

Dartmouth College Computer Science Technical Report  
PCS-TR97-309

## **ASML: Automatic Site Markup Language 1.03**

**Dartmouth Experimental Visualization Laboratory (DEVLAB)**

February, 1997

Charles B. Owen, Fillia Makedon, Glen Frank, Michael Kenyon

### **Abstract**

*Creation of large and complex World Wide Web sites is hampered by the “page at a time” approach of many tools and the programming knowledge and custom software development required for automated solutions. This report describes the development of the Automatic Site Markup Language (ASML). ASML is a new markup language designed to produce large, complicated web sites which can include dynamic content. ASML extends HTML with new, high-level features while still preserving complete compatibility with common browser and server technologies. It has powerful indexing and searching facilities, and enables the automatic translation of document formats. Most importantly, ASML provides HTML-like features at the site level rather than just the page level.*

## Table of Contents

1. Introduction.....	4
2. Related Work.....	5
3. ASML Capabilities.....	7
3.1 Logical modularization of content.....	7
3.2 Site-level content generation.....	7
3.3 Multi-targeting content.....	8
3.4 Location independence.....	8
3.5 Derivative sites.....	9
3.6 Automatic tables of content.....	9
3.7 Content search and presentation.....	10
3.8 Simple CGI scripting.....	11
4. Example ASML Sites.....	11
4.1 The Prehistoric Archaeology of the Aegean.....	12
4.2 ImageTcl Documentation.....	12
5. ASML Markup Format and Examples.....	14
5.1 Terminology.....	14
5.2 Tag Format.....	15
5.3 Syntactic Features.....	17
5.4 Consequences of the ASML Markup Format.....	18
6. Templates in ASML.....	18
7. The ASML Tags.....	19
7.1 Tag: {append}.....	19
7.2 Tag: {base}.....	20
7.3 Tag: {define}.....	20
7.4 Tags: {definelist} and {item}.....	21
7.5 Tag: {else}.....	22
7.6 Tag: {elseif}.....	22
7.7 Tag: {foreach}.....	22
7.8 Tag: {formget}.....	23
7.9 Tag: {highlight}.....	23
7.10 Tags: {if} {else} {elseif}.....	23
7.11 Tag: {img}.....	24
7.12 Tag: {import}.....	24
7.13 Tag: {include}.....	25
7.14 Tag: {index}.....	25
7.15 Tag: {item}.....	25

7.16 Tag: <i>{page}</i> .....	25
7.17 Tag: <i>{search}</i> .....	26
7.18 Tag: <i>{section}</i> .....	26
8. Template Expansion .....	26
9. Searching and Indexing in ASML .....	28
9.1 Tag: <i>{index}</i> .....	28
9.2 Enabling and disabling indexing.....	28
9.3 Tag: <i>{search}</i> .....	28
9.4 Tag: <i>{highlight}</i> .....	29
9.5 Search Methodology.....	30
10. ASML as a CGI Language .....	30
10.1 Security Issues of ASML as a CGI Language.....	32
10.2 Future CGI issues.....	32
11. RTF Import Type.....	32
12. Invoking ASML.....	33
13. Future Work.....	33
14. Acknowledgments .....	33
15. Conclusion .....	34

## 1. Introduction

The World Wide Web (WWW or the web) is a component of the Internet which allows for simple and efficient electronic publication of documents. Automatic Site Markup Language (ASML) is a new markup language designed to automate the construction of World Wide Web sites. It centralizes functionality, decreases duplication of effort, and supplants most uses of scripting languages and custom programming in site development. It has already been utilized to construct a large historic site, The Prehistoric Archaeology of the Aegean [32] as well as many smaller projects at the Dartmouth Experimental Visualization Laboratory (DEVLAB) [7], including the ASML pages themselves [24].

There are many approaches to the creation of resources for the World Wide Web. Documents are electronically *served* in many forms including raw text, hypertext markup language (HTML), graphic images, and executable programs. The source mechanism for content is either transmission of static files or dynamic generation of content from programs (often called scripts). By far the most common mechanism is the delivery of static files in HTML format with enclosures (in-line components) usually in the form of images.

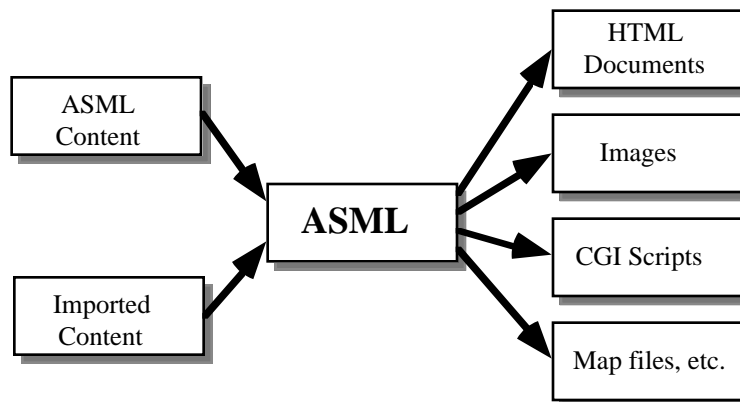


Figure 1 - Use of ASML at a site

Web sites provides detailed multimedia presentations. These presentations must be created (authored) using any one of several approaches. Often content is created by simply writing HTML manually using a text editor. Products also exist which allow for authoring pages without knowledge of HTML in a form similar to working with a word processor. This “page at a time” approach views a site as a collection of discrete hypertext documents. DEVLAB experiences with web site development projects, such as the *DAGS’95 Conference Proceedings* [8], *The Olympics in the Ancient Hellenic World* [1], and *The Prehistoric Archaeology of the Aegean* [32] have clearly exhibited the shortcomings of the page-at-a-time approach. Rice, et. al. discuss similar experiences creating large WWW sites, wherein they state that “trying to correct many of our early errors has become an extremely time-intensive job” [28]. They specify a design process that includes defining structure and format prior to page creation, but clearly found that such decisions cannot be expected to be perfect and later revision can require editing every page in a site. Creating the pages of a site one at a time is similar to creating a word processing document one page at a time.

Were this report created this way, the authors would be forced to edit every single file wherever a change was made to overall document format.

One common solution to this problem has been to “build” sites using custom scripts in computer languages such as Perl [32]. This cumbersome approach has been used in the DEVLAB to create smaller sites for internal use. Other automated solutions also exist; for example, the web site for the DAGS’95 proceedings was built using an Apple Hypercard stack [8]. The Olympics in the Ancient Hellenic World site was created using scripts based on *m4*, a macro language common in Unix systems, and Perl scripts for dynamic content generation [1]. Many other projects in the DEVLAB have been done using the Perl language entirely. These projects assumed a knowledgeable programmer with experience in programming with the Perl language and the need for *custom software development for every single project*. Most DEVLAB projects involve novice programmers as well as students not associated with Computer Science. ASML was developed to better support authoring in such an environment.

ASML incorporates many features which automate the development of sites. It can be thought of as a “markup language for markup languages.” Figure 1 illustrates the use of ASML in a site. Site content can originate in ASML, HTML, raw text, RTF and other formats. Static content served by the World Wide Web server consists of HTML pages, so ASML *incurs no performance penalty*. Also note that the conventional Perl script does not exist in this diagram (though it is still available for more complicated applications). ASML sites can often be “Perl-free.” ASML provides most automatic content generation features which would otherwise require the use of Perl. As an example, the Prehistoric Archaeology of the Aegean site can generate over 500 pages containing unique images. These pages are generated on demand based on the contents of an image database rather than stored for presentation. This content is generated by a single ASML file which serves as a CGI script.

## 2. Related Work

There have been many approaches to automating the development of web sites and to site level authoring. The most common approach is the application of scripting languages. The scripting language approach is based on the creation of custom scripts which either edit an existing site (traversing all pages and changing a background image for example), generate a site from data or directly from a program, or generate content “on-the-fly” as CGI scripts. The most prevalent scripting language on the WWW is the Perl language due to its wide availability [32]. Other systems include Frontier from UserLand software, a popular Macintosh Environment [11]. Use of these systems is summed up by Christopher Hall and Carey Tews in a MacWeek review of Frontier: “Scripting in Frontier is by no means a job for beginners...” [13].

Many graphical HTML editor tools are available. Carl Davis has written a large list with reviews [9]. There are many problems with the existing field of editors, not the least of which being the fluid nature of the WWW. As new browser versions come into existence and new HTML extensions become popular it is difficult for editors to keep up. Few editors support site-level features, the majority being designed for single page editing. Macromedia Backstage Internet Studio 2.0 includes site management features which allow management of groups of pages at a site and automatic verification of link consistency, but no common page format or graphical element support [19].

Microsoft FrontPage 97 has similar site mapping and organization facilities [20]. Web Project Explorer from Haht Software incorporates a system of “Clips”, which are very similar to ASML templates [12]. However, clips are static macros which contain fixed content, as distinct from ASML templates which can be parameterized. As in most of the larger commercial systems, Web Project Explorer has site level organization and visualization tools, but retains a page-at-a-time editing and composition approach.

Bacher and Ottmann promote the idea of “authoring on the fly” [2]. In this paradigm a class lecture presentation is recorded and automatically translated to a complete Hyper-G presentation. This is an example of a highly application specific approach to site level authoring. Hyper-G (now HyperWave) is an alternative to the World Wide Web wherein the server provides site level facilities for management of internal link consistency, indexing, and searching [21]. Lennon and Maurer discuss site level authoring of large systems in the Hyper-G context in [17]. While Hyper-G is a significant improvement over the WWW in many ways, it is not currently available as a large scale alternative and the site level features are primarily concerned with link consistency and content organization, not format. W3DT WebDesigner implements a site level structure including templates defining common page components and automatic generation of site organization [3]. However, WebDesigner generates page templates which must then be edited to include content. Later changes to the template structure are not supported. None of these systems support automatic re-targeting of identical content in multiple formats.

