNd(PAC& pac)
8        // Requires: Called by dispatch
9        // Effects: Reports the end of parsing of an
10       //          entity. Currently, this means
11       //          nothing
12 {
13   return (0);
14 }
```

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  // Requires: Is called by dispatch
9  // Modifies: pac
10 // Effects: Does everything it needs to do when
11 //          the parser hits the start of a DTD.
12 //          Currently, this means creating a new
13 //          DTD object. Returns 0 if successful,
14 //          10 otherwise (although presently there
15 //          is no way not to be successful)
```

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
12   myData->dtd     = new DTD;
13
14   return (0);
15 }
```

B.54 DTDNd.h

```
1  // DTDNd.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
```

```
6
7  inline int DTDNd(PAC& pac)
8        // Requires: Is called by dispatch
9        // Effects: Returns 0. Eventually will do
10       //          what is necessary when the end
11       //          of a DTD is hit
12 {
13   return (0);
14 }
```

B.55 dapNd.h

```
1  // dapNd.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
6
7  inline int dapNd(PAC& pac)
8        // Requires: Called by dispatch
9        // Effects: Does what is necessary when the
10       //          end of an SGML document is
11       //          encountered right after the
12       //          prolog ends. At the present time,
13       //          nothing special needs to be done
14       //          here, so this just returns 0
15 {
16   return (0);
17 }
```

B.56 unimplFn.h

```
1  // unimplFn.h
2  // Jamie Gentry
3
4  #include <portable.h>
5  #include <yaspapi.h>
6
7  int unimplFn(PAC& pac);
8  // Requires: To be called from dispatch.
9  // Effects: Anounces that this function is
10 //          unimplemented, and returns 10. Once
11 //          AM2SGML is complete, there will be
12 //          no unimplemented service functions, so
13 //          this function will be unnecessary
```

B.57 unimplFn.cc

```
1  // unimplFn.cc
2  // Jamie Gentry
3
4  #include <stdio.h>
5  #include "unimplFn.h"
6  #include "notify.h"
7
8  int unimplFn(PAC& pac)
9  {
10   char errMsg[40];
11   sprintf(errMsg,
12     "Dispatch function %d is unimplemented",
13     pac.func);
14
15   notify(errMsg);
16
17   return (10);
18 }
```

B.58 MessageHandler.h

```
1  // MessageHandler.h
2  // Jamie Gentry
3
```

```
#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 []
    {
      return (stringCopy(messages[n].id));
    }

  char* getInstantiatedMsg(int n, char** vars,
          int nVars) const;
  // Requires: n is a valid message number, vars
  //           is an array of strings, and nVars
  //           is the number of elements in vars
  // Effects: Returns a string obtained by
  //          instantiating the elements vars in
  //          the message template number n. (This
  //          is explained in more detail
  //          in the YASP documentation). The
  //          result can be destroyed using
  //          delete []. If there is a problem,
  //          returns NULL

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"
```

```
8
9  MessageHandler::MessageHandler(const
10                 char* fileName):
11                 status(1),
12                 nMsgs(0),
13                 messages((Message*)
14               NULL)
15 {
16   // Open file
17   FILE* msgFile = fopen(fileName, "r");
18   if (msgFile == (FILE*) NULL) {
19     status = 0;
20     return;
21   }
22
23   // The number of messages we have room for so
24   // far
25   int messageCapacity
26                  = 20;
27   messages       = (Message*)
28     calloc(size_t(messageCapacity),
29       sizeof (Message));
30   if (messages == (Message*) NULL) {
31     fclose(msgFile);
32     return;
33   }
34
35   // Get "header" line
36   char* line     = new char[100];
37   fgets(line, 100, msgFile);
38   delete [] line;
39
40   // Get rest of lines
41   do {
42     char* line          = new char[100];
43     if (fgets(line, 100, msgFile) == NULL)
44       break;
45
46     // Get rid of carriage return at the end of
47     // the line
48     if (line[strlen(line) - 1] == '\n')
49       line[strlen(line) - 1] = '\0';
50
51     messages[nMsgs].id = stringCopyUntil(line, 8);
52
53     messages[nMsgs].content
54                         = stringCopy(line, 10);
55     nMsgs++;
56
57     // See if we've ran out of room for messages
58     if (nMsgs == messageCapacity) {
59       messages = (Message*)
60    realloc((void*) messages,
61      size_t(messageCapacity + 20) *
62      sizeof (Message));
63       if (messages == (Message*) NULL) {
64    fclose(msgFile);
65    return;
66       }
67       else
68    messageCapacity += 20;
69     }
70   } while (1);
71
72   // Resize messages...
73   messages       = (Message*)
74     realloc((void*) messages,
75       (size_t) nMsgs * sizeof (Message));
76   if (messages != (Message*) NULL)
77     status = 1;
78
79   fclose(msgFile);
80 }
81
82 MessageHandler::~MessageHandler()
83 {
84   free(messages);
85 }
```

