XSL Editor Prototype

by

Lindsay D. Grace

A proposal for a final project to be submitted

in partial fulfillment of the requirements

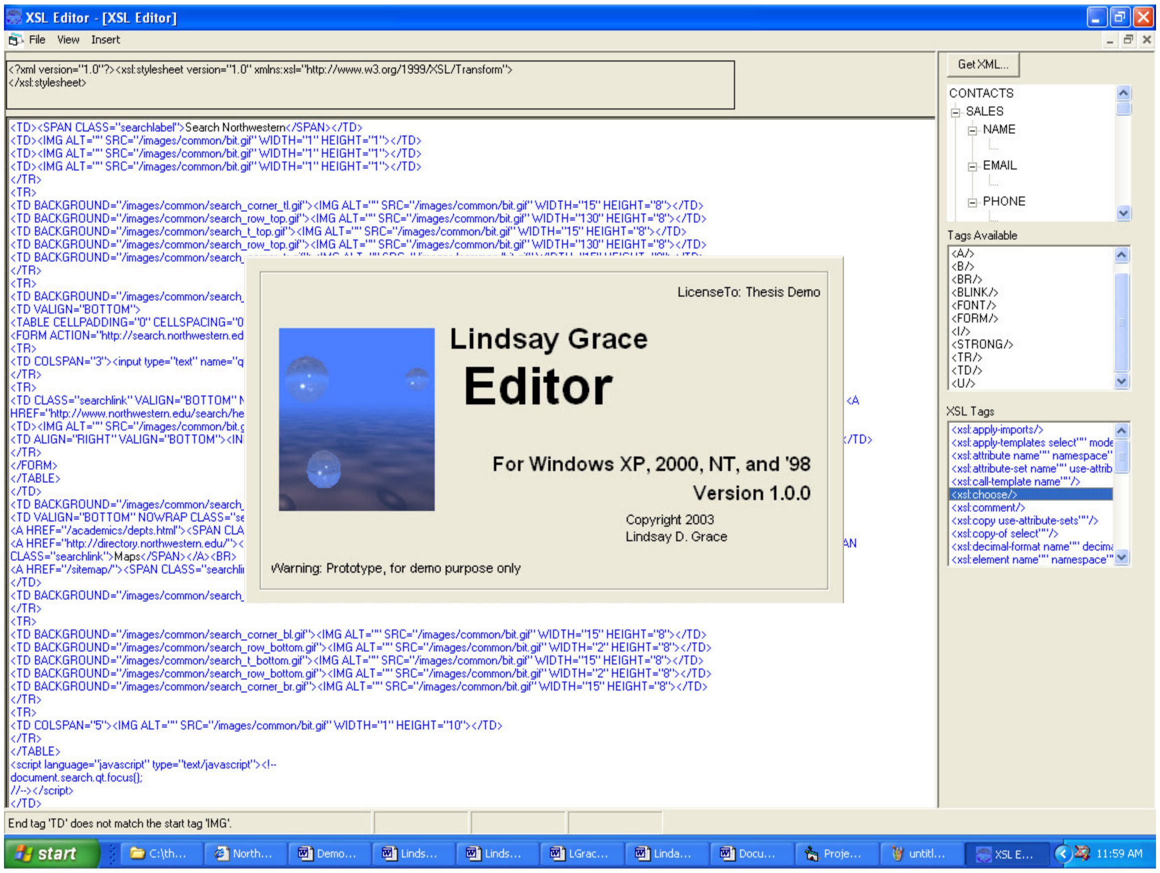
for the degree of

Masters Degree in Computer Information Systems (MSCIS)

Northwestern University

2003

Advisor: Dr. Chang Miao
Submitted: August 13, 2003



```
<?xml version="1.0"?><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

```
<TD><SPAN CLASS="searchlabel">Search Northwestern</SPAN></TD>
<TD><IMG ALT="" SRC="/images/common/bit.gif" WIDTH="1" HEIGHT="1"></TD>
<TD><IMG ALT="" SRC="/images/common/bit.gif" WIDTH="1" HEIGHT="1"></TD>
<TD><IMG ALT="" SRC="/images/common/bit.gif" WIDTH="1" HEIGHT="1"></TD>
</TR>
<TR>
<TD BACKGROUND="/images/common/search_corner_tl.gif"><IMG ALT="" SRC="/images/common/bit.gif" WIDTH="15" HEIGHT="8"></TD>
<TD BACKGROUND="/images/common/search_row_top.gif"><IMG ALT="" SRC="/images/common/bit.gif" WIDTH="130" HEIGHT="8"></TD>
<TD BACKGROUND="/images/common/search_t_top.gif"><IMG ALT="" SRC="/images/common/bit.gif" WIDTH="15" HEIGHT="8"></TD>
<TD BACKGROUND="/images/common/search_row_top.gif"><IMG ALT="" SRC="/images/common/bit.gif" WIDTH="130" HEIGHT="8"></TD>
<TD BACKGROUND="/images/common/search_
</TR>
<TR>
<TD BACKGROUND="/images/common/search_
<TD VALIGN="BOTTOM">
<TABLE CELLPADDING="0" CELLSPACING="0
<FORM ACTION="http://search.northwestern.ed
<TR>
<TD COLSPAN="3"><input type="text" name="q
</TR>
<TR>
<TD CLASS="searchlink" VALIGN="BOTTOM" N                                                            <A
HREF="http://www.northwestern.edu/search/he
<TD><IMG ALT="" SRC="/images/common/bit.g
<TD ALIGN="RIGHT" VALIGN="BOTTOM"><IN                                                               </TD>
</TR>
</FORM>
</TABLE>
</TD>
<TD BACKGROUND="/images/common/search_
<TD VALIGN="BOTTOM" NOWRAP CLASS="se
<A HREF="/academics/depts.html"><SPAN CLA
<A HREF="http://directory.northwestern.edu/"><                                                      AN
CLASS="searchlink">Maps</SPAN></A><BR>
<A HREF="/sitemap/"><SPAN CLASS="searchli
</TD>
<TD BACKGROUND="/images/common/search_
</TR>
<TR>
<TD BACKGROUND="/images/common/search_corner_bl.gif"><IMG ALT="" SRC="/images/common/bit.gif" WIDTH="15" HEIGHT="8"></TD>
<TD BACKGROUND="/images/common/search_row_bottom.gif"><IMG ALT="" SRC="/images/common/bit.gif" WIDTH="2" HEIGHT="8"></TD>
<TD BACKGROUND="/images/common/search_t_bottom.gif"><IMG ALT="" SRC="/images/common/bit.gif" WIDTH="15" HEIGHT="8"></TD>
<TD BACKGROUND="/images/common/search_row_bottom.gif"><IMG ALT="" SRC="/images/common/bit.gif" WIDTH="2" HEIGHT="8"></TD>
<TD BACKGROUND="/images/common/search_corner_br.gif"><IMG ALT="" SRC="/images/common/bit.gif" WIDTH="15" HEIGHT="8"></TD>
</TR>
<TR>
<TD COLSPAN="5"><IMG ALT="" SRC="/images/common/bit.gif" WIDTH="1" HEIGHT="10"></TD>
</TR>
</TABLE>
<script language="javascript" type="text/javascript"><!--
document.search.qt.focus();
//--></script>
</TD>
```

LicenseTo: Thesis Demo

Lindsay Grace
Editor

For Windows XP, 2000, NT, and '98
Version 1.0.0

Copyright 2003
Lindsay D. Grace

Warning: Prototype, for demo purpose only