One approach to site authoring is to author in some environment other than HTML, then convert to HTML. ASML can be compared to HTML conversion utilities (though it has many more features). An example is authoring in Latex, a common document preparation system, then converting the Latex to HTML using the Latex2HTML tool [10,16]. InterBook allows for editing in Microsoft Word and conversion to HTML [5]. Hidden text is used to provide annotation capabilities which InterBook can understand and utilize in the translation process (for links, annotations, etc.). Such tools are not optimized for HTML authoring. They are primarily designed to facilitate conversion of existing content. Because the mechanics of HTML are not available in the alien environment (as they are in ASML), conversion tools limit the page designer’s flexibility. Converters exist for a wide variety of environments include most commercial word processors and page layout programs.

There are many application areas where site-level authoring and the ability to evolve format are very important. One example is on-line museums [18]. Such sites can involve huge amounts of material. A page-at-a-time authoring approach would be cumbersome and later revision (remodeling) time consuming. Another application area is education [4]. Many educators wish to place class notes and other materials on the WWW as a dissemination mechanism, leading to a large number of education specific approaches to site-level authoring and automatic content generation. Indeed, this is the basis for several of the DEVLAB projects such as the Prehistory Archaeology of the Aegean [32]. ANDES Text Markup Language (ATML) has been developed at the University of Southern California [15]. ATML is a higher level markup language, much like ASML, but is specific to the ANDES distant education system. The HTML Course Creator (HCC) has been developed at the United States Military Academy [6]. This system is described as a “course-level” authoring system. It has a graphical user interface and is specifically targeted at courseware development. A system embodying many of the ideas in ASML is CourseWeaver [27].

CourseWeaver is an Apple Hypercard-based system designed for development of courseware for the web. It uses a custom markup similar to that of ASML, provides for multi-targeted output, and handles content input translation. It is tightly directed at courseware production and enforces a specific structure on a site.

### **3. ASML Capabilities**

ASML has many powerful features which automate the generation of large WWW sites. This section describes some of these features and places them in the context of site development. ASML is an integrated tool. All of the following categories of features are supported by a single environment and a single program. This is a major advantage over the application of a collection of disjoint tools.

#### ***3.1 Logical modularization of content***

ASML allows the generation of a complete site from a single ASML document. Such a site may consist of hundreds of pages. As in any large task, breaking the authoring task into smaller modules is very important. Smaller modules limit visibility to only what is of interest to the author at the time and allow multiple authors to participate in development. The modularization in traditional web-sites is *physical*. It is directly related to physical pages and groupings, typically in directories. This often will not be the best organization of work for users. ASML allows *logical modularization*. An ASML document can be broken down into whatever form best suits the user. An example might be grouping all common format elements in a single file, while a set of related pages group in another. Typically a master document will combine all of this content.

ASML always builds a complete site at processing time. However, it is anticipated that partial site reconstruction could be performed based on file dependency determination. Only content which is based on changed files need be reconstructed. The ability to logically modularize content is very useful in such an environment. See section 13 for more details on this proposed capability.

#### ***3.2 Site-level content generation***

The most important feature of ASML is its abstraction of a web site as a single large hypertext document. This is distinct from the traditional page-at-a-time authoring. The terminology *site* in this context refers to all content considered to be a single presentation, not necessarily all content on a specific server. As an example, the Prehistory Archaeology of the Aegean is considered a *site* even though it resides on the same server as several other major projects.

Most web sites have large amounts of redundancy. Typically a navigation method and a basic page appearance is enforced at the site level in order to achieve a consistent user interface. Common graphic elements quickly become resident in the browser cache and are instantly available, so such elements are normally used as much as possible in order to improve performance of the site. This redundancy can be a major impediment to site reformatting. Ideally, the web site development process would start by defining page formats, navigation, and site structure. The DEVLAB has **never** been able to define this content at the beginning of a project. Content acquisition proceeds in

parallel with graphic development, parameters change as new content appears, and there's always someone who looks at the site and suggests a change after it is finished. Creating static patterns for all pages is not effective in that content already produced must be edited.

ASML allows all common content to be defined as *templates*. Templates are referenced with a simple tag which *expands* to the common content. If a common element is changed, only the template definition need be changed. The change will automatically propagate throughout the site. This is true for not only "tops and bottoms" of pages, the most common elements, but also for elements in the page, graphical separators, local tables of contents, and more.

The site terminology used in this document assumes a set of related pages is a site. A server many contain many sites using this definition. The term *server-site* is used to represent all content on a server. ASML does not exclude the sharing of included content among multiple sites on the same server (or even on different servers). The only restriction is that included content be available in the file system. This makes possible common content at the server-site level as well. A server-site may wish to define common elements for page organization and server-site level navigation. These elements can be shared using ASML.

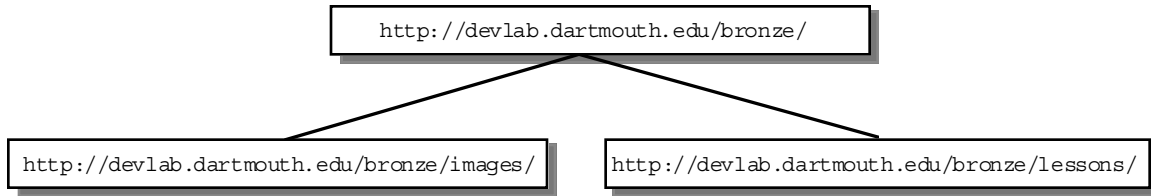
### ***3.3 Multi-targeting content***

In some cases the actual site content can be redundant. Often a site must support *multiple targets*. An example is the issue of *frames*, a physical segmentation of the browser window into independently manageable areas. Frames are a controversial topic on the web today [23]. Many browsers do not support them and many users prefer not to use them. However, they can be a significant user interface improvement. Sites must either choose to use frames, choose not to use them, or provide all content in two formats.

*Multi-targeting* is providing content in more than one more format. A typical usage of ASML is to place actual content into templates. When used this way, multiple pages can be produced which contain that content, effectively providing multiple versions of a site. Other examples of multitargeting include providing longer documents as a set of linked pages and also as a larger document suitable for printing or providing extended and condensed content.

### ***3.4 Location independence***

Most web sites are subdivided into a hierarchical structure of directories and subdirectories which mimics the underlying server file system. This structure is often modeled as a *tree*. Figure 2 illustrates a simple site tree. Dividing a site up in this manner has several advantages. Content in a subdirectory is grouped under the subdirectory name, smaller directories can be searched faster by the server, and common elements can be grouped together. A problem occurs when content in one subdirectory must reference another. As an example, a page in the `lessons` subdirectory may need an image in the `images` subdirectory. Relative addressing can be used to access this image: ``. However, this method may present problems if the enclosing page is moved. Also, though many of the popular the browsers have significantly improved, some older browsers consider `/bronze/images/line.gif` and `/bronze/lessons/../../images/lines.gif` as different files, subject to duplicate caching.



**Figure 2 - Tree structured site layout**

*Location independence* is the ability to produce a site at any required server address. This is a basic feature of ASML. Using the {base} tag, content can be relocated at will. In several DEVLAB projects this feature has been used to maintain simultaneous development and published sites for the same project. The published site was updated occasionally when the content reached stable milestones. The development site contained content in incomplete stages.

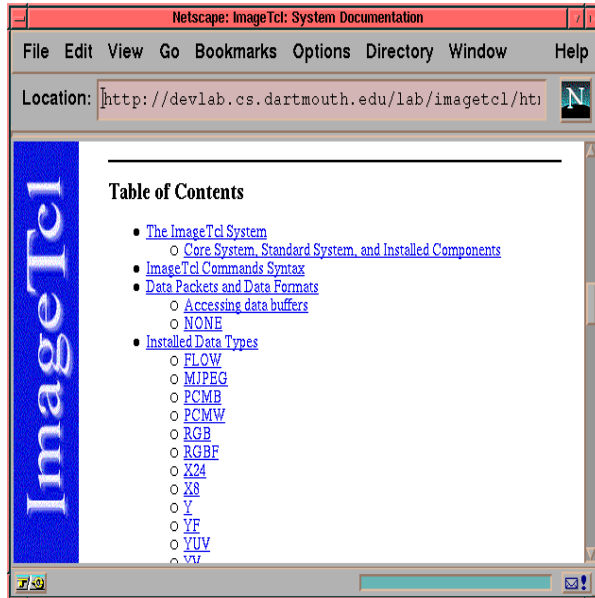
### **3.5 *Derivative sites***

The Prehistory Archaeology of the Aegean site is a *derivative site*. A derivative site is a site wherein the web publication is derived from content in an alien format. This is distinct from a *translated site*, wherein the web publication is a translation of content from an alien format. In the Prehistory Archaeology of the Aegean, content consists of a large set of notes by Prof. Jeremy Rutter of Dartmouth College. Prof. Rutter had already provided these notes to Indiana University, wherein the original Microsoft Word documents were translated to HTML and published [30]. This process produced an effective presentation of the material, but is not able to evolve as the material evolves. The source material is routinely expanded and revised and Prof. Rutter wishes to maintain it in the word processor format. Updating that site to match the current version of those notes would require either repeating the entire translation process or manually determining all of the changes. These are prohibitive projects.

ASML supports *imported content*, content supplied in forms other than HTML or ASML. In the DEVLAB production of the Prehistoric Archaeology of the Aegean, the content is provided as Rich Text Format (RTF) exported from Microsoft Word. That content is imported into ASML when the site is generated and used to construct the pages. The original Word document becomes a source for compilation of the site. Whenever that document changes the site can be rebuild, completely incorporating the changes. This is of critical importance in any application wherein content routinely changes and must appear as both paper and WWW documents.

### **3.6 *Automatic tables of content***

ASML can be configured to automatically construct tables of contents for an entire site, any group of pages, or the contents of a single page. Figure 3 contains a portion of the table of contents for the documentation for the ImageTcl multimedia development system [25]. This document is developed concurrently with the ImageTcl system. Content is added on a regular basis. When new components are added the ASML documentation files automatically rebuild the table of contents to include them.