```
86
87 int MessageHandler::operator!() const
88 {
89   return (status == 0);
90 }
91
92 char* MessageHandler::getInstantiatedMsg(int n,
93     char** vars, int nVars) const
94 {
95   // Get a copy of the template
96   char* templ = getMsg(n);
97
98   // The return string
99   char* ret   = (char*)
100    calloc(size_t(100), sizeof (char));
101  if (ret == (char*) NULL) {
102    delete [] templ;
103    return (ret);
104  }
105
106  // The size of the return string
107  int retSize = 100;
108
109  int nextVar = 0;
110
111  // Where we are in the source string
112  char* pSrc  = templ;
113
114  // Where we are in the return string
115  char* pRet  = ret;
116
117  do {
118    if (*pSrc != '%') {
119      // This is a non-control character
120      *pRet = *pSrc;
121      pSrc++;
122      pRet++;
123
124      if (pRet - ret >= retSize) {
125  ret = (char*)
126    realloc((void*) ret,
127       size_t(retSize + 100) *
128       sizeof (char));
129  if (ret == (char*) NULL) {
130    delete [] templ;
131    return (ret);
132  }
133  retSize += 100;
134      }
135    }
136    else {
137      // We have a '%'
138
139      // Check for %%
140      if (pSrc[1] == '%') {
141  *pRet = '%';
142  pSrc += 2;
143  pRet++;
144  if (pRet - ret >= retSize) {
145    ret = (char*)
146      realloc((void*) ret,
147        size_t(retSize + 100) *
148        sizeof (char));
149    if (ret == (char*) NULL) {
150      delete [] templ;
151      return (ret);
152    }
153    retSize += 100;
154  }
155      }
156      else {
157  for (int i = 0; i < strlen(vars[nextVar]);
158       i++) {
159    *pRet = vars[nextVar][i];
160    pRet++;
161    if (pRet - ret >= retSize) {
162      ret = (char*)
163        realloc((void*) ret,
```

```
164           size_t(retSize + 100) *
165           sizeof (char));
166       if (ret == (char*) NULL) {
167         delete [] templ;
168         return (ret);
169       }
170       retSize += 100;
171     }
172   }
173   if (nextVar < 10)
174     pSrc += 2;
175   else if (nextVar > 10 && nextVar < 100)
176     pSrc += 3;
177   nextVar++;
178       }
179     }
180   } while (*pSrc != '\0');
181
182   *pRet = '\0';
183
184   char* msg = stringCopy(ret);
185   free((void*) templ);
186   free((void*) ret);
187   return (msg);
188 }
```

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
52  int openWrite();
53  // Requires: this is not already open
54  // Modifies: this
55  // Effects: Opens this for writing. If
56  //          successful, returns true; otherwise,
57  //          returns false
58
59  void close();
60  // Modifies: this
61  // Effects: Closes this
62
63  inline int atEOF() const
64    // Effects: Returns true iff we are at the end
65    //          of the file
66    {
67      return (isEOF);
68    }
69
70  char* get();
71  // Modifies: this
72  // Effects: Returns a line from this, including
73  //          the carriage return. If a line
74  //          cannot be read returns NULL. The
75  //          result can be destroyed using delete
76  //          []
77
78  char* get(int len);
79  // Modifies: this
80  // Effects: Returns at most len characters from
81  //          this. If already at end of file,
82  //          returns NULL. The result can be
83  //          destroyed using delete []
84
85  inline size_t readLen(void* dest, size_t len)
86    // Requires: File is open for reading
87    // Modifies: dest
88    // Effects: Tries to read len characters from
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
96  int put(char* data, int len);
97  // Requires: data contains at least len
98  //           characters
99  // Modifies: this
100 // Effects: Writes len characters from data to
101 //          the file. Returns true if
102 //          successful, false otherwise
103
104 private:
105
106 // The filename
107 char* name;
108
109 // true iff file is open
110 int isFileOpen;
111
112 // The mode the file is open in
113 openMode mode;
114
115 // The input file
116 FILE* readFile;
117
118 // The output file
119 FILE* writeFile;
120
121 // A buffer for our reads and writes
122 char* buffer;
123
124 // The size of buffer
125 int bufferSize;
126
127 // true iff we are at the end of the file
```

```
128   int isEOF;
129};
```