```
CONTACTS
  SALES
    NAME
    EMAIL
    PHONE
```

```
<A/>
<B/>
<BR/>
<BLINK/>
<FONT/>
<FORM/>
<I/>
<STRONG/>
<TR/>
<TD/>
<U/>
```

```
<xsl:apply-imports/>
<xsl:apply-templates select"" mode
<xsl:attribute name"" namespace"
<xsl:attribute-set name"" use-attrib
<xsl:call-template name""/>
<xsl:choose/>
<xsl:comment/>
<xsl:copy use-attribute-sets""/>
<xsl:copy-of select""/>
<xsl:decimal-format name"" decima
<xsl:element name"" namespace""
```

End tag 'TD' does not match the start tag 'IMG'.

Table of Contents

List of Figures

Background: Problem Statement and Goals

I believe that there is a general need and demand for an application that shields developers from the intricacies of XSL development. In certain software development projects, there is a need for simple, experiential development of XSL. Often, development teams need a fast and imperfect solution to demonstrate plausibility, or to understand the scope of their endeavors. Typically described as the colloquial *quick and dirty solution*, it is a sometimes-common request in fledgling projects and prototypes.

Likewise, software development at small scales, such as personal web sites and low traffic intranets, require simple, easily maintained and developed software. Intricate integrated development environments (or IDEs) are more complex than these types of solutions demands. Accordingly, in several technologies, simpler development environments have succeeded by offering less detailed authoring control. Such software capitalizes on simple visual interfaces and core capabilities.

In the world of static hypertext markup language (HTML), this is analogous to the comparison of Macromedia's Dreamweaver to Microsoft's FrontPage software. Dreamweaver represents a thorough incorporation of most HTML related technologies. Dreamweaver allows authors full control in authoring. Nearly every element of an HTML document can be specified through the application's array of menus and toolbars. At the other end of the development spectrum, is Microsoft's FrontPage. This tool is much simpler and offers far less control over documents. The simplicity comes at the cost of specificity. FrontPage makes many decisions for you, or simply does not provide a way of completing certain tasks. Since FrontPage costs less than half as much as Dreamweaver, FrontPage has a comfortable place in the industry. If your full-time job is not to make superb HTML web pages, FrontPage does what you need it to, and it does so, for less, and with less of a learning curve.

About seven years ago, the world of static HTML development seems to have been at the same point XSL development is today. At that time, a large number of the entry-level HTML editors were marketed not at the enterprise level developer, but at the individual home user and entry-level developer. Such applications included Sausage Software's Hot Dog Pro, Microsoft's Front Page Lite, and Netscape's Composer. These applications did not expose everything HTML was capable of, but they allowed people to build HTML quickly and easily, without a full understanding of the language.

XSL is nearing the same level of maturity and seems primed for a set of analogous products. Unfortunately, the leaders in the web page editing industry have not been quick to jump at the task of creating XSL editing software for entry-level developers.   This project is a step in that direction.

Overview of Technologies

Before fully discussing the benefits of this project, it is essential to discuss XSL related technologies. The following sections outline the fundamentals of these technologies.

XML Fundamentals

XML, or the extensible markup language, is a W3C recommended standard mark up language designed to facilitate the communication of information between a wide range of systems. XML has emerged as the standard means for communicating data in a non-system specific way. Its historical roots are in Standard General Mark-Up Language (SGML), which means it demonstrates some characteristics with the well-known HTML.

In essence, XML is a language consisting of tags and attributes. A tag is a unit of description. An attribute describes a tag. An example tag may be <thesis>. Attributes of the tag <thesis> may be *written by* and *completed date.* From both a syntactical and conceptual perspective, attributes are contained within the tag. The tag's content is anything that sits between the beginning tag and the end tag. In our example, the start tag may be described as <thesis> and the end tag as </thesis>. These tags describe data. Anything inside the tag is the tag's data. To that extent, HTML and XML are similar.

In implementation, it is the rules of XML that constitute XML. This includes not only the syntax but also the form. Since XML is only a set of rules, it does not prescribe an exhaustive list of core tags, as does HTML. There are a few XML reserved tags that provide standard information about an XML document, but these are not essential to the successful use of an XML data source. XML authors, for all intensive purposes, define their own tags based on a standard set forth by a standardizing body, an architect, or the developer. This full flexibility and extensibility allows for semantically rich markup.

XML source data is typically contained within a text file. It conveys relationships between its content by the relative placement of that content to other content within the document. All that can be known about an XML document is known through its structure and content. This is why XML has sometimes been coined a language of relationships.

If XML has been modeled and authored well, an untrained eye will understand it. Since it is a text-formatted document, it can also be parsed and understood by a wide range of computer platforms and languages. Because XML has this wide range of interoperability, it is excellent for the aggregation of data from multiple sources, and for conversely disseminating information to multiple external clients. As noted in the introduction to the W3C's XML recommendation," XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere"(W3C).

The core goals of XML as stated by the W3C recommendation are:

1.XML shall be straightforwardly usable over the Internet.
2.XML shall support a wide variety of applications.
3.XML shall be compatible with SGML [Standard Generalize Markup].
4.It shall be easy to write programs which process XML documents.
5.The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6.XML documents should be human-legible and reasonably clear.
7.The XML design should be prepared quickly.
8.The design of XML shall be formal and concise.
9.XML documents shall be easy to create.
10.Terseness in XML markup is of minimal importance. (W3C)

(Table 1 – The Core Goals of XML)

Namespaces are another important aspect of the XML recommendation. The namespace is simply a means of associating an XML vocabulary with a context. Consider the English word *project*. It can be used as a verb, as in, *to project one's voice*, or a noun, as in *my software project*. A namespace allows an author to indicate that the word is being used as a verb or as noun. In other words, it would indicate that this instance of *project* refers to the vocabulary of *valid English nouns* or *valid English verbs*. When using the word *project*, you might declare these namespaces in your document, and then specify to which of these namespaces (verb or noun) this *project* belongs. Beginning XML reminds us that namespaces are a "purely abstract entity; it's nothing more than a group of names that belong with each other conceptually" (Cagle, 68).

XML Syntax Fundamentals

XML is a language of relationships. Each element in the document has a relationship to each other element in the document, and each node within the document can be described as parent, child, or sibling to another element. Each tag should describe its content, and the nesting of tags should describe their relationships and the relationships of the data.

XML's ability to describe these relationships is supported by one core rule. This is the rule of well-formedness. Any functional XML document must be well-formed. This means that every beginning tag is closed in the order it was presented. If tags are nested, then, internal tags must be closed before the parent tag is closed.

Consider the following sample XML code:

```
<product>
        <description>Software</description>
        <productnumber>773412039</productnumber>
</product>
```

(Figure 1 – Sample of Well-Formed XML)

This document is symmetrically labeled. Since XML is case sensitive, the closing and end tags must have the same case. The following document segment is not well-formed:

```
<product>
          <description>Software</description>
          <productname >Microsoft Office
          <productnumber>773412039</productnumber>
</PRODUCT>
```

(Figure 2 – Sample of XML, not Well Formed)

This mark up lacks a closing tag for the <productname> element and the <product> end tag does not have the same case as the <product> start tag. The document segment does not adhere to the syntactical rules of XML, and would therefore not be considered usable XML.

Other fundamental constraints of XML well-formedness include:

- All attribute values must be enclosed in double quotation marks
- Attribute names must not be repeated in the same element.
- XML special characters can not be used as XML data (e.g. <. >, &)

When an XML document has met these criteria, it is considered "well-formed." If an XML document is not well-formed, it is not usable by XML parsers.

XML Validity

An XML document must also be valid. A valid XML document is one that adheres to the rules of relationships and content specified by the XML schema. A validating parser checks the validity of an XML document using the validation rules specified in a separate document.

XSL

XSL, or the extensible style sheet language, is the standard means of reformatting an XML document. Currently, XSL supports the transformation of XML data into a new

XML, HTML, or text document. It is important to note that XSL is a language. A well authored XSL typically contains logic, variables, and other artifacts of traditional programming languages.

XSL syntactical rules are very similar to those of XML. All XSL documents must be well-formed. XSL documents are typically validated against the XSL namespace. The XSL namespace contains all of the valid XSL specific elements and attributes. There are nearly one hundred XSL reserved keywords that define the syntactical language of XSL.

XSL is used on the client-side and the server side. Because parsing an XML document and applying a stylesheet is a memory intensive process, XSL transformations are most often executed on the server side. Server side transformation also means that client machines see no change in the type of data them must consume.

XSL employs several other W3C standards for the retrieval and formatting of XML documents. These are defined in the next few sections.

XPath

XPath is the query language used to extract data nodes from an XML document. XPath's basic syntax is much like DOS path, where each node in a tree is retrieved by adding a back-slash to the node's name (e.g. "C:\my documents\mydocument.doc"). XPath developed as a more efficient method of retrieving data from an XML document.

Although XPath contains many of the same syntactical requirements of XML and XSL, it does have a separate recommendation by the W3C.

XML Schema Fundamentals

The valid relationships of each element in an XML document are designed in and relayed by an XML schema. These XML schemas must be modeled to effectively represent the content provided within the XML document. A well-constructed XML schema should be intelligible and should illustrate its semantic content to casual readers.

It is important to recognize that the design of an XML schema greatly affects its long-term usability. Poorly designed schemas are difficult to work with and will likely not meet needs as they develop. Because XSL is used iteratively, a compact XML document will prove most useful. Designing an appropriate schema is one of the most important tasks in proper XML development. It is often difficult to decide to what level of specificity the schema should describe its content. In general, DOM schemas should be written with the most faithful description of its source data.

Consider the data evolution chain from source data to presentation of that data: Source Data → Component → XML → Transformation → Presentation. Presentation should, and will, change frequently whereas changes to the source data—assuming there has been good data modeling before development—are far less common. Presentation must adjust to changes in web browsers and the new means of communication that sprout as quickly as presentations are launched. The second reason for adhering to the content of the source data is that it is much easier to ignore data when transforming an XML document than it is to add data. XML documents that have a high degree of specificity allow a transforming template to ignore unneeded information, while those that are vague or do not contain necessary information force the transforming template to evaluate the XML data and create any needed information. Highly descriptive XML should relay the data it contains both in the order, and in the structure of its schema. A solid XML schema is the foundation of a successful XML based implementation. The quality of a schema is often reflected by the XSL document that consumes it. A poor schema will likely require an unwieldy XSL.

XML Parser

Tying it all together

The parser is an essential component to any XML and XSL based application. The XML parser, like many parsers, dissects the contents of a document into its various parts. This software component sits between the application and the XML files, shielding the developer from the intricacies of the XML syntax by reading and interpreting the XML. It also checks for syntactical errors. Although it is possible to use XML without a parser, most XML applications are based on some sort of XML parser.

XML parsers come in two types: non-validating and validating. Non-validating parsers do not check documents the document’s structure and content; they only check that the document is properly marked-up according to XML syntactical constraints. In addition to checking well-formedness, validating parsers verify that the document meets specific requirements of structure and content as outlined by the related schema This check is completed using a DTD or Schema that is either internal or external to the XML file being parsed.

DTD

The purpose of a Document Type Definition (DTD) is to define the legal building blocks and the legal structure of an XML document. DTDs can be declared inline in your XML document so that the document can carry its own description, or as an external reference. DTD was developed before XML and is not the ideal technology for validating documents. Therefore, much of the development of document description and validation standards focuses on XML Schema.

XML Schema

The word schema has two meanings in the general vocabulary of the XML development community. There is the conceptual structure of each document as outlined by a data modeler or architect. The preceding section, entitled XML Schema Basics, outlines the fundamental decisions required for creating an effective standard content model for XML documents (i.e. a Schema). There is also the W3C recommended methodology for validating a document, known as XML Schema.

XML Schema is an XML based alternative to DTD, providing a description of the structure and the legal building blocks of XML documents. XML Schema provides a means to define the elements, attributes, and child elements that can appear in a document, the number of each that can appear, what type of data each element and attribute can contain, and what their default values are. Microsoft originally proposed XML Schema, but now that it is a W3C proposal, it is poised to replace DTD.

Its advantages are apparent:

- It is easier to learn than DTD
- It is extensible to future additions
- It produces richer and more useful schemas than DTD
- It is written in XML,
- It supports data types
- It supports namespaces

For the remainder of this document, the term schema, refers to the conceptual schema as first mentioned. When it is necessary to describe the XML Schema as a validating document, it will be described as the *XML Schema formatted validation document*.

Document Object Model (DOM)

The XML Document Object Model (DOM) is a standard API (Application Programmer Interface) for XML documents that provides access to a wide variety of applications. With the XML DOM, a programmer can create an XML document, navigate its structure, and add, modify, or delete its elements. An XML parser can be used to load an XML document into the memory of your computer. Once the document is loaded, its information can be retrieved and manipulated by accessing the DOM. The DOM represents the XML document in a tree view.

As its name suggests, the DOM is a representation of an XML document as an object. Therefore XML documents can be manipulated by a set of properties and methods accessible through the object's interface. In essence, the DOM, is a standard API making it possible to create the same solution accross programming languages and platforms. While programmers may need to use different programming languages, they do not need to change their programming model.

For this project, I have chosen to use version four of Microsoft's XML parser. It provides both DOM and SAX support.

Simple API for XML (SAX)

The Simple API for XML, or SAX, is an alternative API to the previously discussed DOM. Unlike DOM, which creates a tree representation of the XML documents, SAX does not have a default object model. A SAX parser is only required to read in an XML document and execute events based on what it encounters. Instead of parsing and populating the entire tree of relationships between XML data nodes, SAX simply announces the relationship of one node to another. SAX parsing treats the XML data as a stream, and unravels that data based on the relationship of a current element to its direct ancestor. For this reason, a SAX parsing requires less memory, but it also often more complicated to code queries against.

While it is possible to recreate the entire set of relationships within an XML document through SAX, it is much easier to get that information in the DOM. SAX provides benefits in three distinct situations:

- When small pieces of data must be extracted sequentially from a large XML document or DOM
- Where the DOM's relatively large size is more costly than the act of reconstructing an XML document from a SAX parse
- Where memory resources are so constrained that the DOM is unacceptably large.

XHTML

Extensible hypertext markup language (XHTML), the W3C's recommended replacement of HTML, is similar to HTML 4.01. However, it is a simplified, version of HTML that combines the syntactical rules of XML with the basic presentation elements of HTML 4.01. The main goal of XHTML is to return to the original goals of HTML; standardized presentation of content. As such, XHTML removes much HTML that had to do with appearance and flair. Many of the attributes that defined color or effects are deprecated in XHTML. The W3C recommends using Cascading Style Sheets or similar technologies with XHTML to promote a more distinct separation of presentation tier information.

The reasons for this move are twofold. First, it is readily apparent that the HTML standard had been muddled by web browser vendor's creation and adoption of tags and attributes not generally decided upon by a central body (W3C, XHTML). Simply, HTML 4.01 is an amalgam of decisions made separately, and in some cases contradicting an existing consistency in style and validity. Tags such as "blink" (<blink>) were introduced by a specific vendor, and then inappropriately added to the standard, only to be deprecated later. Unfortunately, the vendors aim to maintain backwards compatibility with prior versions of their browsers. While the tag may officially be deprecated, it may continue to be understood by future browsers to maintain backwards compatibility between upgrades. Likewise, competing vendors want to make sure that their products seem to have the same level of functionality that other market participants have. In turn, the competitors begin to support tags that are not part of the standard. XHTML alleviates these problems by:

1. Simplifying the standard. XHTML reduces the amount of high-resolution presentation control, and substitutes it with simple structure that can be consistently interpreted by a variety of browsers.
2. Putting the standard in XML format, so that it may be validated using the dominant validation standard. If a developer uses an invalid tag, that document will no longer be considered valid XHTML and should generally be rejected by the rendering browser.

The second reason for the move to XHTML is more often the subject of debate. The argument works as follows.

Current browsers running on PCs are sophisticated and supported by enough processing power to resolve most HTML even if it does not conform to HTML 4.01's required syntax. On the other hand, XML is a strict markup language that requires everything be marked up correctly, resulting in "well-formed" documents. As new digital devices proliferate, many will not have the resources to interpret a "bad" markup language. XHTML pages can be read by all XML enabled devices and, while waiting for the world to upgrade to XML supported browsers, XHTML forces developers to write well-formed documents that will work in all browsers.

This has several added benefits:

- It continues the trend toward separating structural and semantic markup from presentation-based information
- In doing so, it condenses documents and makes their presentation more consistent between clients (because there is less room for interpretation).

As HTML is slowly replaced by HTML, it will become a very important technology.

Currently XHTML has not been widely adopted. For existing websites, it will require heavier use of Cascading Style Sheets and a substantial review of existing presentations. As companies begin to transition from classic ASP to .Net based web forms there may be a proportional growth of XHTML content on the web. In these code conversions, developers will be rewriting much of the presentation tier and it would be an ideal time to transition to a new presentation standard.

Ancillary Technologies

XML and the supplemental family of technologies are new and—as a centerpiece of Microsoft’s, IBM’s, Sun’s, and other’s web strategies—are the focus of intense development. Therefore, there is a growing proliferation of proprietary and non-proprietary XML related standards and technologies. This discussion serves to introduce the most important of these standards. Please review the bibliography for more comprehensive information and personal research.

Relevance and Significance of Research

The Solution Described:

This project endeavors to simplify the use of XSL related technologies by providing a simple, easy to use application that allows developers to author documents in the XSL and XPath languages. The XSL they develop using the tool can then output content as HTML, XHTML, or any language the user requires.

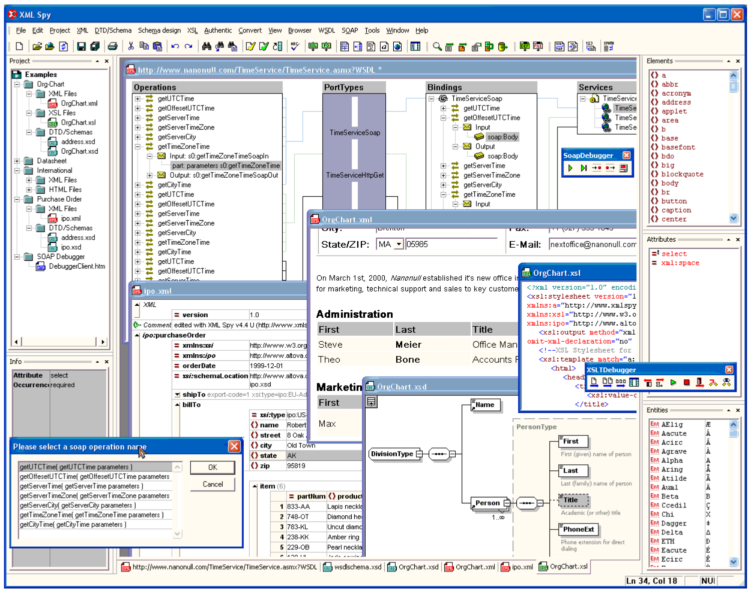
The application will serve several types of people in the development community:

1. Developers who understand a specific markup language but are not versed in the specifics of XSL development.
2. Developers or managers that are interested in developing prototypes before hiring a full time staff member to develop XSL.
3. Small development shops, that wish to exploit the benefits of XSL development without the staffing expense of hiring an XSL developer, or training developers to learn more XSL than they expect to use.
4. Developers using low memory devices or with special requirements not met by the more widely used XML and XSL IDEs.

It will do so by providing a simple, familiar interface. This interface will be guided by metaphors common to the most popular types of applications (e.g. word processors and HTML editors). It will offer easy, fast development of functional XSL.

Comparison of Related Products

Xmlspy®



(Figure 3 – Screen Shot of XMLSpy® IDE)

Altova's Xmlspy® is a quality product that has the majority of Windows development market share. It is a comprehensive IDE for the development of all XML related technologies. The product aids in the development of DTD's, XML, XSL, XDR, XML Schema and nearly every other XML related technology that is described by the W3C's recommendations.

Xmlspy® offers users:

- XML editing and validation
- Visual schema and DTD editing
- XSL editing and debugging
- Database connectivity and database content modeling tools

More Software than the Average Developer will Use

This wide range of support comes with several drawbacks. Providing this much support give the program far more functionality than the average user will use. Developers who are new to XSL development, and may be struggling with simple concepts like well-formedness, are highly unlikely to edit SOAP and WSDL. Likewise, developers who need help with XPATH are less likely to need SOAP editing. Since SOAP and WSDL are primarily used in advanced XML processes, such as web services, this is level of support exceeds the basic user's needs. SOAP and WSDL development code is also generated as part of the .net development IDE, making the need to edit such source code rare. Full-time XML and XSL developers may find themselves using many of the product's features, but many developers divide their time between XML and other technologies. For this reason provides far more functionality than the average user is likely to employ. This additional functionality comes at the cost of system resources, and if not used, represents a clear inefficiency.

High Level of Complexity

Supporting this wide range of features also comes at the expense of simplicity. A standard session of editing with XML spy may require oscillating between five window views. The tool provides source code views, data grid views, hierarchical views, and views specific to the language with the user is working. In sum, the user always has more than one way of viewing XML and XSL data, but not clear reason for using one view over the other. In some cases, certain command cannot be executed until a user switches from a view. These are artificial rules, in that they are imposed by the IDE, and only add to the user's difficulty in trying to complete their task. This unneeded complexity may be intimidating to inexperienced users.

Insufficient Xpath Builder

XML spy also offers a tool called the XPath analyzer. The tool is designed to aid in the debugging of XPath queries. Since there is no tool to help in creating queries, Xmlspy® has limited use to new developers. The tool could better serve new users by providing a fast, easy way to build XPath queries.

Xmlspy® is Not User Extensible

Xmlspy® literature touts its third party plug-in architecture. This allows other software developers to create software that can extend the functions of Xmlspy. Unfortunately, developers must understand how to create such plug-ins. Users who wish to use these plug-ins must find them and must trust the source to ensure that this new software will not compromise their system's security. Many applications provide a user extensible architecture that allows users to add functionality to the tool without developing software. For new users, user extensibility may be more useful than third party plug-ins. Especially if no third party software providers exploit this ability.

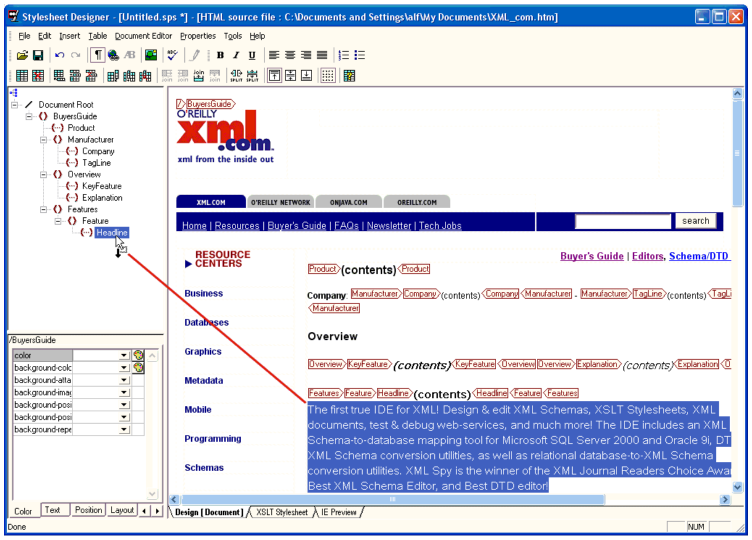
Product Price

Despite these short fallings, Xmlspy® is a costly solution. While Xmlspy sold for little over $200 in 2001, the latest Xmlspy® version costs more than $350. Enterprise editions of the software cost slightly more than $1000. Despite Altova's claim that "At

$399 for a single-user license, (Xmlspy® 5 Professional Edition) Xmlspy® 5 will generate substantial cost savings over competing products," (Altova) the product may still be considered expensive by most buyers.

For many companies, and individual developers, this is far too much to pay for software that will be used infrequently or to complete simple tasks.

StyleVision™


Stylesheet Designer - [Untitled.sps *] - [HTML source file : C:\Documents and Settings\alf\My Documents\XML_com.htm]
File Edit Insert Table Document Editor Properties Tools Help
Document Root
BuyersGuide
Product
Manufacturer
Company
TagLine
Overview
KeyFeature
Explanation
Features
Feature
Headline
/BuyersGuide
color
background-colc
background-atta
background-imag
background-posi
background-posi
background-repe
Color
Text
Position
Layout
Done
O'REILLY
xml.com
xml from the inside out
XML.COM
O'REILLY NETWORK
ONJAVA.COM
OREILLY.COM
Home | Resources | Buyer's Guide | FAQs | Newsletter | Tech Jobs
search
RESOURCE CENTERS
Business
Databases
Graphics
Metadata
Mobile
Programming
Schemas
Buyer's Guide | Editors, Schema/DTD
Product (contents) Product
Company Manufacturer Company (contents) Company Manufacturer . Manufacturer TagLine (contents) TagLi
Manufacturer
Overview
Overview KeyFeature (contents) KeyFeature Overview Overview Explanation (contents) Explanation
Features Feature Headline (contents) Headline Feature Features
Design [Document]
XSLT Stylesheet
IE Preview
NUM

The first true IDE for XML! Design & edit XML Schemas, XSLT Stylesheets, XML documents, test & debug web-services, and much more! The IDE includes an XML Schema-to-database mapping tool for Microsoft SQL Server 2000 and Oracle 9i, DT XML Schema conversion utilities, as well as relational database-to-XML Schema conversion utilities. XML Spy is the winner of the XML Journal Readers Choice Awa Best XML Schema Editor, and Best DTD editor!

(Figure 4 – Screen Shot of StyleVision™ IDE)

Altova also markets a product called StyleVision™. Its aim as described by Altova, is to allow, " Web designers with little prior knowledge of XML to easily create advanced XML websites" (Altova). This product offers the following main features:

- Visual design of XSL stylesheets
- Schema driven stylesheet design
- Separation of data and markup
- Integration with Altova's suite of XML related tools
- Customized forms and layout

Stylevision™ is one of the few products of its kind available. It's clear goal to simplify the development of XML and XSL by inexperienced users, is well approached, if not achieved.

Stylevision™ is *A Data Query Tool Not an HTML editor*

The product lacks the flexibility and familiar visual interface elements that most web designers are used to. The editor does not have an effective source editing tool and lacks other core elements that web designers would expect. Its general look and feel is more like a data query tool than a traditional HTML editor is. Quality HTML editors help users design aesthetically pleasing pages through tools and WYSWIG editing. Stylevision™ offers only a few tools, and focuses on the development of XSL code. Little attention is paid to HTML editing.

Stylevision™ Lacks User Extensibility

Like other Altova products there is little attention paid to user extensibility. However, unlike Xmlspy, Stylevison™ does not even support third party plug-ins. End users who wish to simply add to the library of markup tags, for example, cannot do so. Users must buy upgrades or other new software to add the simplest functionality to their existing program.

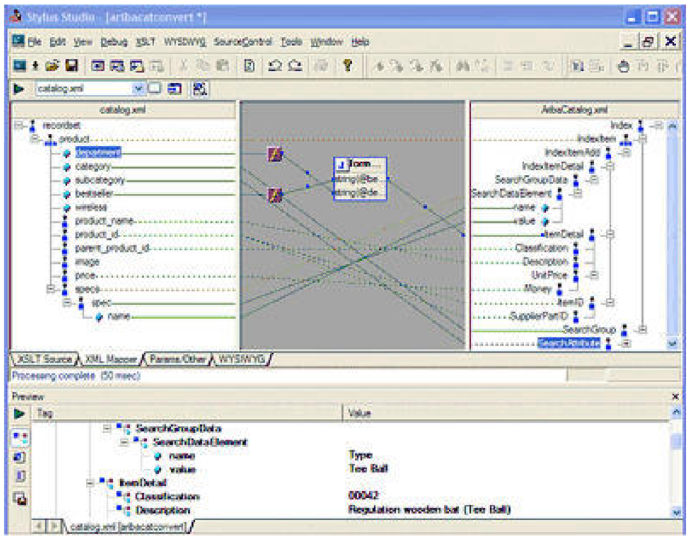
Stylevision's™ XPath Analyzer is Insufficient for New Users

Stylevision™ uses the same XPath Analyzer provided with Xmlspy. This tool, as discussed, does not help build XPath as much as it helps debug existing XPath queries.

Stylevision's™ Price

Stylevision™ is also an expensive product. Its base price of $299 for a single user license makes it impractical for small development shops and home users.

Sonic Stylus Studio


Stylus Studio - [aribacatconvert *]
File Edit View Debug XSLT WYSIWYG SourceControl Tools Window Help
catalog.xml
AribaCatalog.xml
XSLT Source
XML Mapper
Params/Other
WYSIWYG
Processing complete (50 msec)
Preview
Tag
Value

(Figure 5 – Screen Shot of Stylus Studio IDE)

Sonic Stylus Studio has been available nearly as long as Xmlspy®. It is used by nearly 30,000 developers worldwide (Sonic). The product was originally sold by the Excelon Company, which was acquired by Sonic two years into Stylus Studio's history (SD Times). The product is stable, but lacks the breadth of functionality that Xmlspy® and Stylevision™ offer. Stylus Studio's main features are:

- XSLT editor and debugger
- XML schema designer
- XML document editor
- XQuery editor
- Reverse engineer XSL from HTML
- Project and source control

Stylus Studio is marketed as a WYSIWYG XML-to-HTML editor. The application has seen a few refinements in its history but it relies very heavily on familiarity with XML and XSL. New developers might find it conceptually difficult to navigate. A single user license for stylus studio is $395.00.

Stylus Studio Lacks HTML Designer

The design of Stylus Studio reflects a bias toward data driven presentations. Like StyleVision™, Stylus Studio does not offer much for aesthetic designers who want the ability to make many visual changes. Instead, stylus studio is very good at helping a user extract XML data using XSL. This is only half of what the developer may want.

Supports XQuery, Instead of XPath

Interestingly, Sonic Software decided to support the W3C's XQuery instead of XPath. According to the W3C, XQuery is very similar to XPath. XQuery is an adaptation of XPath that affords developers a little more flexibility. The W3C recommendation was last edited on May 2003, making it a more current topic than XPath. Nevertheless, since XQuery is so new, some recent version of XML parsers do not support it. The Microsoft XML parser for example, does not support XQuery.

Open Source Development Initiatives

There are several open source initiatives for products that aid in the development of XSL. These include:

Treebeard:

Treebeard is an editor that supports XSL transformations. It is written in Java and may be used on Mac OS 10 and Windows. According to sourceforget.net, Treebeard is "a text editor that allows the loading and editing of an XML document and an XSLT . . .it also can apply the XSLT to the XML and display the output for further editing/saving." Work on Treebeard began in July 2002 (Source Forge).

Xsldbg:

Xsldbg is a debugging tool for XSL. It is described as "a debugger for xsl/xslt stylesheets which has functionality similar to a Unix/Linux 'gdb'" (Source forge). The application is limited to XSL debugging and does not offer visual tools for authoring XSL. Work on xsldbg began before September 2001,

One of the main problems with these open source solutions is that they are not geared to new developers. Upon reviewing the log of development history, it is clear that these two products are factioned development initiatives with sometimes unclear final objectives. The development goals seem to change. These two projects do not represent complete solutions. Treebeard lacks debugging, while xsldbg lacks an editor.

With both of these products, there is also the clear vulnerability that development may completely stop on any one of these projects before bugs are resolved or the project is complete.

Product Benefits:

This project endeavors to create an application that gathers the benefits of existing applications and extends those benefits by simplifying the user interface, increasing flexibility, extensibility, and by creating a product that is lightweight.

From a user perspective, the main reasons to use this new XSL editor over industry incumbents are:

1. The product is simple to use.
2. The product allows rapid development without a lot of set up or preemptive decision-making.
3. It consistently creates well-formed HTML, or other markup standards
4. The product is easily modified and updated to support other markup languages.
5. The product is a better value than industry incumbents are.
6. The product is lightweight

Simplicity of Use

The tool is simple to use because it provides an uncluttered user interface and common metaphors. The visual interface is basic, and a majority of the necessary decisions are made for the user as default settings.

In reviewing personal experience and software reviews, it is evident that complicated interfaces encourage confusion. While a plain graphical user interface may not be in fashion, it is often the hallmark of good usability. In following, this editor uses only the most basic elements. It does not use the 3-dimensional effects, gradient fills, and asymmetrical buttons that plague the latest applications.

Simplicity is also encouraged by limiting the number of choices the user must make. While such limitations would normally detract from an applications ability to meet more advanced usage, it should not for this application. Since this application is an editor, advance scripting decisions can be entered directly into the text editing window, allowing more advanced user to add their touches unobstructed.

Rapid, Simple Development through Abstraction of XSL and XPath Details

The users of this application will have to understand little about XSL. The program allows users to simply click and drag content into the document they are creating. Users familiar with these technologies can then tweak the resulting source code, as they desire.

Creating Xpath queries, for example, requires little knowledge of the Xpath language. Instead, the details of creating the queries are extracted into simple click and drag behaviors. Likewise, details about the individual markup language are extracted into toolbar icons. Knowledgeable developers can tweak the code created by the editor, or editing the document directly. If a user forgets a tag name, or its available attributes, the user can use the program's tag library as guidance. The complete application then demonstrates the strength of traditional HTML tools, blended with the XSL support of XML IDEs.

Consistent Development of Well-formed HTML

The editor's default mode is to produce well-formed HTML. It provides a real time indicator that identifies when a subject document is not well-formed or contains syntactical errors. This exceeds the capabilities of traditional HTML authoring tools, which tend not to produce well-formed HTML. In contrast, XML IDEs tends to provide real time well-formedness checks, but do not help in authoring HTML.

The Product is Easily Modified and Updated

This product will prove to be both extensible and flexible because of design decisions discussed in the following sections. Users of this product should find that adding a new library of markup tags is simple. They should also find it easy to customize the product to meet their specific needs. Both points are extremely important in promoting the long-term use of this application. While it would be simpler to produce an application with very specific support constraints, the application would prove far less durable in the tumult of industry changes.

Product Value

Since value is determined by price and quality, it is important that the application be developed against standard, proven methodologies and foundations. The software from which this application was developed has a stable heritage of revision and proven historical use. It may have been more enticing to treasure hunt for exotic software that offered greater functionality, but the cost of such software would be high. In researching, I discovered software components that exhibited erratic behavior, offered little technical support, but provided innovative functionality. Since the target user of this application is a novice user without the time to debug software, it made little sense to take the risks involved in incorporating such software into the application.

This XSL editor is created from a royalty free XML parser and controls. The software foundations are standard components that are used by multiple applications within the Windows operating systems. This allows the editor to be redistributed at lower cost than the industry incumbents. Much of the software used already exists on client machines. Much of the software has been stable and tested for years. For the end user this means reliability and a cheaper final cost.

Product is Lightweight

The product is lightweight, requiring few memory and processor resources. This makes the application a good choice for specialized development environments, low cost client machines, or other situations in which machine resources are scarce or need to be carefully utilized. If for example, a developer wishes to develop an XSL stylesheet from a handheld device, this application would suit there needs better than larger applications like Xmlspy. Products such as Opera Software's Opera web client and Irfanview's picture viewer have remained in use simply because the application takes less time to load and fewer resources to run. These lightweight applications offer paired down functionality, but meet the needs of enough users to warrant continued development. Irfanview, for example, has been used for nearly 10 years outlasting a variety of more complicated viewers.

Research Findings and Design

Software Design

The core design goals for this application are two of the most important goals in any software design plan:

- Extensibility
- Flexibility

Because many XML related standards are not mature, it is important that this application was designed with extensibility and flexibility in mind. As XSL grows, this application can grow. As the standards change, this application can change. This is also true of the markup language support.

The goals of extensibility and flexibility are encouraged by using the following technologies:

- An XML data source
- Microsoft's Component Object Model

XML Data Source

One of the most striking attributes of the design of this application is that it is reflexive. It uses XML to enhance its extensibility. The library of available tags for the authoring languages are stored in an XML file. This provides the benefit of easy maintenance and relatively low overhead. There are clear benefits to using XML instead of a database or file resource.

A database resource would require the installation of database software if it did not already exist on the client machine to which the application is installed. Since databases are whole sets of software in themselves, a database adds another level of complexity and increases the potential points of failure. As a data resource, only the retrieve functions of the database would be used. It is unlikely in a design like this that there would be much use for inserts, updates, and deletes. That means that only a small portion of the database software will be used for this application. For these reasons a database resource would be a poor choice.

A file resource, or flat file, is also not a reasonable option. The primary weakness of any flat file is its inferiority to databases in describing relationships. It is possible, but cumbersome, to describe data in detail through a flat file. Unfortunately, data quality must be managed by the application that uses the flat file. Databases, at least provide mechanism to ensure the quality of transactions. Connoly and Begg's description of the limitations of file based systems also highlight a wider range of concerns:

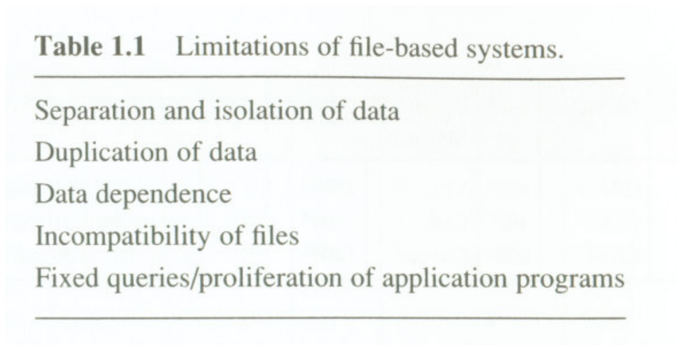


Table 1.1 Limitations of file-based systems.

Separation and isolation of data
Duplication of data
Data dependence
Incompatibility of files
Fixed queries/proliferation of application programs

XML brings the strength of both approaches together. XML can easily describe relationships. It has inherent quality checks (e.g. well-formedness constraint). It can be interpreted quickly with little overhead and far less software than a database.

Microsoft Component Object Model Based Design

The Microsoft Component Object Model, or COM, is a proprietary object model. Kirtland describes COM as " a language-independent, system-level object model that provides a standard way for components and applications to interoperate" (Kirtland, 10). Microsoft leverages it's Active X technology to create class based applications that are usable by most modern programming languages. These components are stored in dynamic link libraries (or DLLs) in the file system of the operating system in which they will be used. The application programming interface (API) to each of these objects is then exposed through a programming IDE (e.g. Visual Basic or C++) and accessible as properties, methods, and events of the component (Kirtland).

The component object model is an especially important technology for client server and 3-tiered architectures. Although this application is neither, the flexibility to which these architectures aspire matches the aspirations of this application's design. The component based architecture helps to provide a reasonable level of flexibility and potential for enhancement.

The COM objects allow for the abstraction of very low-level details. For example, the details of parsing an XML document are taken care of by the XML parser component.

If there are changes in the parser code, the old component need only be replaced with the new one. When the application is recompiled, the new component will be used. This simplifies the often-cumbersome process of version management and allows the developer to focus on building a solution quickly, from a set of atomic parts.

The program design requires several components created by Microsoft. These include

- **The MSXML parser** (*msxml.dll*):
  The XML parser from which all parsing is executed.
- **The Scripting File Systems Object** (*scrrun.dll*):
  The object that provides access to the windows file system for file access as well as the fundamental utilities of the windows scripting environment.
- **The Microsoft HTML object library**(*mshtml.dll*):
  The object that provides some of the support for HTML authoring and editing.

The Implementation

The application was written in Visual Basic because the language's simplicity allows for rapid prototype development. Writing in Visual Basic also allows the easy transitions to other derivative languages such as Embedded Visual Basic for Windows CE, or advanced languages like Visual Basic .Net.

The application's core capabilities are to:

- Parse, error check, and render XML and XSL documents.
- Create well-formed HTML, XHTML, or other markup.
- Integrate CSS, JavaScript and other client technologies into an extensible style sheet easily.
- Preview markup and design in a WYSIWYG environment.
- Use an XML schema for building XPath.

Since the world of markup languages is still immature, the application is designed to allow for design and development of XSL stylesheets producing various markup languages. This level of extensibility is achieved by the XML based resource.

Upon startup, the application reads an XML file and adds valid tags, as described by the XML, to its list of available markup. If the application needs to support a new markup standard, the user need only add the required XML document to the application's library of markup resources.

The schema of these XML documents is based on the W3C's schema for describing markup languages. The resource file for the list of valid XSL files, for example, is an exact copy of the XML document provided for download from the W3C's website. As the standard changes, a user need only download the XML file and replace their existing copy with the new one. This process can also be automated in revisions of the application.

A sample XML fragment for the *for-each* XSL tag is listed below. This sample represents one fragment in the entire XML file resource for XSL tags:

```
<e:element-syntax name="for-each">
        <e:in-category name="instruction"/>
        <!—Attribute for for-each element -->
        <e:attribute name="select" required="yes">
                <e:data-type name="node-set-expression"/>
        </e:attribute>
        <e:sequence>
                <e:element repeat="zero-or-more" name="sort"/>
                <e:model name="template"/>
        </e:sequence>
</e:element-syntax>
```

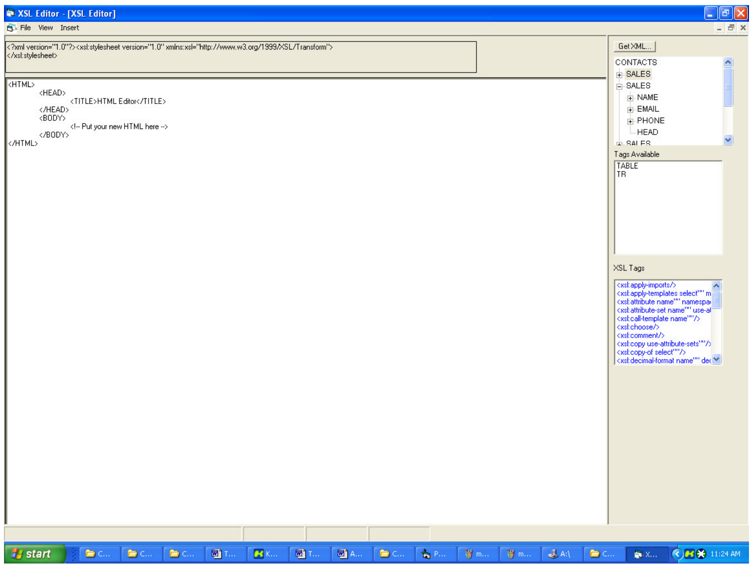
(Figure 6 – Sample XSL Fragment from XSL Tag Library)

The XML document resources combine the simplicity of use common to text files with the clarity and reliability of data common to database resources. The application parses these resources using SAX based parsing because the XML file resource can be large and only a subset of their content is required for use.

Once all of the XML document resources have been loaded, the user is presented with the graphical user interface of the editor.

Product Description:

Graphical User Interface Design