**Figure 3 - ImageTcl documentation table of contents**

Construction of new tables of contents is achieved using ASML templates and counters. While the structure is easy to implement, it remains entirely under user control. Hence, multiple simultaneous tables of content are possible, as are views which are table based or use graphical bullets.

### ***3.7 Content search and presentation***

Most larger web sites require indexing. Numerous tools are available for implementation of search engines at any level of a site. However, these tools are often complicated to install and are not easily customized for a particular site. ASML has a built-in search engine which can be easily implemented using only ASML (no Perl scripts are required). Figure 4 illustrates the search page in the Prehistoric Archaeology of the Aegean. This site has three major categories of content: lesson text, images, and bibliographies. A search can examine a single section or the entire site.

Figure 5 is an excerpt from a page in the ImageTcl documentation site. In this site a search field has been added to every page. This is common content (as are all the components shown here), so they are generated by an ASML template.

The ASML search engine is based on an index file for the site which is constructed when the site is built. This file allows for very fast search operations. Search results are obtained very quickly with minimum load on the web server. An information retrieval-based search engine scores pages which match the query and ranks the pages in descending relevance.



Figure 4 - Prehistory Archaeology of the Aegean search page

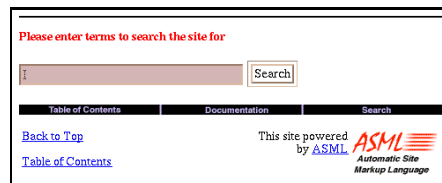


Figure 5 - ImageTcl documentation search

A unique feature of ASML is search highlighting. A page located using the search engine can be accessed with all search terms highlighted in red. In addition, the browser is automatically advanced to the first search term located in the document. A mechanism for sequential browsing of search terms is being studied for possible future inclusion.

### 3.8 Simple CGI scripting

One of the most complicated elements of site development is *Common Gateway Interface* (CGI) scripting. Scripting is required for processing data forms and dynamic content generation. Scripting usually involves programming, a skilled activity. ASML can serve as the scripting environment in many applications. Production of a script is little different from production of a page. The mechanism for acquisition of forms variables is automatic in ASML. All form variables are converted to templates. All issues of CGI protocol are managed by ASML, requiring no user knowledge us the underlying mechanisms.

## 4. Example ASML Sites

Several web sites have been created using ASML. This section describes two of these sites and discusses how ASML simplified site development.

## ***4.1 The Prehistoric Archaeology of the Aegean***

A major web project at the DEVLAB has been the Prehistoric Archaeology of the Aegean [32]. Figure 6 and Figure 7 illustrate several pages from this site. This project and ASML development proceeded concurrently and the design of this site was a major influence on the design of ASML. It also provided a large amount of user testing. It should be noted that the content developers for this site mostly consisted of students with no previous experience in WWW development. The site provides 29 lessons in archaeology and is designed for student use. Over 500 images of archaeological digs, historic locations, and significant artifacts are included in the site. All images are presented in a small browsing format, a larger display format, and the full high-resolution scan size.

This web site incorporates nearly all features of ASML. The site is a derivative site and can be reconstructed as the Microsoft Word content is updated (indeed, it has been updated several times). The content is imported as RTF text. The ASML search engine is an integral feature of the site and four indices are generated. Over 500 images are available in the site. The pages which present these images are generated dynamically by an ASML CGI script. The page layout and navigation mechanisms were changed repeatedly in the process of site development. Both client-side and server-side image maps are used extensively in the system. The server-side image map files are created using ASML. In spite of the fact that this site incorporates more than 100 pages, it is produced by 14 ASML files with a total of only 1759 lines. Over a third of these are a list of the available images.

## ***4.2 ImageTcl Documentation***

The ImageTcl multimedia development system has been developed at the DEVLAB for research in media data analysis [25,26]. This is a large system and has been in development for several years. The documentation for the system was initially a set of Latex documents. However, as the system grew, this document rapidly exceeded 200 pages and a decision to pursue electronic dissemination was reached. The documentation was initially manually converted to HTML.

When ASML became available the documentation was migrated to ASML. The ImageTcl documentation is illustrated in Figure 8. The conversion of the HTML documentation to ASML was done in incremental stages. ASML can be used as a simple wrapper for HTML, so initially the pages were converted to this form and a simple site build page included them. Addition of templates to define common elements was done incrementally and has since become quite extensive. This is a common migration mechanism, illustrating that ASML can be added to an existing project with minimal effort. The advanced features of ASML, such as indexing and templates, immediately become available. The original Latex document had many defined macros to simplify command documentation. This mechanism had been lost in the conversion to HTML. ASML allowed a similar structure to be recreated. Note the common page structure. Each page also includes a search engine query form. The table-based layout of the `imagefft` command parameters illustrated in Figure 8 is constructed using the following ASML templates, all user defined:

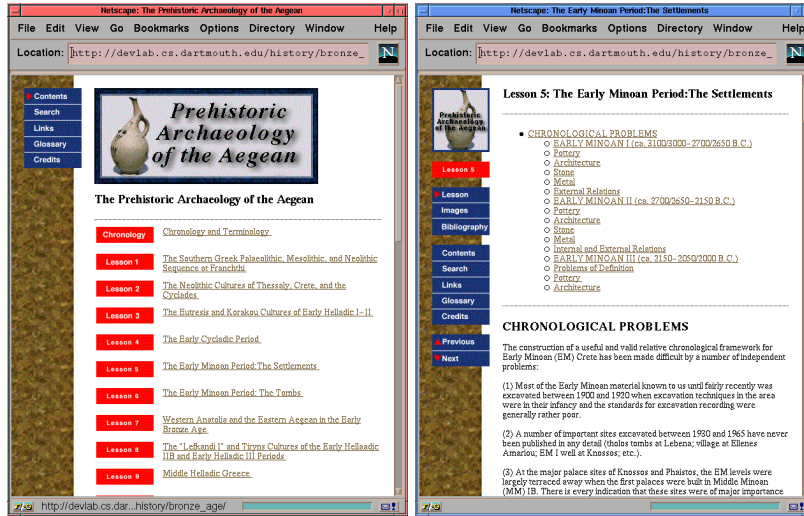


Figure 6 - Prehistoric Archaeology of the Aegean site

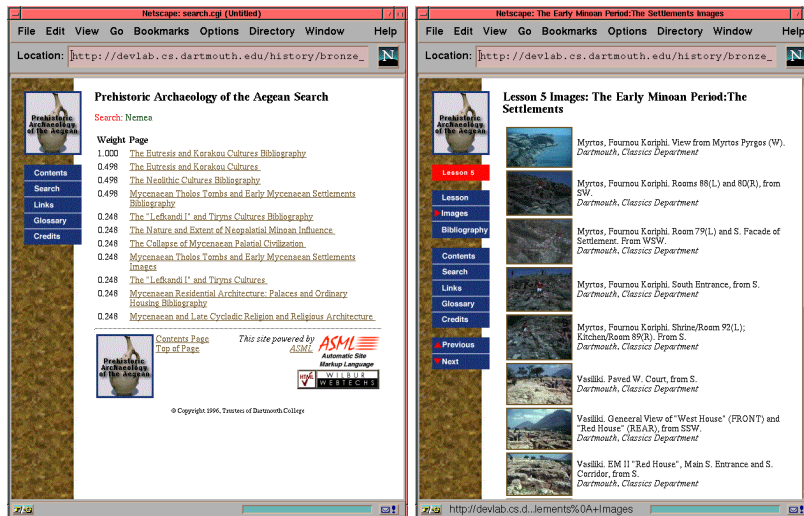


Figure 7 - Prehistoric Archaeology of the Aegean site

```
{cmdstart cmdname="imagefft" cmdtype="Object"}
{cmd left="Header:" right="imagefft.hxx"}
{cmd left="Directory:" right="image"}
{cmd left="Object creation:" right="imagefft object ?commandlist?"}
{sink-support support="YF" chan="0"}
{source-support support="YF" chan="0" pools="No"}
{cmd left="List commands" right="-forward"}
{cmd right="-inverse"}
{cmd right="-sink channel type"}
{cmd right="-source channel type"}
{cmdend}
```

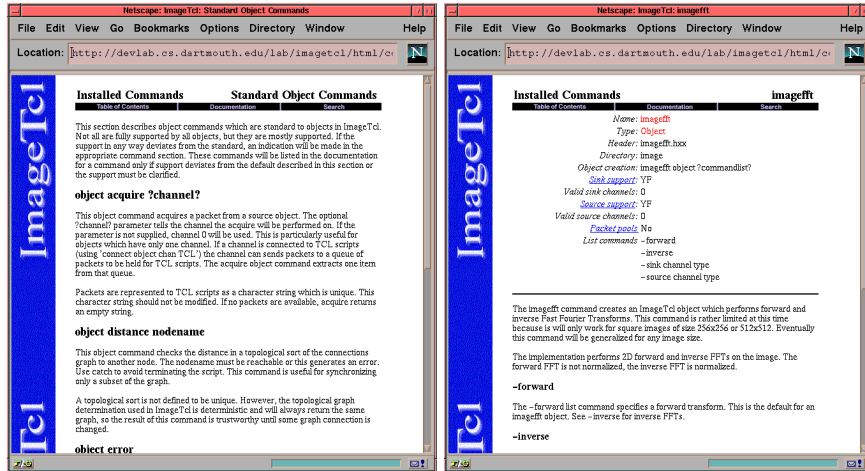


Figure 8 - ImageTcl documentation

This example lists several ASML templates. These are all elements of the site created by the site author, not base ASML features. ASML is a general purpose system; it is not targeted at a specific application. The highly application-specific elements of this example have been produced specifically for this application.

## 5. ASML Markup Format and Examples

This section describes the ASML markup format. As a markup language, ASML incorporates *tags* in a form very similar to HTML. A *tag* is a markup element. HTML and ASML are both tag-based.