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)
```

```
73      return;
74
75    switch (mode) {
76    case read:
77      fclose(readFile);
78      readFile = (FILE*) NULL;
79      break;
80
81    case write:
82      fclose(writeFile);
83      writeFile = (FILE*) NULL;
84      break;
85
86    case append:
87      // This is not yet handled...
88      break;
89    }
90
91    isFileOpen = 0;
92
93    return;
94  }
95
96  char* File::get()
97  {
98    // If we're at the end of the file, don't even
99    // try...
100   if (isEOF)
101     return ((char*) NULL);
102
103   // Make sure file is properly opened...
104   if (!isFileOpen || mode != read)
105     return ((char*) NULL);
106
107   for (int i = 0; 1; i++) {
108     int rc = fgetc(readFile);
109     if (rc != EOF) {
110       buffer[i] = char(rc);
111       if (bufferSize == i) {
112   buffer  = (char*)
113     realloc((void*) buffer,
114       size_t(bufferSize + 100) *
115       sizeof (char));
116   if (buffer == (char*) NULL)
117     return (buffer);
118   bufferSize += 100;
119       }
120       if (buffer[i] == '\n') {
121   buffer[i + 1] = '\0';
122   return (buffer);
123       }
124     }
125     else {
126       isEOF = 1;
127       if (i == 0)
128   return ((char*) NULL);
129       else {
130   buffer[i] = '\0';
131   return (buffer);
132       }
133     }
134   }
135 }
136
137 char* File::get(int len)
138 {
139   char* ret = new char[len + 1];
140   if (fgets(ret, len + 1, readFile) ==
141       (char*) NULL)
142     return ((char*) NULL);
143   return (ret);
144 }
145
146 int File::put(char* data, int len)
147 {
148   if (mode != write)
149     return (0);
150
```

```
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}
```

B.62 notify.h

```
1  // notify.h
2  // Jamie Gentry
3
4  void notify(char* msg);
5  // Modifies: stderr
6  // Effects: Prints an error message (including
7  //          msg) to stderr, and exits.
```

B.63 notify.cc

```
1  // notify.c
2  // Jamie Gentry
3
4  #include <iostream.h>
5  #include <stdlib.h>
6  #include "notify.h"
7
8  void notify(char* msg)
9  {
10   cerr << "ERROR: " << msg << endl;
11   cerr << "Contact the AthenaMuse consortium" <<
12     endl;
13
14   exit(-1);
15
16   // Does not really return...
17   return;
18 }
```

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"
```

```
6
7  char* pubIDFileName(const char* name)
8  {
9    for (int i = 0; i < pubIDsLen; i++) {
10     if (strcmp(pubIDs[i].name, name) == 0)
11       return (pubIDs[i].fileName);
12   }
13   return ((char*) NULL);
14 }
```

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
Version (IRV)//ESC 2/5 4/0",
18    ISO_646_1983_AHLID}
19 };
20
21 int charsetID(const char* name);
22 // Requires: name is a string
23 // Effects: Returns the ID corresponding to the
24 //          character set name. If name is
25 //          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 < IDsLen; 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)
```