```
<?xml version="1.0"?><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

```
<HTML>
        <HEAD>
                <TITLE>HTML Editor</TITLE>
        </HEAD>
        <BODY>
                <!-- Put your new HTML here -->
        </BODY>
</HTML>
```

```
<xsl:apply-imports/>
<xsl:apply-templates select"" m
<xsl:attribute name"" namespa
<xsl:attribute-set name"" use-a
<xsl:call-template name""/>
<xsl:choose/>
<xsl:comment/>
<xsl:copy use-attribute-sets""/>
<xsl:copy-of select""/>
<xsl:decimal-format name"" de
```

(Figure 7 – Screen Shot of the XSL Editor)

The user interface design is derived from a study of the most common applications in the Windows operating systems. It reflects some assertion of fundamental design theory. The greatest spatial weight, for example, is given to the subject of most action, the document pane. Likewise, tools are centrally located and clearly indicated.

The prototype's graphical user interface is analogous to an artist's first sketches. The choices in the interface are made to be suggestive, not final. They are not in final ink form, but help to provide a sense of what may be. The GUI elements are more functional than an expression of aesthetic form. In order to demonstrate the potential of an application, user interface elements must be added. In most cases, these elements do not benefit the aesthetical qualities of the project as much as they aid the usability of that product. As mentioned in later sections, developing this project beyond the prototype stage would require more comprehensive application of user interface design and graphic design.

The following section outlines these decisions.

The User Interface

The application is based on a center pane user interface model familiar to users of Microsoft Word, Microsoft Front Page, and Macromedia Dreamweaver. There is a center pane in which a subject document is loaded, a right hand-side resources panel, a top mounted toolbar and menu section, and a bottom mounted status panel. All documents are edited from the center pane.