### 5.1 Terminology

Several important terms which are essential to understanding ASML are *tag*, *end tag*, *attribute*, *value*, and *expand*. Markup in ASML (and HTML) is in the form of tags. Tags can be stand-alone elements which denote an event in the document or they can be containers. As an example, the HTML `<img>` tag indicates placement of graphic content at a point in the document. This tag is stand-alone, the graphic location being the event. A tag which serves as a container surrounds other content. An example container in HTML is the `<p>` tag which indicates a paragraph. All content between the `<p>` tag and the `</p>` *end tag* is a single paragraph. An *end tag* indicates the end of the contained content. An *attribute* is an option on a tag which directs or enhances its functionality. In HTML the `ismap` attribute of the `<img>` tag indicates that the image is to be used as an image map. A *value* is a string of information associated with an attribute. In both ASML and HTML, attributes are assigned values using the equal sign. An example HTML value would be the assignment to the `src` attribute on the `<img>` tag (``). This value indicates the Uniform Resource Locator (URL) for the image.

ASML *expands* tags when processing an ASML document. Expansion replaces the tag with different content. No ASML tags remain in the output of ASML execution (except when specifically added using the escape mechanism), so browsers need not be modified to support any ASML syntax. Indeed, it is invisible to the WWW user. The term

expand will be used throughout this document to indicate the processing of an ASML tag. Note that some ASML tags have conditional and repetitive processing functions and some tags expand to empty content.

## 5.2 Tag Format

The format for tags and end tags in HTML is:

```
<tagname attribute=value attribute=value>Contained content</tagname>
```

Tags are enclosed in < and >. End tags are enclosed in </ and > and do not support attributes. Values which include characters other than alphanumeric characters, periods, and the underline character must be enclosed in quotation marks. The format for ASML tags and end tags is:

```
{tagname attribute=value attribute=value}Contained content{/tagname}
```

The fundamental design criterion for ASML was that it be *familiar to the HTML user*. Hence, the tag format is virtually identical to HTML. The only difference in notation is that curly braces { and } are used in place of the conventional < and > of HTML. The example below shows a typical HTML <img> tag and the corresponding ASML {img} tag:

```

{img src="/images/bar.gif" alt="---" align="center"}
```

The use of an alternative markup delimiter allows intermingling of ASML and HTML tags while keeping the identity of each type of content intact. Thus, ASML can be thought of as an extension of HTML rather than a replacement. It is easy to add ASML functionality by simply adding ASML tags to an existing HTML document. ASML does not replace HTML and HTML tags can still be used throughout the document.

There are distinct differences in the attribute values of these two tags. These differences are due to the automation incorporated in ASML. Image sizes are determined from the image, so there is no need to include height and width options in the ASML {img} tag. Also, ASML is conscious of a home directory for a site (both on the server and on the local file system), so absolute addressing is relative to the site, not the server (in this case inside the `bronze` directory of the server). This allows the simple and quick retargeting of a site. The following is an example ASML source file:

```
{-- Sample ASML File --}
{base system="/usr/tenaya/devwww/bronze" server="/bronze"}
{page file="index.html"}
<html>
<home><title>Home Page</title></home>
<body>
<h1>Home Page</h1>
</body>
</html>
{/page}
```

The first line of this example is an ASML comment. The comment will not appear in the final HTML file. The comment syntax is described in more detail in section 5.3. The available ASML tags are described in greater detail in section 7. The first tag is the `{base}` tag which defines the location on the site (in this case a single page site) in both the local file system and the server file system. The second tag is the `{page}` tag, indicating that an HTML page is to be generated. The filename for this page will be `/usr/tenaya/devwww/bronze/index.html` because it is in the local file system. The server name is considered to be `/bronze/index.html`. An ASML file can continue with as many page tags as desired. It can also “include” additional ASML files so that a site can be broken into logically related components, as opposed to physically related components such as HTML pages. The relationship between ASML files and HTML pages is usually one to many. As an example, the Prehistoric Archaeology of the Aegean site uses a single ASML file to produce the text, bibliography, and images page for all lessons. This file is only 182 lines long.

The file created by `{page}` is not limited to HTML. Indeed, map files, raw text, and even other ASML files can be created using the page tag. All content between the `{page}` and `{/page}` is written to the file (with ASML processing, of course). If no ASML tags are included, the page tag can be used just to group HTML files into one source file. This would be a simplistic use of ASML.

The following example illustrates ASML’s power in further detail:

```
{-- Sample ASML File --}
{base system="/usr/tenaya/devwww/bronze" server="/bronze"}

{define name="top"}  {-- top of page template --}
<html>
<home><title>{title}</title></home>
<body bgcolor="#ffffff">
{/define}

{define name="bottom"}  {-- bottom of page template}
</body></html>
{/define}

{-- Generate a home page --}
{page file="index.html"}
{top title="Home Page"}
<p>Content in the page</p>
{bottom}
{/page}

{-- Generate another page --}
{page file="other.html"}
{top title="Another Page"}
<p>Content in another page</p>
{bottom}
{/page}
```

In this example a standard page *top* and *bottom* have been defined. The define tag will be described in detail later. In this usage, ASML is similar to a macro language such as `m4`. However, the use of tags keeps macro expansion an explicit operation. Experience using `m4` as a generating language for web pages has shown that implicit expansion of macros can be more trouble than it's worth. As an example, a standard `m4` macro is the `index` macro. This causes difficulties anywhere the word "index" is contained in a page. Conflicts such as this required a large amount of editing and manual quoting to prevent inadvertent macro expansions. The ASML tag format clearly distinguishes all content which will expand.

### 5.3 Syntactic Features

Tags in ASML are of the form: `{tagname attribute=value attribute=value...}`. Values are usually contained in quotation marks, but these can be omitted if there are no spaces in the value or if the value consists only of another tag, a somewhat more liberal policy than in HTML. HTML tags generated automatically by ASML will always be syntactically correct, using quotation marks where necessary. A tag can have 0 or more attributes. Omitting `"=value"` for an attribute is identical to `attribute=""`. In other words, an omitted option is identical to a blank one. Unknown attributes are either ignored, passed through to a matching HTML tag (see the `{img}` tag for an example), or used to define a local template. *End tags* are of the form `{/tagname}`. No abbreviations are allowed for tag names and ASML is *case sensitive*, which means that attribute and tag names are sensitive to case. This is distinct from HTML.

End tags are rarely optional in ASML, they are either used or not used, depending on the tag; the documentation explicitly states when they are used. As examples, `{img}` and `{base}` do not use end tags, whereas `{page}` does. In a few limited cases (such as the `{item}` tag) an end tag is optional. A template expansion indicates an event in the document, the location where the template will appear. A template definition, on the other hand, contains the content for the new template which is being defined and, therefore, requires an end tag.

ASML can contain tags within tag names or attribute values. More detail of this feature will be included in the section on template expansions. The following is a valid ASML tag:

```
{lesson-{lesson}-text}
```

The inner `{lesson}` tag will expand first to create the name of the outer tag. In this example, `{lesson}` might expand to the lesson number and `{lesson-1-text}` expand to the text for lesson one. The ability to produce multiple pages from imported content is valuable in courseware production.

Comment sequences begin with `{--` and end with `--}`. All content between the start and end of a comment sequence, including tags and new-lines, is ignored by ASML. Nested comment sequences are not permitted, and the notation `--}` will always end a comment sequence. Comments are not translated to HTML comments or produced in any way in the output page. HTML comments are often underutilized due to the fact that any user can view the source of a page and see the comments. ASML comments are fully available to the site author, but not the casual user.

The ASML *escape character* is `\`. Preceding any character with `\` indicate that the character is to be treated literally. This allows for braces and backslashes in ASML output. In addition, a `\` at the end of a line is a continuation to the next line. This is particularly useful, since all content between many tags is treated literally in ASML. The above examples would generate a leading blank line in the output content. This can be prevented by placing a `\` after the page tag. Since blank lines are ignored in HTML, adding the escape character is optional and is often omitted, though it can be used to produce denser HTML output if desired. It is common in ASML development project to not look at the output HTML at all.

#### 5.4 Consequences of the ASML Markup Format

ASML is an markup language for markup languages, but is not an SGML-compliant markup language. *Standard Generalized Markup Language* (SGML) is a system for defining markup languages [14]. HTML is an *SGML-compliant markup language* in that it can be described using an SGML *document type descriptor* (DTD). ASML incorporates many features such as repeated expansion, tags inside tags, and conditional expansion which are beyond the structure of SGML. Hence, ASML is not SGML-compliant. However, all content is actually served in HTML, which is SGML-compliant and widely supported. ASML provides higher-level features while retaining a common distribution format.

The markup format used in ASML was a difficult decision. Many alternatives were examined including the `<@tag>` structure used in ATML [15]. However, it was felt that that format would be difficult to distinguish from HTML and would be confusing for novice page authors. The use of braces is not ideal in that it conflicts with the format proposed for *cascading style sheets* [31]. Cascading style sheets are a proposal for format specification in HTML documents. This proposal has been adopted by the Microsoft Explorer browser. As an example, to set the text color of the “H1” elements of an HTML document to blue, the following style sheet entry would be used:

```
H1 { color: blue}
```

The braces can be included using the escape mechanism in ASML:

```
H1 \{ color: blue\}
```

This is clearly not ideal. When ASML was under development the cascading style sheet proposal had been considered dropped. It has only recently been revived. Several syntactic modifications for simplifying this problem are being examined including structures similar to the extended quote in the Perl language or inclusion of raw content from additional files [32]. The ASML markup format also conflicts with JavaScript [22]. Most of the same issues apply in that case.

## 6. Templates in ASML

A fundamental design element of ASML is the *template*. A template is text, possibly with optional “fill-in” content, which is accessed using an ASML tag. The `{define}` tag is a tag which creates templates. Templates can define components of pages which are common throughout the site in a single place (global features of a web site).