```
{
  return (a->p);
}

#else

inline void*& ADLENAddress(ADLEN& a)
{
  return (a._.p);
}

inline void*& ADLENAddress(ADLEN* a)
{
  return (a->_.p);
}

#endif

inline int& ADLENLen(ADLEN& a)
{
  return (a.len);
}

inline int& ADLENLen(ADLEN* a)
{
  return (a->len);
}

inline void*& AHLAddress(AHL* a)
{
  return (a->p);
}

inline void*& AHLAddress(AHL& a)
{
  return (a.p);
}

inline int& AHLIdent(AHL& a)
{
  return (a.i);
}

inline int& AHLIdent(AHL* a)
{
  return (a->i);
}

inline char* ETYName(ETY& e)
{
  return ((char*) ADLENAddress(e.name));
}

inline char* ETYName(ETY* e)
{
  return ((char*) ADLENAddress(e->name));
}

inline unsigned int OFLENOffset(OFLEN& o)
{
  return (o.rel);
}

inline unsigned int OFLENOffset(OFLEN* o)
{
  return (o->rel);
}

inline unsigned int OFLENLen(OFLEN& o)
{
  return (o.len);
}

inline unsigned int OFLENLen(OFLEN* o)
{
  return (o->len);
}

```

```
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
103inline char* stringCopy(const char* s, int n)
104     // Requires: cc is a string, n >= 0.
105     // Effects: Returns a copy of cc from
106     //          character n on. If n is greater
107     //          or equal to the size of cc,
108     //          returns NULL. The result can be
109     //          destroyed with delete [].
110{
111  return (stringCopy(s + n));
112}
113
114char* 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
122inline 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
131int 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
137char* 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
144int 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
151char* stringCat(const char* s1, const char* s2);
152// Requries: s1, s2 are strings
153// Effects: Concatenates string s1 to s2, and
154//          returns the result.  The result
155//          can be destroyed with delete [].s
156
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
```

```
15 char* stringCopyUntil(const char* s, int n)
16 {
17   if (n >= strlen(s))
18     return ((char*) NULL);
19   char* ret = stringCopy(s);
20   ret[n]    = '\0';
21   return (ret);
22 }
23
24 char* stringCat(const char* s1, const char* s2)
25 {
26   int ls1        = strlen(s1);
27   int ls2        = strlen(s2);
28
29   char* ret = new char[ls1 + ls2 + 2];
30
31   for (int j = 0; j < ls1 + ls2 + 2; j++)
32     ret[j] = '\0';
33
34   for (int i = 0; i < ls1; i++)
35     ret[i] = s1[i];
36
37   for (i = 0; i < ls2; i++)
38     ret[ls1 + i] = s2[i];
39
40   ret[ls1 + ls2] = '\0';
41
42   return (ret);
43 }
```

B.70 Makefile

```
1  # Makefile
2  # Jamie Gentry
3
4  # Directories
5  includeDir = $(HOME)/include/YASP
6  libDir     = $(HOME)/lib
7
8  # Trash
9  rubbish    = *.o *~ core test out \#*
10
11 # Application object files
12 appObjs    = parseSGMLFile.o test.o dispatch.o \
13               unimplFn.o notify.o SGMLObject.o \
14               sGet.o find.o random.o File.o msg.o \
15               MessageHandler.o xEnOp.o etySt.o \
16               xEnRd.o charsetID.o pubIDFileName.o \
17               xEnCl.o utOpW.o utWrt.o utRd.o \
18               DTDSt.o DTD.o eltSt.o SGMLElement.o \
19               eltNd.o proSt.o SGMLDocument.o
20
21 # Compiler Flags
22 CPPFLAGS    = -I/$(includeDir) -DAIX_SYS -DGNUCC
23 LDFLAGS    = -L$(libDir)
24
25 .cc.o:
26    g++ $(CPPFLAGS) $(CFLAGS) -Wall -c $< \
27            -o $@
28
29 ########## Application ##########
30 test: $(appObjs)
31    g++ $(LDFLAGS) -o $@ $(appObjs) -lyasp
32
33 test.o: test.cc parseSGMLFile.h DTD.h \
34           SGMLDocument.h SGMLElement.h
35
36 parseSGMLFile.o: parseSGMLFile.cc \
37           parseSGMLFile.h DTD.h SGMLDocument.h \
38           dispatch.h SGMLElement.h \
39           MessageHandler.h
40
41 dispatch.o: dispatch.cc dispatch.h unimplFn.h \
42           notify.h sGet.h random.h find.h msg.h \
43           sFre.h xEnOp.h etySt.h xEnRd.h decSt.h \
44           etyNd.h xEnCl.h utOpW.h utWrt.h utCl.h \
45           utOpR.h utRd.h proSt.h DTDSt.h DTDNd.h \
```