The center pane area represents the main tasks of the application. The right panel offers ancillary resources for development. The upper task bar is the center of action, and the lower status bar provides ongoing information. Each of the elements in a particular pane were placed there to contribute to this general theme. This arrangement is common to users of Macromedia's Deramweaver, Microsoft's Front Page, Microsoft's Interdev, and to a more limited extent, Microsoft Office products.

Center Pane

From within the center pane, the editor provides three core views to its users.

1. Editor

   The editor view allows text editing of the subject XSL document. It is also the view in which XPath queries are inserted into the XSL document. The editor was created using the Microsoft Rich Text control as the text editing area.

2. Visual Editor

   The visual editor provides a basic WYSIWIG editor for the XSL subject document. The visual editor was created using the Microsoft Browser Control and dynamic HTML. The browser control offers an API to the basic functions of Microsoft's Internet Explorer. To enable visual editing of an HTML document, the elements of the document are manipulated through JavaScript and convenience functions offered through Microsoft' scripting engine.

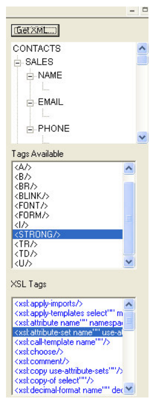
3. Browser View

   The browser view provides a preview of the document. It executes the XSL against the provided XML document and renders the result in an HTML browser. The browser view is also based on the Microsoft Browser control. It simply uses the control to display a document after the XSL transformation has been executed.

The editor view is the default view.

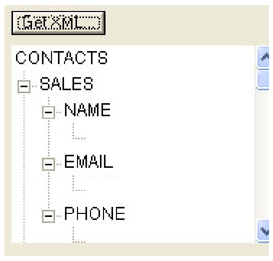
Right Panel

The right panel is best described as the resource pane. It contains the group of controls that offer editing resources. These resources consist of the XML resource, the Markup Resource, and the XSL tag resource. Since these resources are the main reason for developing the software the right panels is prominent and always



(Figure 8 – Screen Shot of XSL Editor Right Panel) visible.

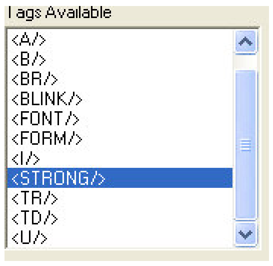
The XML Schema Resource:



(Figure 9 - XML Schema Resource Screen Shot of XSL Editor)

Figure 9 is a tree view of the sample XML document that the user selected. This tool helps in authoring XPath queries by demonstrating the relationship between elements. Users may double-click an element from this view into their document to add the respective XPath. The resource is always visible, making access to the XML and XPath queries simple and straightforward. Since this is one of the main reasons for developing the software, it only makes sense that the tool is readily available.

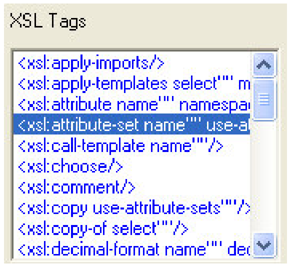
The Markup Resource:



(Figure 10 - Markup Resource Screen Shot of XSL Editor)

The markup resource (figure 10) is a simple list of all available tags for the loaded markup language. Users can click and drag from this list to add a new tag to their XSL document. The list is created dynamically every time the program is run. If a user wants to add to this list, they need only edit the XML document by adding the new tag anywhere in the library resource document.

The XSL resource



```
<xsl:apply-imports/>
<xsl:apply-templates select"" m
<xsl:attribute name"" namespa
<xsl:attribute-set name"" use-a
<xsl:call-template name""/>
<xsl:choose/>
<xsl:comment/>
<xsl:copy use-attribute-sets""/>
<xsl:copy-of select""/>
<xsl:decimal-format name"" dec
```

(Figure 11 - XSL Resource Screen Shot of XSL Editor Panel)

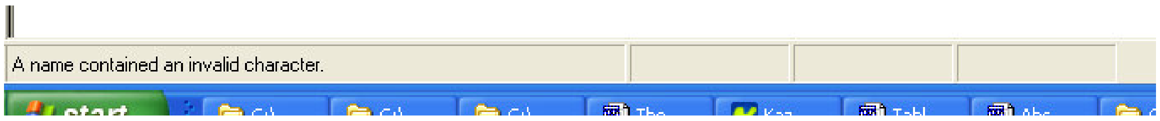
The XSL resource (figure 11) is a simple list of all available XSL tags. Users may double-click items on this list to add a new tag to their XSL document while in edit mode. This list is built dynamically.

Menu and Toolbar

The menu and toolbar are straightforward. The menu provides access to the standard file functions; Create, New, Open, Save, Save As. It also provides the means for switching between views, text editing, and exiting the program.

The toolbar provides the basic HTML editing functions and a second interface for the text editing functions; Cut, Copy, Paste.

Status Panel


A name contained an invalid character.

(Figure 12 - XML Editor Status panel

The most important element of the status panel is the XML well-formedness check. As a user is editing their document, the status panel will indicate whether the document is well-formed. If the document is not well-formed it will display a short description of the problem. For more details on the problem, the user can double click on the description in the status panel.

Building the Graphic Design Elements

The application's visual design was drafted in pencil and paper first. I reviewed a number of programs that had received accolades for their general usability, and then built upon their general layout. I also used programs that were successful at providing their users with a comfortable environment in which to develop. I tried to review a mix of programs involved in editing and design. The general hypothesis was that a design programs aid in aesthetic decisions making, while source code editing programs aid in the general task of creating and revising code. These programs include:

- Adobe Photoshop
- Macromedia Dreamweaver
- Microsoft FrontPage
- Microsoft Word
- Microsoft Notepad
- Microsoft Visual Basic
- Microsoft Wordpad
- Macromedia Freehand.

Once I had decided on a general layout for the user interface development tools, I used digital media tools to create specific graphical elements. The splash screen graphic and program icon were created using the Persistence of Vision Ray (PoV-Ray) Tracer. It was hoped that ray trace graphics would provide the updated look that invites users to feel confident and comfortable with the software they are using. PoV-Ray also offered

- A comfortable, programming oriented authoring environment e
- Free, high end graphics tools

I also used Microsoft Paint to author the icons created for the design view toolbar. I used Paint because it is easy to use and costs nothing for owners of the Windows operating systems. The graphics are recognizable crude, but for a prototype, sufficiently communicate their meaning.

The two icons on the project selector page were copied from the Windows XP icon library. The windows XP icon-editing standard, as published by Microsoft, indicates that icons should be designed in high resolution and with three-dimensional perspective. Since these images are designed against that standard, they also add to the look and feel of the software. The icons communicate graphically, the action of creating a new file and opening a new file respectively.

Using the Program

Using the editor is very straightforward. Users may create a new document, open an existing document and save documents using standard menu items and shortcut keys. The Microsoft File System Object and the Common Dialogue control are the foundation for these file operations. The Microsoft File System Object facilitates the foundation for writing to, reading from, and creating text files. The Common Dialogue control provides the graphical interface common to file selection for file *open*, *save*, and *save as* functions in the windows environment

File Open

Files may be opened from any local or network drive to which access is allowed in the Windows operating system. Since the program is built against the Common Dialogue control users should experience no difference between other applications in their ability to open files.

When files are opened, they are immediately split into two sections. The first section is the XML and XSL header. This section contains the XML declaration and the XSL namespace declaration and related content. The second section is the body of the XSL. This section contains the main content of the subject document including all elements child to the XSL document definition element. When a user edits a document, they are only editing the body of the XSL document.

There are two primary benefits to splitting the XSL file. First, it allows the document to be edited and validated as a freestanding document in XML format. Because Microsoft's Internet Explorer is an XML enabled browser, it will automatically treat any document with the XML declaration as an XML document. This means that when a document is previewed it is previewed as XML, not as the markup language with which it is written. By removing the XML and XSL document identifiers, this problem is avoided.

Secondly, by removing these two header sections from the editable region in an XSL document the user is protected from the small syntactical errors that disrupt the validity of their XSL document. If for example, a single letter in the XSL namespace declaration contains the incorrect case, the document will not be recognized as a usable XSL by some parsers. For beginners this can be a very frustrating fact.

Once a document is opened successfully, it is parsed. If there are errors in the document, those errors are described in the error pane.

File Save

File *save* and *save as* are simple text file write functions. They concatenate the XSL header and the XSL body back together and save the user's work in a single text file. By default, all files are saved with the .xsl extension. Users can change the file extension, as they would be able to do from any Windows program. They can also save over an existing document, or navigate to a mapped network drive or virtual directory.

File New

File new uses the file *open* functions to open a selected template file. The template file contains a valid XSL header and basic body content. Users may create their own templates by creating a standard XSL file, or by copying them from other resources. The main goal in using template files is to allow users the ability to extend and customize the program. More experienced users can create customer templates and provide those template to others. If for example, each XSL document should have a similar look and feel, then a user can create a template and use that template for subsequent development. The notion of templates should be familiar to users of Microsoft Word and Macromedia Dreamweaver.

Validation and Well-formedness

All content in the subject document pane is subject to parsing at every edit. As the user types new content, for example, the content of the edit pane is loaded into a Document Object Model. If there are any errors thrown by the parser's error object, the description of that error is abbreviated and expressed in the error panel. When a user wants more specific information about the particular error they can double click on the error pane. The error will then be fully described and the location of the error will be highlighted in the edit view pane.

Creating XPath

To create XPath queries the user should load a sample XML document into the XML schema pane. To insert XPath into their XSL document, the user then needs to double- click on the element they wish to query using XPath. The analogous XPath will then be pasted into the edit view panel of the XSL document with which they are working.

More detailed instructions on using the editor follow in the sections entitled Using the *Editor: Basic Walkthrough* and *Test Case Scenarios*.

Using the Editor: Basic Walkthrough

The editor is designed to be used like common document editing programs. The user typically begins by creating a new document. The easiest way to create a new document is to select the *create new file* button from the new project modal window. Users may also create a new document by selecting *new* from the file window.

After selecting the new file button the user document window will contain a basic HTML file from which to start authoring. This document has only the HTML declaration tag and the HTML body tags, along with a comment specifying where new HTML should be placed. At this point, the user can add any code they wish to the document. If the code does not conform to the XML standard, the status bar in the lower left hand side of the application will describe the error. If the user is still uncertain of the error, double clicking the error status panel will highlight the approximate location of the error being described.

If the user needs to add code to the document, they may do so by directly typing the code into the document-editing window or by double clicking any of the tags in the tag resource windows of the resource pane.

If the user intends on using XPath then they should click the *get XML* button in the right hand portion of the user interface. This allows the user to select an XML document from which XPath queries will be determined.