Templates also have the power to build tables of contents and other cumulative components using the `{append}` tag. The tags for creating templates are described in section 7. More detail on the *expansion of templates* is included in section 8.

There are two types of templates in ASML, *global templates* and *local templates*. The differentiation between global and local templates can be likened to scope in programming languages. When ASML documents are processed, global templates remain in existence throughout the processing. Global templates are commonly used to define common page elements. Local templates are created by tags and exist only for the duration of the tag expansion. As an example, the procedural tag `{foreach}` creates a local template which is a counter indicating the number of times the *foreach* body has executed. Tag expansion allows local templates to be created in order to fill-in blanks in the template expansion.

Templates must contain complete ASML elements. A lone beginning or end tag is not allowed. As an example, the following is an invalid template in ASML:

```
{define name="pagestart"}{page file="xyz.html"}{/define} ←INVALID
```

This example is invalid because it contains a `{page}` start tag with no matching end tag. This feature prevents the confusion of templates which must be used in pairs in order to achieve proper tag matching and the possibility of using wrong pairs. A better solution for this example would be to create a template for an entire page:

```
{define name="doc-page"}{page file="{url}"}{doc-body}{/page}{/define}
```

This example is better in that the page will be filled in with a body which is defined as a template. HTML content can be split among ASML tags, however. Commonly a “top-of-page” template will be created which includes the `<page>` tag. An “end-of-page” template would then include the `</page>` end tag.

## 7. The ASML Tags

The section describes the ASML tags in detail. The tags are listed in alphabetical order. Many tags have multiple uses or formats. All tags are listed in order, but some tags have a reference to their description in another subsection so as to group the description of related tags.

### 7.1 Tag: *{append}*

The `{append}` tag is used to append to an existing template. It works exactly like the `{define}` tag except that if a template already exists with the name as supplied by the value of the `name` attribute, the context contained in the tag is appended to the template. (See the section 7.3 for more detail on the definition of templates in ASML.)

The `{/append}` end tag is required as is the `name` attribute. The `expand` attribute is supported by the `{append}` tag. Nesting characteristics are as in the `{define}` tag.

## 7.2 Tag: *{base}*

There are two attributes for the base tag: `system` and `server`. HTML sites have an inherent schizophrenetic nature in that there are two addresses by which pages are known. The *system address* is the root directory for the site in the local file system. The *server address* is the root directory for the site in the server file system. They are nearly always different due to the intentionally limited scope of the server view. If both attributes are supplied, their values are used to specify the system and server directories:

```
{base system="/usr/tenaya/devwww/bronze" server="/bronze" }
```

If `{base}` is not included, the defaults are “.” for the system directory and “/” for the server directory. The base tag should be the first tag encountered since many future tags depend on it. Trailing /’s on directory names are ignored. There is no base end tag.

A second use of the base tag is to *expand to these addresses*. Effectively, the tag serves as a pre-defined template. Use of `{base}` alone expands to the system root directory with no trailing slash. In the above example, `{base}` would expand to `"/usr/tenaya/devwww/bronze"`. If the options `server` or `system` are supplied, the tag will expand to the specific base address. Example:

```
{base system="/usr/tenaya/devwww/bronze" server="/bronze" }  
<a href="{base server}/index.html">Home</a>  
<p>This content is rooted at {base system} in the local system.</p>
```

This example will expand to:

```
<a href="/bronze/index.html">Home</a>  
<p>This content is rooted at /usr/tenaya/devwww/bronze in the local system.</p>
```

## 7.3 Tag: *{define}*

The most important feature of ASML is the ability to define templates. The `{define}` tag performs this function. In ASML a template is a name with associated text. The name of a template becomes a new ASML tag which, when used, expands to the associated text.

The attributes for the define tag are `name` and `expand`. The `name` attribute is required. The `{/define}` end tag is also required. All text between `{define}` and `{/define}` is associated with the template name and can be later expanded by using the template name as a tag. Example:

```
{define name="bgcolor"}bgcolor="#ff0000"{/define}  
<body {bgcolor}>
```

Everything between `{define}` and `{/define}` is stored literally. No intervening ASML tags are expanded at all, though this behavior can be overridden using the `expand` attribute.

ASML relies on *deferred expansion* of enclosed tags. In the above example, `{title}` is a template name that is not defined when the template *top* is defined. Using this feature a template can have contents supplied at expansion time. Likewise, it can incorporate complex site building features which will be processed when the tag is expanded. The `{define}` tag gives ASML both template and subroutine features simultaneously.

As mentioned before, it is possible to have tags inside tag names and option values. Hence, the template name can itself be an expansion of another template. Example:

```
{define name="lesson-title-{lesson}" }Lesson Title{/define}
```

Templates can be freely redefined as necessary in ASML. A template must be defined before it is expanded. Templates can also be appended to using the `{append}` tag. ASML processes source files sequentially.

The `expand` attribute on the `{define}` and `{append}` tags overrides the deferred expansion feature, forcing any contained tags in the template to be immediately expanded. This is provided as an alternative behavior and is particularly useful when the same template names are used repeatedly in a document. No value is required for the `expand` attribute. An example of the use of `expand` is:

```
{define name="toc-entry" expand}{title}{/define}
```

In this example the template *toc-entry* will be defined as the contents of the *title* template. If *title* is used as a fill-in element in templates, it may very well change later in the document, so the normal deferred expansion would not yield the expected results. Where `expand` not used the contents of the *toc-entry* template would have been `{title}`, not the expansion of `{title}`. If the *title* template is repeatedly redefined for each page, the final table of contents would expand to a sequence of `{title}` tags which would all expand to the last definition of the *title* template.

The `{define}` tag allows unlimited *nesting*. A template body can include `{define}` and `{append}` tags (as well as any other tags). The execution of these tags is deferred until expansion of the template.

#### **7.4 Tags: *{definelist}* and *{item}***

A common problem in web site creation is *lists of items*. As an example, the Prehistoric Archaeology of the Aegean web site has 29 lessons. Each lesson has a long and short title. This could be done using 29 `{define}` tags for each title size, but it was preferred that new pages be easy to add and that the system keep track of the number of pages automatically. The `{definelist}` and `{item}` tags provide this functionality. The `{definelist}` tag requires an end tag. The `{item}` tag is one of the few times in ASML when an end tag is optional. Items are automatically ended by the next item or by the `{/definelist}` end tag.

The attributes on the `{definelist}` tag are: `name`, `from`, `by`, and `defineto`. The `name` attribute determines the template name for each item in the list. This name will have a number automatically appended to it at template expansion time. This number starts at the value specified by the `from` attribute and is incremented by the value specified by the `by` attribute for each item, basically forming an *item counter*. If the `from` attribute is not supplied it

defaults to 1. If `by` is not specified, it defaults to 1. The `defineto` attribute names a special template which will be set to the number of items defined. The following is an example of the use of `{definelist}` and `{item}`:

```
{definelist name="lesson-title-" defineto="lesson-count" from="1" by="1"}
{item}The Early Cycladic Period
{item}The Early Minoan Period
{item}Minoan Architecture
{/definelist}
```

This tag will create four templates. Three will be named: `lesson-title-1`, `lesson-title-2`, and `lesson-title-3` and will contain the lesson titles in the list. The fourth will be named `lesson-count` and will be set to the final template append number, in this case 3. These features can be later used in conjunction with the `{foreach}` tag to generate multiple pages automatically and to iterate over a list of items.

### 7.5 Tag: `{else}`

See the `{if}` tag.

### 7.6 Tag: `{elseif}`

See the `{if}` tag.

### 7.7 Tag: `{foreach}`

The `{foreach}` tag implements a simple looping construct in ASML. This allows for creation of multiple pages based on some common template. As an example, the Prehistoric Archaeology of the Aegean site uses a `{foreach}` loop to generate the 29 lessons, filling in the titles, next and previous lesson information, and text for each [32]. The `{/foreach}` end tag is required. All content between `{foreach}` and `{/foreach}` is considered to be a loop body. Nested loops are fully supported in ASML, a loop body can contain another loop.

The `{foreach}` tag attributes are: `counter`, `from`, `to`, and `by`. The `from`, `to`, and `by` attributes define the *numeric stepping of the tag* as in the `{definelist}` tag. The `from` and `by` attributes default to 1. The `to` attribute indicates the stopping point for the loop and is required. The `counter` attribute specifies a counter in the form of a local template. While `{define}` tags defines templates which are persistent, (usable anywhere after the definition), the `counter` attribute specifies a *local template* which will only be valid in the loop body. It will expand to current loop count.

```
{foreach counter="lesson" to="{lesson-count}"}
  <p>{lesson-title-{lesson}}</p>
{/foreach}
```

This example illustrates several unique features of ASML. Note the expansion of a template (`{lesson-count}`) inside the `to` attribute value. This allows `{lesson-count}` to be defined by an earlier `{definelist}` tag. Also, note the expansion of a tag inside a tag name. This simple feature gives ASML an easy to use multidimensional array capability.

The `{/foreach}` end tag is required. The `counter` attribute can also be called `variable` or `var`. These alternative names have been retained for previous version compatibility and should not be used in new development. If the initial value of the counter (the `from` attribute value) exceeds the `to` attribute value, the body will not be executed. This determination is based on the sign of the `by` attribute. Negative values indicate the counter will be counting down. Positive values indicate the counter will be counting up. A `by` value of 0 is invalid. All counting in ASML is integer-based.

## 7.8 Tag: `{formget}`

`{formget}` is a unique tag in ASML. It accepts any attributes and values supplied and expands to a *form data string* using the GET method. This is identical to the data string generated by a form in an HTML page which utilizes the GET method. This provides a powerful mechanism for sending data using a URL.

The rules for generating *form data strings* in HTML require that all non-alphanumeric values be converted to hexadecimal and spaces converted to `+`. Options are delimited by `&` and `=` signs. The `{formget}` tag hides this mechanism and simply generates the string. Example:

```
<a href="http://www.yahoo.com/bin/search{formget p="HTML"}">Search for HTML</a>
```

This example will generate:

```
<a href="http://www.yahoo.com/bin/search?p=HTML">Search for HTML</a>
```

This is a simple example, but with hexadecimal conversions and many attributes the string can get quite complex. ASML hides this completely, allowing a form to be simply and easily simulated or allowing for transmission of arbitrary information to a CGI script. There is no `{formget}` end tag.