```
46           proNd.h dapNd.h eltSt.h eltNd.h \
47           docNd.h pi.h parseSGMLFile.h text.h \
48           re.h skip.h
49
50 unimplFn.o: unimplFn.cc unimplFn.h notify.h
51
52 notify.o: notify.cc notify.h
53
54 SGMLObject.o: SGMLObject.cc SGMLObject.h
55
56 sGet.o: sGet.cc sGet.h random.h
57
58 find.o: find.cc find.h notify.h parseSGMLFile.h \
59           File.h charsetID.h pubIDFileName.h
60
61 random.o: random.cc random.h
62
63 File.o: File.cc File.h random.h
64
65 msg.o: msg.cc msg.h
66
67 MessageHandler.o: MessageHandler.cc \
68           MessageHandler.h random.h File.h
69
70 xEnOp.o: xEnOp.cc xEnOp.h File.h File.h random.h
71
72 etySt.o: etySt.cc etySt.h random.h
73
74 xEnRd.o: xEnRd.cc xEnRd.h File.h random.h
75
76 charsetID.o: charsetID.cc charsetID.h
77
78 pubIDFileName.o: pubIDFileName.cc pubIDFileName.h
79
80 xEnCl.o: xEnCl.cc xEnCl.h File.h
81
82 utOpW.o: utOpW.cc utOpW.h File.h random.h
83
84 utWrt.o: utWrt.cc utWrt.h File.h random.h
85
86 utRd.o: utRd.cc utRd.h random.h File.h
87
88 DTDSt.o: DTDSt.cc DTDSt.h DTD.h parseSGMLFile.h
89
90 DTD.o: DTD.cc DTD.h
91
92 eltSt.o: eltSt.cc eltSt.h SGMLElement.h random.h \
93           parseSGMLFile.h
94
95 SGMLElement.o: SGMLElement.cc SGMLElement.h \
96           SGMLObject.h random.h
97
98 eltNd.o: eltNd.h eltNd.cc parseSGMLFile.h
99
100proSt.o: proSt.cc proSt.h parseSGMLFile.h
101
102SGMLDocument.o: SGMLDocument.cc SGMLDocument.h \
103           SGMLElement.h
104
105########### Clean ##########
106
107clean:
108   $(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]

```
1  <!DOCTYPE anth [
2     <!ELEMENT anth   - - (poem+)>
3     <!ELEMENT poem   - - (title?, stanza+)>
4     <!ELEMENT title  - O (#PCDATA)>
5     <!ELEMENT stanza - O (line+)>
6     <!ELEMENT line   O O (#PCDATA)>
7  ]>
8
9  <anth>
10    <poem>
11       <title>The SICK ROSE
12       <stanza>
13          <line>O Rose thou art sick.
14          <line>The invisible worm,
15          <line>That flies in the night
16          <line>In the howling storm:
17       <stanza>
18          <line>Has found out thy bed
19          <line>Of crimson joy:
20          <line>And his dark secret love
21          <line>Does thy life destroy
22    </poem>
23 <!-- This is taken from the TEI guide -->
24 </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.

```
1  ----------
2  <Document>
3  <ANTH>
4  <POEM>
5  <TITLE>
6  </TITLE><STANZA>
7  <LINE>
8  </LINE><LINE>
9  </LINE><LINE>
10 </LINE><LINE>
11 </LINE></STANZA><STANZA>
12 <LINE>
13 </LINE><LINE>
14 </LINE><LINE>
15 </LINE><LINE>
16 </LINE></STANZA></POEM></ANTH>
17 </Document>
18 ----------
```

These are the expected results.