To use the XPath builder users need only select an item from the hierarchical view of the XML document. A single mouse click adds the *XSL value-of* tag with the correct XPath query to the cursor position of the document window.

The user may also edit their document by using the design view windows. To enter the design view, the user must select *design view* from the *view* menu. The design view provides visual editing capabilities instead of text editing capabilities. In design view users can click and drag to edit the appearance of the document. The design view does not allow XPath editing.

Once a user is content with the changes to their document, they can view the results through the browser view. To do so the user must select *browser view* from *view* menu.

In the course of editing, a user can save their work by choosing *file save* or *file save* as from the file menu. To close the application the user must select *exit* from the *file* menu or click the Windows close icon in the upper right hand corner of the application window.

Research Findings and Limitations

Test Case Scenarios and Description

Case 1: *Create a New XSL File*

The user should start by running the application. The new file dialog should appear in the center of the user screen. Begin by choosing a template for XSL construction. The list in the upper left hand of the new file dialog window represents the available template files. Choose one by clicking on it. This will update the file name text box immediately below the template list box. When the user is content with the new file name, click the *create new file* button in the lower-right of the new file dialog window.

The file is now ready to be edited. There should be a very simple document in the document window. Begin by typing the word "test" to the immediate right of the text that reads "<!—put your HTML here—>." Note that there should be no XSL errors in the status panel. Now being to make the word test bold, by typing the HTML start tag "<B>" directly before the word test. The status panel should read, "end tag 'body' does not match the start tag 'B'" This message indicates that the document is not well-formed because it lacks the required closing "B" tag. Double-click the status bar panel to indicate where the problem exists and learn more information about the problem.

To correct the problem type "</B>" immediately after the word "test." Now use the WYSWIG editor. Select *design view* from the design menu. Select the word "test" and click the underline button (marked by a U) on the toolbar. The editor has now added the underline tag to the document. To view your changes switch back to editor view by selecting *edit view* from the *view* menu.

While in view menu, add an XPath query. Click the *get XML* button on the upper right-hand side of the editor. Select the file named, sample.xml. Place the mouse cursor directly after the word test in the subject document window and click-once. This will place the cursor, so that the XPath query is inserted directly after the word "test." Now select the first node in the XML tree view window. An XSL value of statement and an XPath query should now be placed beside the text document.

Save the document by clicking save from the file menu.

Case 2: *Using the Editor to Edit an Existing Document*

The user should start by running the application and selecting open existing file from the new file dialog box. A file open dialog box should appear. Navigate to the file to be opened, and select it by either double-clicking the file name or by selecting the open button on the lower right hand side.

If the file is well-formed, the document will be loaded into the editor. Once the file has completed loading, the user should select Design view from the View menu. The document should now display as rendered HTML. From the design view the user should use the bold button to emphasize a section of text. To do so, select the desired text with the mouse and click the *bold* button in the formatting tools toolbar. To view the effect of the change, select *edit view* from the view menu.

Now attempt to insert an XPath query. From within the edit view, click the get XML button on the right hand side of the editor. Select a sample XML document from the file open dialog. Return to the subject document window, and place the text cursor where the user would like an XPath query to be inserted. Using the XML schema window, select an XML node to insert. An XSL value of node will be inserted with the corresponding XPath query.

Save the file by clicking *save* from the *file* menu.

Limitations in Scope

Design and Development

The functional application serves only as a prototype for a more complete solution. Many of the functions and features of the application have been implemented in an exploratory way. The implementation, and all related code, purposes to demonstrate the potential of the described design. The source code for the application does not represent a finished application, as much as it means to prove the possibility of specific features and functionality.

It is clear that an application of this nature would require the benefit of a full team of design and developers to truly compete with applications that have been built by such teams. Respectively, the scope of this project is limited to the efforts and abilities to which I am capable in several months of research and development.
Recognizing that the application is neither required to be, nor capable of being developed to a final product within a reasonable amount of time, the project has easily recognizable limitations.

These limitations include, but are not limited to:

Development for a Windows Operating System

Because a vast majority of the computer systems used for business and personal use are Windows based machines, it made sense to develop the application in Windows.

Alternative operating systems, such as Linux and the Macintosh's OS X, would have hindered this enterprise. According to several sources the Linux, operating system offers few quality IDE's for developing programs of this nature. Macintosh's OS X would have forced the application to be used by a limited group of individuals simply because the operating systems has lower market penetration than Windows.

Lack of Comprehensive Visual Theme and Graphic Design

In the interest of time, little energy was spent on the aesthetic elements of the application. A few icons were gathered from royalty-free resources. I did pay attention to the essentials of graphical user interface design, but successive revisions of the application would benefit from a trained professional's artistic hand.

Lack of Multi-document Editing within a Single Instance of the Application

A multi-document editing interface is one that allows multiple subject documents to be edited from within the same application. Supporting multi-document editing within the editor would have added to the programmatic and user-experienced complexity of the applications. Documents such as Windows Notepad or Windows Paint are simplified by the ability to edit only a single document at a time. For novice users the decision to not support multi-document editing interfaces means ease of use. As such, this application does not have a multi-document editing interface.

<u>**Limited Markup Support**</u>

To expedite coding the program has only an HTML resource from which to gather markup tags. Creating a XHTML, WML or other markup resources would not contribute to the proof that this application's hypotheses are reasonable. The single HTML resource proves the flexibility of the XML based resource as well as five more resources would.

<u>**Development in Traditional Visual Basic Instead of .Net Technologies**</u>

The application was developed using traditional Visual Basic. It is immediately apparent that an improved application would be developed in .Net technologies. At the start of this project, .Net development tools were neither inexpensive, nor well distributed. Developing in Visual Basic . Net, for example, would have meant a hefty investment in the .Net framework IDE, and use of a limited number of development resources.

Likewise, development of the application in ASP.net would have meant sparse development resources and a change in the general approach to the problem. Simply, a web based version of this application would be widely divergent from the XML editing incumbents, and awkward to use. This project was designed to run on a client machine because the overhead of parsing and editing a document across a network would be slow and cumbersome. This is particularly true of .Net's reliance on post back for control validation and form submittal.

As networking technology improves this may become more reasonable. In that time, the amount of literature on developing in .Net technologies may have grown. By then it may be reasonable to reassess the redesign of this application using a .net paradigm in the .net framework.

.Net controls and other developer resources were also limited at the start of this project. Microsoft provides a wizard for the conversion of legacy Visual Basic code to .Net Visual Basic code. When the time is right, a source code conversion, if not a redesign, can be quickly executed.

A more committed effort would involve the transition of this software to the web services model. The bulk of the processing code in this application sits in Visual Basic modules and classes. As such, converting to web enabled, or web service product would be reasonable and straightforward. The primary hurdle is in a redesign of the user interface.

Client Implementation

For several of the network limitations listed above, this project is strictly designed as a client based application. Its lightweight and simplicity should allow it to be easily converted to embedded Visual Basic for Windows CE, or other systems in which lightweight editing supercedes the need for comprehensive functionality.

Hard Coded Paths and Other Portability Concerns

To simplify production and standardize installation, this program has a small list of hard coded file paths. In total, the three paths represent the following:

- The file path to the file resource for XSL tags
- The file path to the file resource for Markup tags
- The file path to the available document templates

These hard-coded file paths encourage a standard installation accross machines, making it easy to find, replace, and update file resources. When the application is developed beyond the prototype, these paths can be stored in XML resources or the Windows Registry.

Lack of Help Files, Tutorials, and Other User–centered Support Documentation

Since the editor is merely a prototype, there is no user supporting documentation. When the program is ready for general distribution supporting documentation would be required. The aim of this project was to demonstrate potential, and as such, the existence of standard documentation does not add to the claims of this endeavor. The *application use* sections of this document represent the only user-centered documentation.

Developed Against a Microsoft Version of the Parser

As discussed, the source code for this project was developed against the Microsoft XML parser. There are clearly alternative parsers in the software marketplace. The low cost, fidelity to W3C standards, and the library of documentation made the Microsoft parser a quality choice. Future development may benefit from choosing a more lightweight XML parser or by choosing one that is not specifically designed by the makers of the Windows operating system. For prototyping purposes, the XML parser makes sense.

Development Using Freeware and Existing Components

Many of the controls and components used to develop this editor are provided as part of the Windows operating system. Other components were freeware. This kept cost of production down, and provides the editor one of its core benefits; low cost. If this were not a prototype, it may make sense to investigate more complete software solutions.

Lack of Direct XSL Transformation

The editor prototype does not support direct transformation of an XML document by the subject XSL template. The reasons for this are technical, not conceptual. The program uses temporary files instead of direct memory addressing. To support direct XSL transformations there would need to be a way to load both the XML file and XSL file into memory, execute the transformation, and load the result into an additional temporary text file so that the Microsoft web client control could display the document. Such processing is possible but it would require the user to specify an XML document for transformation even when the user is not executing Path queries. Since the browser, view should support both XML dependent and XML independent browser previews I create the simpler of the two. The browser view currently only supports XML independent previews. This means a user can view their changes, but they cannot view their changes, as they would be applied to an XML file. Supporting this behavior would require either an additional control (one for XML dependent previews and one for XML independent previews) or some switch for toggle the two types of previews on or off.

Since that change significantly complicated the user interface and the source code of the application, I decided to leave it incomplete. In its current form this make the XSL editor more effective for authoring XSL than testing and debugging XSL. Users can preview their changes against an XML file by adding a reference to the XSL they have authored and previewing that XML document in an XML parsing browser such as Microsoft Internet Explorer.

Lack of Line Numbers, and Other Typical Editor Elements

The editor was created using a Microsoft common control called the Rich Text control. This particular control is commonly dispersed on Windows machines and offers comprehensive documentation for developers. It is proven stable and because it already exists on many Windows machines improves the application's portability and ease of installation.

The software market offers several high quality editor text controls that provide more functions. There were several reasons not to develop against these other controls. The text editor controls cost as much as $500 for unlimited distribution. Some controls were offered by companies or individuals for which there is no indication that they will continue to do business or provide support for their software. The Microsoft controls offer proven support, and are provided by a company with a clearly stable future.

Likewise the software has proven its stability through repeated use, where trial versions of other text controls evidenced unexpected and erratic behavior.

Successive versions of the applications should consider the investment in a more capable text editing control. The control provided by Anthony Dunleavy, for example, may be a good choice for a more finished application. Dunleavy's control is described as a rich text code box providing a line number indicator and advanced code-editing functions.