## 7.9 Tag: `{highlight}`

The `{highlight}` tag is used to highlight search results in a page. It is commonly used in conjunction with the `{search}` tag. It also provides a mechanism to jump to search results in a page. The `{highlight}` tag is discussed in detail in section 9.

## 7.10 Tags: `{if}` `{else}` `{elseif}`

The `{if}` set of tags in ASML provide for conditional components in a page. The form of the `{if}` construct in ASML is: `{if} {elseif} {elseif} {else} {/if}`. Nested if constructs are allowed. The `{if}` tag is based on a Boolean comparison of two parameters. These parameters are supplied as attribute values. There are several names for the attributes and they can be used interchangeably. The left half of the comparison option name is `left`, `this`, or `a`. The right side option name is `right`, `that`, or `b` (compare `this` to `that`, `left` to `right`, or `a` to `b`). The comparison option is specified by the `test` attribute. The default is equality. Test attribute values can be: `=`, `==`, `!=`, `<`, `>`, `<=`, `>=`, `eq`, `ne`, `lt`, `gt`, `le`, `ge`. The first set of options will test based on automatic content determination. If

content of both sides begins with “-” or a number, numeric comparison is assumed. The *letter-based options* (eq, etc.) force a string comparison. Comparisons are case insensitive.

ASML supports nested comparisons as well as multiple `{elseif}` tags within a single `{if}` tag.

### 7.11 Tag: `{img}`

The ASML `{img}` tag enhances the functionality of the HTML `<img>` tag. The `{img}` tag is only valid when generating an output page (within a `{page}` tag). There is one directly utilized attribute for the `{img}` tag, the `src` attribute. This attribute duplicates the standard HTML `src` attribute except that the file is considered to be relative to the *site* rather than relative to the *server*. All other attributes are passed through, though `height` and `width` are treated specially. As an example, the following tags:

```
{base system="/usr/tenaya/devwww/bronze" server="/bronze"}
  {img src="/images/xyz.gif" alt="img"}
```

will expand to the following HTML tag:

```

```

The `{img}` tag automatically tests for availability of the image and generates the `width` and `height` attributes. If the `width` and `height` attributes are supplied in the ASML `{img}` tag, they override the automatically generated versions. Automatic width/height generation can only occur for GIF and JPEG images contained in the site directory tree. URLs which specify a protocol or a different server are passed through without modification to the path or addition of a `width` and `height` attribute. The graphic image must be available in the site tree so that ASML can read it to determine height and width when processing the pages. ASML verifies the file available as part of this process.

### 7.12 Tag: `{import}`

The `{import}` tag is used to import foreign data formats with HTML translation. At this time only Rich Text Format (RTF) is supported. An import operation loads a foreign file format, but does not expand it into the current document. The `{section}` tag described below performs that function. The ASML import system is designed to support complex input operations such as importing a single, large file and splitting it among many output pages.

The basic attributes for the import tag are: `src`, `name`, and `type`. Specific import types may have additional options. Every import operation is assigned a name as specified by the `name` attribute value. This name will be used to refer to the imported content at a later point using the `{section}` tag. The `src` attribute value indicates the filename which is the import source data. This path is not modified by `{base}` tag contents since import data is unlikely to be in the site directory tree as it is not being served to the web in that raw form. The `type` attribute indicates the type of the file. At this time only `rtf` is supported. If the type option is omitted, ASML will attempt to determine the file type from the `src` value suffix.

An import operation in ASML automatically converts content to HTML body content. No `<head>` or `<body>` tags are included. The `{section}` tag is used to place contents from an import operation at specific points in pages.

### **7.13 Tag: `{include}`**

The `{include}` tag is used to include source from another file. It can be used to modularize a complex ASML document. It can also be used to include other HTML files into a site. The only supported attribute is `src` which indicates the source file. Filenames supplied to the `{include}` tag are not modified since they are not assumed to be in the site directory tree. An example of the `{include}` tag is:

```
{include src="bronzelessons.asml" }
```

### **7.14 Tag: `{index}`**

The `{index}` tag is discussed in detail in section 9.

### **7.15 Tag: `{item}`**

See section 7.4.

### **7.16 Tag: `{page}`**

The `{page}` tag starts generation of a page. It is used to generate HTML pages as well as map files, text files, CGI scripts, or most any other text output format. The available attributes are `file`, `index`, and `cgi`. The `file` attribute value indicates the output file name. This file will be in the site directory tree and will be modified to a full path and directory name. If the `file` attribute is omitted, the page will be output to the standard output device. This feature is typically used when ASML is utilized as a CGI scripting language.

The `index` attribute indicates if the page is to be indexed by the ASML indexing and searching system. If omitted, the page will not be indexing. Indexing must be explicitly specified. If a blank value or `all` is supplied, the page will be included in the “all” index. If another name is specified, an index with that name will be created if necessary and the page added to that index. This allows for multiple indices for a site. Every indexed page is automatically included in the `all` index as well as any other specified index.

The `cgi` attribute has no value and is used to indicate that the output page is to be a CGI script. The only real difference for a CGI script is the required execute file permissions. Generation of a CGI script is distinct from CGI runtime page generation (the `{page}` tag with no file option). The latter is described in more detail in section 10.

Pages are automatically set to all users read permission on creation in file systems supporting security. CGI scripts are automatically set to all users read and execute permission.

### 7.17 Tag: `{search}`

The `{search}` tag is discussed in detail in section 9.

### 7.18 Tag: `{section}`

The `{section}` tag expands to a *section* of an imported data file. This is a sophisticated tag with many options which allows content to be extracted from an imported file. It also allows for generation of a table of contents of that extracted content.

The primary attribute for the `{section}` tag is `src`. The value of this attribute specifies a name of content imported using an `{import}` tag. This allows multiple import text to be combined in single pages. This attribute must be supplied. The `type` attribute specifies the type of extraction. The value can be blank (equivalent to omitting the option) or `toc`. If `toc` is specified, a *table of contents* for the section will be generated rather than section text. The table of contents is HTML nested lists and includes links to the headings in the text. The table of contents entries are based on heading levels in the text. Often an `H1` tag is used in a page to indicate the title of the page and that title should be omitted from the table of contents. The `omithead` attribute can take a value which is a heading level. All levels less than or equal to this value will be omitted from the table of contents. If not specified, this tag defaults to blank, which is no omission. Any trailing punctuation is stripped in table of contents extraction.

There are several options for specifying the range of a section. The `begin` and `beginafter` attributes specify the beginning of a section. If omitted or set to blank the section is considered to begin at the beginning of the document. The value specifies a regular expression. The `begin` attribute indicates that the section will begin when the regular expression is matched. The `beginafter` attributes specifies that the section will begin after the match. ASML is careful to always include proper HTML elements, so these expressions need not include HTML tags. As an example, `beginafter="Bibliography"` will begin after "Bibliography" in a document, but will not include any closing paragraph or heading tags. Likewise, `begin="Bibliography"` will include any opening tags preceding and enclosing the word "Bibliography."

The end of section can be specified using the `end` or `endbefore` attributes. These options specify regular expressions which will indicate when a section ends. The default behavior or result of a blank string is to match the end of the file, an exception to the regular expression format.

## 8. Template Expansion

Each tag in ASML is tested against the *system tags* listed in the previous section. If the tag is not an ASML system tag, it is tested to see if it is a template. Templates include those templates defined by the `{define}`, `{definelist}`, and `{append}` tags as well as special local templates as created by the `{foreach}` tag. A tag representing a template is expanded to the template contents.

Template expansion in ASML will continue until there are no ASML tags remaining. This means that templates can contain tags which will, themselves, be expanded. ASML uses deferred expansion of all templates, so templates

with tags are defined as including the tags, **not the expansion of the tags**. The expansion takes place where the template is used. Note that templates in ASML can be used in pages, other templates, and even tag names and attribute values. (Templates in attribute names are not supported, though). Deferred expansion can be overridden in the `{define}` and `{append}` tags using the `expand` attribute.

There are three available attributes on a template expansion. The `add` attribute value contains a number that will be added to the template. Specifically, the template expansion will be treated as a number and the value of the `add` option added to it. This is commonly used to compute next and previous lesson numbers or any other numeric counters in a page.

The `index` and `split` attributes provide a mechanism for splitting a template into pieces. This is particularly useful for specifying several items in a single template, especially in large lists. The value of the `split` attribute defines a regular expression which can be used to split a template into pieces. The desired piece is determined by the value of the `index` attribute. The first of these pieces is numbered one (all numbering starts at one in ASML). The default value for `split` is “&”, the standard ASML split value. An example of template expansion using `split` is:

```
{define name="var"}A&B&C{/define}
    {var}
{var index=1} {var index=2} {var index=3} {var index=4}
```

This will expand to:

```
A&B&C
A B C
```

The `index=4` example expands to the empty string since there is not a fourth piece. Also, use of the template alone expands to the full template regardless of index character use (& is not reserved). Splitting can also be used to break user input in forms into components as in separating a comma delimited list.

Any attributes supplied to a template expansion other than `add`, `index`, and `split` are automatically defined as local templates. A local template is a template that exists only as long as the template tag is being expanded. This provides a simple parameterization method in ASML. A template can be created which has one or more fill in components. These components can be indicated using attributes on the template tag. An example of template parameterization is:

```
{define name="item"}<tr><td>{first}</td><td>{second}</td></tr>{/define}
```

In this example a template named `item` has been created which might provide a table row with two data cells. The template can be utilized as in this example:

```
{item first="A" second="B"}
```

This example will expand to:

```
<tr><td>A</td><td>B</td></tr>
```

Local templates can override global templates. Local templates are always tested prior to global templates. This can be used to provide a *default* for a local template simply by creating the default value as a global template. If the local template is not defined as an attribute on the template tag and is not available as a global template an error will be generated.