Implementation

WYSIWYG Editor Only Supports HTML

For demonstrative purposes, the editor only supports WYSIWG editing of HTML. Future enhancements would require the additional support of other browser viewable markup languages. This task will require considerable research and coding which exceeds the scope of this project.

The Applications Only Supports the “XSL” Namespace

Technical limitations in the Microsoft parser require that only “XSL” namespace and namespace declaration be used. Microsoft’s parser did not support the simple substitution of a DOM document’s namespace declaration or a specific tag's declaration. A significant amount of code was produced simply to support accurate communication of parse errors that relate to namespace errors. When Microsoft’s parser supports write access to the namespace object, a significant amount of that code can be removed. At that time, the editor will be able to support any namespace a user decides.

The inability to use anything but the XSL namespace, also simplifies the editing process. The theories involved in namespaces may be complicated for novice users. This limitation therefore contributes to the program’s simplicity of use.

Relies on Temp Files for WYSWIG and Preview Mode

The program relies heavily on temporary files to support WYSIWYG editing and document previews. The free Microsoft software that allows this functionality is file based. AS such, files have to be created to allow users to edit or preview their changes. Future development may afford investment in higher quality software infrastructure to support WYSIWYG editing and preview independent of temporary files.

Does Not Support Axis Queries

The program only supports simple XPath queries. Help in creating axis-based queries is not supported, but the program will support such queries if they are entered into the editor. Creating an axis query builder was too complicated and time consuming for the demonstrative nature of this project.

Limited XPath Control and Query Structuring

The application prototype endeavors only to demonstrate the potential in its design. The sample XPath generated by the application is not means to demonstrate the full breadth of reasonable queries. As such, queries are inserted using the default XSL value of element. In practice, such queries would have optional parameters specifying the kinds of XSL element to use with the specified query. A list of radio buttons for example, would allow the user to specify whether they want to insert a template query, an XSL for-each query, an XSL copy-of query or other related elements. Future development of the application would require a more complete effort in this direction.

<u>XPath Queries Only Supported in Edit View</u>

XPath builder is only support in the edit view for this implementation of the editor. Future implementations should offer the XPath builder utility in the design view. This additional functionality was left out to simplify development of the prototype.

<u>Lack of Intellisense Generic Solution</u>

Many successful IDEs and editors provide a user-prompting element commonly known as Intellisense. The device is used frequently. When a user edits source code, the IDE displays a list of context sensitive prompts. If for example, a user types the beginning of a font tag in HTML, an Intellisense capable editor would list all of the attributes available for the font tag. The user can then select an attribute from the list of available attributes. The foundation for Intellisense enabled editing is already a part of this prototype. In this implementation a Visual Basic Combo box control is populated with al of the available tags. Unfortunately, the design requires a way to dynamically locate a control based on the position of the user text cursor. The Microsoft Rich Text control does not provide programmatic information to coordinates of the active cursor. The Windows user API information does provide X and Y coordinates of the active mouse, but locating the Intellisense prompter to the mouse pointer location would only create confusion. For demo purposes, the Intellisense device is disabled in the prototype. When considering the alternative text editing controls, the requirements of Intellisense device as designed will be considered.

Summary

The purpose of this project was to explore the possibility of creating an extensible style sheet editor for a specific set of development needs. These development needs included:

- Developers who understand the fundamentals of markup languages but are not versed, or do not have the time to be trained, in the specifics of XSL development.
- Small development projects, that wish to exploit the benefits of XSL development without the staffing expense of hiring an XSL developer, or training developers to learn more XSL than they expect to use.
- Developers using low memory devices or with special requirements not met by the more widely used XML and XSL authoring tools.

The existing XML and XSL authoring software fails to meet these development needs. The existing software is limited in variety, usability, and value. Software applications such as Xmlspy® demonstrate design decisions targeted toward meeting the needs of experienced, well-educated XML developers. These applications require a considerable amount of background before an individual user can effectively use them. The XSL editor produced by this project demonstrates the possibility of a program that is simpler to use and continues to offer the core functionality that the development community requires.

Research into the standards of XML, authoring XSL, and related technologies was conducted to guide the project's efforts. The examination of an evolving set of markup standards calibrated the program's design toward extensibility and flexibility. These design principles help to ensure the durability needed for an XSL editor that supports multiple markup languages. In implementation, the program uses the extensible mark up language as a data resource for the retrieval of markup standards. This provides great benefit because developers will be able to tailor the application to their specific needs without the complications of installing additional software, using a macro recorder, or recompiling the application source code.

While this application is merely a prototype it succeeds in demonstrating that a simple approach ads benefit. There are a myriad of enhancements that should be implemented when the project is moved beyond the prototype stage. These are described in detail within the limitations of scope section of this document. The most notable of these enhancements include an improved graphical user interface, a more compressive set of tools for the graphical editing of XSL, and the potential conversion of this project from a Visual Basic 6 to .Net Technologies or Embedded Visual Basic.

Appendix A

Visual Basic 6.0 Source Code

(Code not provided in this digital copy of thesis)
Source code provided on compact disc

Annotated Bibliography

Altova, The Xmlspy Suite, 5 May, 2003
<http://www.altova.com>

Altova.com is the official website of the software manufacturer for the Xmlspy and Stylus Studio products. The manufacturer discusses the design goals and target audience for these products through marketing materials and information. The website provides information about Altova's intentions in developing and marketing the Xmlspy suite of software.

Begg, Carolyn and Thomas Connolly(2002), Database Systems, Essex, England:Addison Wesley

Connolly and Beg discuss the fundamentals of database design, history, and implementation. The book introduces the benefits and pitfalls of using database systems. Core topics relevant to the research of this project included:

- Relational models
- Database analysis and design
- Emerging Trends
- Semi structured Data and XML

Britt, James and Teun Duynnstee, Profesional Visual Basic 6 XML, Birmingham, UK: Wrox Press, 2000

An overview of the core concepts and practices for programming applications that use XML from within the Visual Basic 6 environment. The authors demonstrate best practices and provide essential source code. Core topics covered by the book include:

- Introduction to XML
- XML Document object Model
- Maximizing Performance of XML Applications
- Storing and Retrieving XML Data

Hunter, David, Kurt Cagle, Chris Dix, Roger Kovack, Jonathan Pinnock, Jeff Rafter, *Beginning XML*,Birmingham, UK: Wrox Press 2002

This book gives provided a fundamental overview of XML and related technologies. The publisher provides a website that offers source code for developing applications using XML and related technologies. The SAX parsing class used in the XSL editor was obtained from that website.

Kirtland, Mary, *Designing Component Based Applications*. Washington: Microsoft Press, 1999

A detailed overview of Microsoft's Component Object model as it relates to development in C++ and Visual Basic 6. The book offers instruction on the core concepts of designing component based applications and best practices in implementation.

Microsoft Software Developers Network, XML Development, 29 April 2003 <http://msdn.microsoft.com/xml/default.asp>

The Microsoft Software Developers Network (MSDN) is an online resource for software developers using Visual Basic, C++, XML and a wide variety of other languages. The MSDN website provides sample code and technical data about Microsoft's software. This includes information about specific controls, components and known issues in software.

Stylus Studio, Sonic, Vers. 5
2003, Windows XP, download
<http://www.sonicsoftware.com/cgi-bin/sonic.cgi/download_menu>

Sonic Stylus studio is a relatively popular XSL authoring tool. By installing the program and experimenting with its interface, I was better able to judge the quality of the application.

Stylevision, Altova, Vers. 5
2003, Windows XP, download
< http://www.xmlspy.com/products_xsl.html>

Stylevision is part of the Xmlspy suite provided by Altova software. Stylevision is not nearly as popular as Xmlspy, but the product's main goals are similar to this project's goals. The software is aimed at easing the process of developing XSL documents. By installing the program and experimenting with its interface, I was better able to judge the quality of the application.

Redish, Janice C., User and Task Analysis for Interface Design,
New York:.Wiley, 1998

This book has a few chapters on best practices in user interface design and development. Although the book's primary aim is to introduce concepts of task analysis, it provides a reasonable amount of information about the designing for usability.

Source Forge .Net, XML Projects, 5 May, 2003
<http://sourceforge.net/>

Source Forge is one of the largest collections of open source software on the Internet. Last reviewed on May 5, 2003, this website maintains a database of major open source software initiatives. Source Forge was the primary resource for identifying XSL related software available in the open source community.

The World Wide Web Consortium, Extensible Markup Language, 10 April 2003
<http://www.w3.org/XML/>

The World Wide Web Consortium (W3C) provides information on standards development for a range of Internet protocols and standards. The W3C recommendations provide detailed information about particular technologies and the concepts behind standardizing decisions. The W3C's XML recommendation informed much of the XML research for this project.

The World Wide Web Consortium, Extensible Markup Language, 15 April 2003
<http://www.w3.org/Style/XSL/>

The W3C's XSL recommendation informed much of the XSL research for this project. The website provided the library of XSL tags in XML format and guided development of the XSL editor.

The World Wide Web Consortium, Document Object Model, 15 April 2003
<http://www.w3.org/DOM/

The W3C's Document Object Model recommendation informed much of the DOM research for this project. It also framed an understanding of the Microsoft implementation of the DOM and indicated potential short fallings in Microsoft's design.

The World Wide Web Consortium, Markup, 5 May 2003
<http://www.w3.org/MarkUp/>

The W3C's markup recommendation informed much of the markup research for this project.

The World Wide Web Consortium, XPath, 12 May 2003
http://www.w3.org/TR/xpath

The W3C's XPath recommendation informed much of the XPath research for this project. The website's explanation of the XPath language helped in developing the software to create XPath queries.

W3 Schools, XSL Tutorial, 11 April 2003
<http://www.w3schools.com/xsl/default.asp>

The W3C schools are a tutorial-based collection of web pages designed to aid in the instruction and understanding of W3C standards. These tutorials provide vender independent perspective on implementing W3C standards in software. The XSL tutorial was useful in establishing a basic understanding of the technology.

W3 Schools, DOM, 20 April 2003
<http://www.w3schools.com/dom/default.asp>

Another section of the W3C School website, this DOM tutorial provided a basic, platform independent discussion of DOM technology

W3 Schools, DOM, 20 April 2003
<http://www.w3schools.com/xhtml/default.asp>

Another section of the W3C School website, this XHTML tutorial provided a basic, platform independent discussion of XHTML technology. Some of the research presented here was derived from this XHTML tutorial.

XML Spy, Vers. 5, Altova
2003. Windows XP, download
< http://www.xmlspy.com/products_ide.html>

Xmlspy is the leading IDE for XML development. By installing the program and experimenting with its interface, I was better able to judge the quality of the application.