## 9. Searching and Indexing in ASML

A major feature of ASML is searching and indexing. ASML can generate index files in a B-Tree derived format for fast and efficient searching. It supports creation of search index files, multiple files per site for different search groups, ASML based CGI search scripts, and highlighting and jumping to search terms.

### 9.1 Tag: *{index}*

Indexing in ASML is almost totally automatic. The `index` attribute on the `{page}` tag specifies the names of the index and every indexed page is automatically added to an index called `all`. Once indices are collected, the `{index}` tag is used to write them to a file. The attributes for the `{index}` tag are `src`, which specifies the name of the index and `file`, which specifies the file name under which the index will be stored. The file name is relative to the site directory tree.

In ASML the default extension for an index file is `.indx`. Index files are binary files and are not directly viewable or editable. They use a common format for all machine architectures, though, so they need not be built uniquely for each platform.

### 9.2 Enabling and disabling indexing

Indexing in ASML is performed on the output content when the `index` attribute is added to the `{page}` tag. Often it is desired that some components of pages not be searched. In particular, common top and bottom elements of pages may weigh a search with redundant word usage. When ASML is indexing content, it searches for the special terms: `asmlindex` and `asmlnoindex`. Usually these will be enclosed in HTML comments as in this example:

```
<!-- asmlnoindex -->
This content will not be indexed by ASML
<!-- asmlindex -->
This content will be indexed by ASML
```

If these keywords are not found in a document the default is to index the entire document.

### 9.3 Tag: *{search}*

The `{search}` tag in ASML initiates and manages searches of indices. In ASML a search is performed, then the results of the search become available. It is possible to perform multiple searches with each search assigned a name.

The `name` attribute is required on every search. There are three forms of the `{search}` tag: the `dosearch` form, the count form, and the access form.

The `dosearch` form is specified through the use of the `src` and `dosearch` attributes. The `src` attribute indicates the file to be used. This file location is relative to the site directory tree due to the fact this is usually used at runtime. The `dosearch` attribute specifies the search string. This is a space delimited list of words to search for. There is an additional numeric attribute named `max`. If specified, this option limits the number of matches allowed by the search.

The count form of the `{search}` tag only uses the `name` attribute. It must be used after a search has been performed. It will expand to the number of hits in the search.

The access form of the `{search}` tag allows access to search results. The attributes for this form (other than `name` which is always required) are `search` and `item`. The `search` value specifies the index of the search result to be accessed, starting at one. The `item` value specifies what component of the search result the tag will expand to. The options are: `server`, `file`, `title`, `h1`, and `weight`. The `server` value causes the `{search}` tag to expand to a path name on the server for the file. This is an absolute path on the server, not a path relative to the base tag. The `file` value causes the `{search}` tag to expand to the file name on the local file system for the search result. This will always be relative to the server root for security reasons discussed section 10.1. The `title` value causes the `{search}` tag to expand to the page title for the search result. This is the contents of the `<title>` tag for the page. The `h1` value causes the `{search}` tag to expand to the first `<h1>` tag contents for the search result page. The `weight` attribute value causes the `{search}` tag to expand to the weight of the search result. This is a fixed point number displayed with a period between 0.000 and 1.000 which indicates the weight of the search result match. It is supplied for display in search pages.

#### **9.4 Tag: *{highlight}***

The `{highlight}` tag implements word highlighting. It is generally used to highlight terms which are the result of searches. The attributes for the highlight tag are `src`, `dosearch`, `color`, and `base`. The `src` value indicates the file name of the file which will be loaded and highlighted. This tag is expanded to the contents of the file indicated by the `src` option with the search terms highlighted. The `dosearch` value is a search string which indicates which words will be highlighted. Normally this will be set to the same value as the `dosearch` attribute in the original search. The `color` value defines the color for the text highlighting.

The `base` attribute is used to generate an HTML `<base>` tag in the highlighted page. Most pages will contain relative URLs. Indeed, ASML will automatically generate some relative URLs if possible, especially on `{img}` tags. Relative URLs are considered by a browser to be relative to the location of the containing page. However a highlight script may be in an entirely different location on the server, hence the browser cannot be aware of the proper addressing of relative URLs. The HTML `<base>` tag simulates the location of a page as different from the actual location as a support for this relocating characteristic. The `base` attribute on the `{highlight}` tag should

contain an **absolute URL** for the source page for the highlight (including the protocol and server name). The requirement that the URL be absolute is a requirement of HTML. See section 10 for examples using the `{highlight}` tag and the `base` attribute.

Highlighting of page content is done through the addition of `<font color="color" >` tags. The color default is `#ff0000` and can be changed using the `color` attribute. An anchor is created in the page immediately before the first occurrence of a search term with the name `highlight` (`<a name="highlight"></a>`). This can be used to jump the browser directly to the first search term occurrence.

## 9.5 Search Methodology

Searching in ASML is based on a system of weighted terms. The search terms are compared to the documents and a weight is accumulated. The resulting pages are sorted by decreasing match weight. The weight of a word in a document is dependent upon its usage and its length. Longer words have slightly higher weights than shorter words. Words which are emphasized or in headings have a higher weight than other words.

## 10. ASML as a CGI Language

When World Wide Web servers were first developed it was recognized that server designs could not anticipate all possible content that would be served. In particular, some sites serve dynamic content, content which does not exist as static pages stored in the file system. In addition, response to a query may be dependent upon the content of forms as transmitted by the browser.

The traditional approach to CGI is to write scripts in languages such as Perl or to use compiled programs written in languages such as C and C++. ASML provides an alternative in that ASML can be used to generate CGI functionality for servers. An ASML page can be used as a script in a Unix system, just as most other scripts, using the `#!` notation on the first line of the file to indicate the executable program which will process the script, in this case `asml`. As an example, a script may begin with:

```
#!/usr/local/bin/asml
```

The remainder of the file will be ASML. ASML will ignore the above text. (The text is not technically a comment in ASML, but is rather ignored because it is not in any tag, so it does not generate any output.)

CGI scripts are expected to generate data as a response to the server. By default, ASML will generate HTML. The output page must be contained in `{page}` tags. For CGI scripting, the output must be to “stdout”, the Unix standard output device. ASML will default to standard output if no `file` attribute is supplied on the `{page}` tag. The following is an example ASML CGI script:

```
#!/usr/local/bin/asml
{page}
<html>
<head><title>CGI Script Response</title></head>
```

```

<body><p>CGI Script Response</p></body>
</html>
{/page}

```

In this example the ASML script serves only to respond with a page. There is no conditional content. Conditional content in WWW pages is usually either dependent upon on forms variable content or loaded from dynamic files. ASML can load content, so dynamic content in that form is supported. ASML also processes forms variables. Form variables are automatically converted to templates by ASML. As an example, if a form has a text control named `email`, an ASML template named `{email}` will automatically be created when ASML is invoked as a CGI script. The contents of the template will be the contents transmitted from the text control. Both GET and POST form methods are supported by ASML.

The capability can be used for more than just forms. The `{formget}` tag in ASML can be used to transmit data from a page to an ASML CGI script using the same mechanism as that used by the GET method for forms controls. This technique can be easily used to generate dynamic content pages.

In addition to the form variables, a few additional templates are automatically created when ASML is invoked as a CGI script. `{SERVER_NAME}` is set to the full name of the server. `{SERVER_PORT}` will be set to the server port. These templates are created from the environment variables sent to ASML by the server and provide important information for scripts which may have to create links or a `<base>` tag.

A simple example CGI script written in ASML is a search highlight page. A highlight script example is:

```

#!/usr/local/bin/asml
{base system=".." server="/bronze"}
{page}
{highlight src="{searchfile}" dosearch="{searchtext}"
 base="http://{SERVER_NAME}{searchserver}"}
{/page}

```

This example illustrates many options used in ASML. Normally the search page will use the `{formget}` tag to send the `searchfile`, `searchtext`, and `searchserver` variables. These are all elements of the search index. Such a link might be created using this form:

```

<a href="/bronze/cgi-bin/highlight.cgi{formget
 searchserver="{search name="all" search="10" item="server"}"
 searchfile="{search name="all" search="10" item="file"}"
 searchtext="Nemea, Greece"}#highlight}>
 {search name="all" search="10" item="title"}</a>

```

This example may seem complex at first. The link is passing the 10th result of the search for “Nemea, Greece” to the highlight script. The `{formget}` tag is used to construct a GET format URL which contains the forms variables `searchserver`, `searchfile`, and `searchtext`. The `searchserver` variable contains the server name for the search result. Likewise, the system filename for the result is contained in `searchfile`. The `searchtext` item contains the

search text used so the `{highlight}` tag will know what text to highlight. The underlined link text in this example will be the title of the search result page. The `#highlight` anchor on the end of the URL will force a jump to the first highlighted term.

### ***10.1 Security Issues of ASML as a CGI Language***

Security is always an issue in server and CGI design. The World Wide Web can be accessed globally and attacks on web sites must be anticipated. The major security concern for site designers is CGI scripts. Content served as pages is finite and predictable, so the server can protect from subterfuge. However, CGI scripts must protect against any attacks from an infinite number of form data possibilities. Form data is easily forged. As an example, passing a filename may be required for dynamic content. This structure is commonly used with the `{highlight}` tag as described above. The major concern is that the relative addressing using “..” may be used to access files not normally available to the server.

For this reason, the `{highlight}` tag will not read content outside the server search tree. Specifically, the “..” operator is ignored. This is a first approximation at ASML automatic security. Future ASML development will include a security structure for CGI scripts which will default to a secure environment, thereby alleviating user concerns in this area. This step must be taken carefully since CGI scripts may explicitly need to access data outside the server directory tree in an alternate, protected area. In particular, unformatted data may be protected from server access and only be valid as an ASML access. The planned support is default protection with a new security override option on all tags which access files.

### ***10.2 Future CGI issues***

ASML CGI support for dynamic content is currently quite powerful. However, there is no current support for stored data as in forms responses, guest books, etc. The planned support for these requirements includes new functionality on the `{page}` tag for appending to files. In addition, the possibility of a simple database connection is under consideration, allowing SQL or other database query language statements for database manipulation and query. Results will be in the form of ASML templates which can then be expanded into generated pages.

## **11. RTF Import Type**

The `{import}` tag basic attributes are extended for the Rich Text Format (RTF) type with the following new attributes: `h1`, `h2`, `h3`, `h4`, `h5`, `h6`. RTF does not include a mechanism for specifying headings. Even if it did, there is no guarantee that document producers will properly utilize this information. Hence, the `h1-h6` attributes in ASML allow the specification of rules by which a heading can be deduced from the document text. The value of these options is a comma or space delimited list of keywords. The keywords available are: `first`, `allcaps`, `mixedcase`, `strong`, `em`, `never`. The default values are: `h1="first, strong"`, `h2="strong, em, allcaps"`, `h3="strong, em, mixedcase"`, `h4-h6 default to "never"`. The paragraphs of the import text are scanned for these properties and the list of properties is considered a combination combined using Boolean and (all properties must be true).

The `first` property is set for the first paragraph of the document. It is not uncommon that the first element of the document is the title. Hence, it is the default for `h1`. The `strong` property is set if the entire contents of the paragraph is enclosed in `<strong>` tags. The `<strong>` tag is the translation of the RTF bold property. Spaces are not scanned for this property. The `em` property is set if the entire contents of the paragraph is enclosed in `<em>` tags. The `<em>` tag is the translation of the RTF italics and underline properties. Spaces are not scanned for this property. The `allcaps` property is set if all characters in the paragraph are upper case. The `mixedcase` property is set if lower case letters exist in the paragraph. The `never` property is not set for any paragraph. Hence, it effectively disables this headings type.

Headings are always determined in RTF in a hierarchical fashion and only one H1 tag is allowed, in keeping with the HTML specification. Headings following previous headings will always be either less than the previous headings, equal, or increasing in level by at most one.

## 12. Invoking ASML

ASML is currently only available for Unix platforms. It is invoked as a command line program with options. The normal option is a filename which will be processed. The file extension is required:

```
asml bronze.asml
```

The `.asml` extension is not enforced by ASML, but is recommended. The `-v` switch can be used to determine the ASML version:

```
% asml -v
ASML: Automatic Site Markup Language v1.03
(c) 1996, Trustees of Dartmouth College
```

## 13. Future Work

ASML is currently in version 1.03. Many new features are anticipated for further development. Most HTML tags, especially the `<a>` tag, will be duplicated in ASML, giving the system more power to direct the page generation process. The search engine is very basic at this time and new, alternative, structures are anticipated to provide the site-designer with more flexibility in content searching. The current structure of ASML builds new sites when `asml` is invoked. While ASML is very fast, it may be inefficient to generate a large site in this way. A system of automatic dependency checking is planned which will allow only pages which changed content to be rebuilt. The previously mentioned CGI security enhancements are under development. These features are predominantly designed to protect the unskilled user from security mistakes. Finally, an extended quotation system for cascading style sheets and JavaScript is anticipated.

## 14. Acknowledgments

Development of ASML was supported through a grant from the Dartmouth College Venture fund. The Prehistoric Archaeology of the Aegean web site was supported through a grant from the Foundation for the Hellenic World.

This site was based on material supplied by Jeremy Rutter. Pauline Christo served as a development team leader on the project. Previous DEVLAB web projects have been supported by the Foundation for the Hellenic World, the Women in Science Program, Addison-Wesley, and the National Science Foundation. Samuel A. Rebelsky offered advice on eliminating “programmerese” from the system. Many other people have contributed to the development of ASML through input into the development process, use of the system in site development, or participation in previous non-ASML projects.

## **15. Conclusion**

ASML is a new approach to site-level World Wide Web development. An important feature of ASML is the view of a web site as a complete document, not a collection of disjoint pages. It retains the simple structure of HTML and total compatibility with existing browsers and servers while providing new high-level capabilities such as centralization of common page elements, powerful search and indexing features, data import, and more. The syntax is similar to HTML and, therefore, easy to learn and use. Though the environment has programming features it is not fundamentally a programming environment and no programming skills are required to use ASML. In many cases ASML can replace programming solutions in site development.

## Bibliography

1. *The Ancient Olympic Games Virtual Museum*, <<http://devlab.dartmouth.edu/olympic/>>.
2. C. Bacher and T. Ottmann. Tools and services for authoring on the fly. In *Proceedings of ED-MEDIA'96*, pages 7-12, Boston, MA, 1996. Association for the Advancement of Computing in Education.
3. M. Bichler and S. Nusser. Developing structured WWW-sites with W3DT. In *Proceedings of WebNet'96 World Conference of the Web Society*, pages 7-12, San Francisco, CA, 1996. Association for the Advancement of Computing in Education.
4. E. S. Bos, A. Kikstra, and C. M. Morgan. Multiple levels of use of the web as a learning tool. In *Proceedings of ED-TELECOM'96 World Conference on Educational Telecommunications*, pages 31-36, Boston, MA, 1996. Association for the Advancement of Computing in Education.
5. P. Brusilovsky, E. Schwartz, and G. Weber. A tool for developing adaptive electronic textbooks on WWW. In *Proceedings of WebNet'96 World Conference of the Web Society*, pages 64-69, San Francisco, CA, 1996. Association for the Advancement of Computing in Education.
6. C. A. Carver and C. Ray. Automating hypermedia course creation and maintenance. In *Proceedings of WebNet'96 World Conference of the Web Society*, pages 82-87, San Francisco, CA, 1996. Association for the Advancement of Computing in Education.
7. *The Dartmouth Experimental Visualization Laboratory*, <<http://devlab.dartmouth.edu/>>.
8. *The Dartmouth Institute for Advanced Graduate Studies (DAGS)*, <<http://www.cs.dartmouth.edu/dags/>>.
9. C. Davis, *Carl Davis's HTML Editor Reviews*, <<http://homepage.interaccess.com/~cdavis/editrev/>>.
10. N. Drakos, From text to hypertext: A post-hoc rationalization of LaTeX2HTML, in *Proceedings of the First World Wide Web Conference*, Geneva, Switzerland, May 1994,
11. *Frontier Community Center*, <<http://www.scripting.com/frontier/>>.
12. *Haht Software*, <<http://www.haht.com/>>.
13. C. Hall and C. Tews, UserLand's scripting system fulfills its manifest destiny, *MacWeek*, 10(28), July 22, 1996.
14. *Information Processing Systems – Text and Office Systems – Standard Generalize Markup Language (SGML)*, ISO IS 8879.
15. W. L. Johnson, T. Blake, and E. Shaw. Automated management and delivery of distance courseware. In *Proceedings of WebNet'96 World Conference of the Web Society*, pages 225-230, San Francisco, CA, 1996. Association for the Advancement of Computing in Education.
16. L. Lamport, *Latex, A Document Preparation System*, 1986, Addison-Wesley, Reading, MA.
17. J. Lennon and H. Maurer. Aspects of large World Wide Web systems. In *Proceedings of WebNet'96 World Conference of the Web Society*, pages 298-303, San Francisco, CA, 1996. Association for the Advancement of Computing in Education.
18. F. Makedon, J. Ford, M. Kenyon, and C. Owen. Ancient museum collections and the web. In *Proceedings of WebNet'96 World Conference of the Web Society*, pages 315-329, San Francisco, CA, 1996. Association for the Advancement of Computing in Education.

19. Macromedia Corporation, <<http://www.macromedia.com/>>.
20. Microsoft FrontPage, <<http://www.microsoft.com/frontpage/>>.
21. H. Maurer. *Hyper-G, now HyperWave*, 1996, Addison-Wesley Longman, Harlow, England.
22. Netscape Corporation, *JavaScript Authoring Guide*,  
<<http://home.netscape.com/eng/mozilla/2.0/handbook/javascript/>>.
23. J. Nielson, *Why Frames Suck (Most of the Time)*, December, 1996,  
<<http://www.useit.com/alertbox/9612.html>>.
24. C. B. Owen, *Automatic Site Markup Language*, <<http://devlab.dartmouth.edu/asml/>>.
25. C. B. Owen, *The ImageTcl Multimedia Development System*, <<http://devlab.dartmouth.edu/imagetcl/>>.
26. C. B. Owen and F. Makedon, Multimedia data analysis using ImageTcl. In *Gesellschaft für Klassifikation e.V.*, University of Potsdam, Potsdam, Germany, March 12-14, 1997, to appear.
27. S. A. Rebelsky, CourseWeaver: A tool for building course-based webs, In *Proceedings of ED-MEDIA '97*, Calgary, Canada, June 14-19, 1997, to appear.
28. J. C. Rice, P. F. Merrill, and C. L. Hawkins. Procedures for creating useful web sites. In *Proceedings of WebNet'96 World Conference of the Web Society*, pages 413-418, San Francisco, CA, 1996. Association for the Advancement of Computing in Education.
29. J. B. Rutter, et. al., *The Prehistoric Archaeology of the Aegean*, 1996,  
<[http://devlab.dartmouth.edu/history/bronze\\_age/](http://devlab.dartmouth.edu/history/bronze_age/)>.
30. J. B. Rutter, et. al., *The Prehistoric Archaeology of the Aegean*, 1995,  
<<http://www.indiana.edu/~classics/aegean/Rutter.html>>.
31. H. W. Lie and B. Bos, *Cascading Style Sheets, level 1*, W3C Recommendation REC-CSS1-961217, December 17, 1996, <<http://www.w3.org/pub/WWW/TR/REC-CSS1>>.
32. L. Wall, and R. L. Schwartz, *Programming Perl*, 1991, O'Reilly and Associates, Inc., Sebastopol, CA.