Electronic Journal of Vedic Studies

Volume 6 (2000), Issue 1

Virtual Vidhāna

E-Textnology (XML and XSL/T) and Vedic Research

John Robert Gardner

ISSN 1084-7561
DOI: http://dx.doi.org/10.11588/ ejvs.2000.2.1520

Virtual *Vidhāna*:

E-Textnology (XML and XSL/T) and Vedic Research[1]

John Robert Gardner, Ph.D.
Architect
Sun Microsystems

Introduction

This article explores potential of electronic text technology, specifically Extensible Markup Language (XML) and Extensible Stylesheet Language for Transformation (XSL/T), in the study of Vedic electronic text resources. Its original form, or companion version, can be found in the *International Journal of Tantric Studies*, cross-linked with this article. The reader is advised to read some of the introductory sections, especially regarding XML, as the syntax of that markup language is pivotal to understanding the workings of the XSL/T tools presented here. In addition, due to the necessity for clear explanation of this technology, a detailed, incisive, and otherwise comprehensive treatment of the Vedic concepts discussed here is not possible. In fact, the Vedic notion of *vidhāna* (discussed below) is offered more as an analogue to XSL/T's function rather than as an object of direct inquiry in and of itself in this paper.

Not unlike the competing demands for developmental time that a scholar of the humanities finds when trying to add digital technology to their arsenal of research techniques, this article strives for an appropriate balance. Accordingly, the citations related to mantra use are points of academic reference rather than a suggestion of rhetorical conclusiveness.

It is the objective and intent of this paper that the reader will download the materials linked from this article and begin working with them.[2] First it is recommended to replicate the examples by themselves, then gradually begin varying the script codes (sections you can edit are clearly marked) for your

---

[1] The first two sections were adapted from material previously prepared for http://vedavid.org/xml and ATLA-CERTR, at http://rosetta.atla-certr.org/CERTR/ATLAS/index.html and http://www.oasis-open.org/cover/atlas.html.

[2] http://crossasia-journals.ub.uni-heidelberg.de/index.php/ejvs/rt/suppFiles/1516/0, also mirrored at http://vedavid.org/xml/docs/.

Electronic Journal of Vedic Studies, Vol. 6, Issue 1, 2000.
DOI: http://dx.doi.org/10.11588/ejvs.2000.1.1516

own needs. Second, readers are highly encouraged to discuss their questions and work with this material on the Indology list serve. It's been a while since digital research methodology has become sufficiently powerful to move beyond discussion of fonts, and there is great potential in XML and its related technologies, which I term "X-nology."

A final preliminary note, as I've touched on the fonts issue, regards the sample Rig Veda, in XML, which I am providing with this article. The scheme for the text is ITRANS (e.g. "aa" for long a, etc.) as XML is case sensitive and this simplifies the processing. The file contains *udātta* accent markings (with semicolons ";"), but for the word search it might be best to make a copy in which you strip the semicolons if they are not needed. Mind you, this is more than traditional word searching, XSL/T enables contextual word searches—e.g., occasions of *asti* in hymns about Indra, attributed to Viśvāmitra. It should be noted, however, that XML fully supports Unicode in the UTF-16 set, but for research it is faster and the tools are free to work in simple transliteration.

The sample XML RV file has mandala, hymn, verse, and mantra marked with each given its own unique identifier (e.g., rv1, rv1.1, rv1.1.1, rv1.1.1a; respectively). The XSL/T tools will enable you to add topics of your own interest and then, in turn, perform contextual searches for terms or combinations of terms. It's up to the reader, and the discussions carried on the list serve, as to how far we all advance our field with this work and the sharing of both our frustrations and success.

In the case of XSL/T, the reader who is already familiar with this powerful document command set will know that I've barely scratched the surface. Still, what is possible with a simple set of basic commands is quite powerful. In addition, I ask that the technically savvy reader accept the more intuitive descriptions for the commands provided in some places rather than the actual verbatim from the specification in order to ease the learning curve. Additionally, a forthcoming book from Prentice Hall, in the series edited by the creator of Markup languages, Charles Goldfarb, is written for the non-programmer and includes some more complex tools for working with these kinds of texts (*The XSLT and XPath Handbook*, Gardner and Rendon, Dec. 2000).

It remains to each reader and her/his collection of electronic text resources relevant to their work as to what additional functions can be found. The examples below concentrate on the core functions. For instance, I will not

present the perfunctory and largely esoteric lines of code which come before and after the key operators I am explaining. These are somewhat obtuse to the uninitiated and are themselves largely unchanging from example to example. As a result, they could deflect the reader from understanding basic operations and can therefore be learned later as needed. Unfortunately, I can assure you of no macrocosmic wisdom or significance lying behind these code snippets beyond the brute reality that the scripts don't work without them.

Most of the introduction for XML should be reviewed at IJTS. A thumbnail sketch of XML is provided here. The following discussion assumes that the reader has a working knowledge, or at least familiarity with, the basics of markup technology. In other words, some rudimentary grasp of how HTML looks and works will make the following discussion infinitely more fruitful. In the absence of such prior knowledge, the astute reader will nonetheless find the basic underpinnings of XML technology remarkably accessible. An appendix regarding access to and installation of the related tools has been included. It is also assumed that, in order to actively deploy this technology, the reader has a working familiarity with the World Wide Web, and the ability to download a file and perform simple installations of software.

The paper begins with a short overview of XML. After that, a step by step introduction to the basics of XSL/T is provided (sample scripts are included with this article, see below). Next, more sophisticated XSL/T tools for Vedic research and a brief discussion of how XSL/T and *vidhāna* are provided. Finally, a detailed Appendix for Windows and Mac installations and running of scripts, as well as making your own edits, is provided.

I. Extensible Markup Language

It is fitting that the early inroads of X-nology to humanities research include Vedic studies. Scholars of Sanskrit were among the earliest pioneers in electronic text technology, or e-textnology. I'm referring of course to the landmark achievement of Lehman and Ananthanarayanan in 1971 with their digitization of the Rig Veda and Shatapatha Brāhmaṇa under a National Endowment for the Humanities grant. This is not surprising considering the

long-standing deference to Sanskrit linguistic theory throughout the Artificial Intelligence community.[3]

There are many forms of markup which grew out of the early efforts of many individuals in text research. The "parent" structure of markup languages is Standard General Markup Language, or SGML. SGML specifies the rules by which any markup language, including XML, can be written. The most well-known of SGML's offspring is HTML. HTML, for all the success of the World Wide Web, has shortcomings however. Take for instance the occasion of *italics* in any given sentence. An HTML page will show you italics and it's up to you to figure out why they are there (is it designating a foreign word, or title, or emphasis, for example). The basic technology behind this is remarkably simple, however.

Angle brackets, or less-than/greater-than symbols, are used to indicate an element of a document which can be described by a general name, such as one to designate an author: <author>Shakespeare</author>. These brackets and the information they contain are usually called tags.[4] They generally appear at the beginning and immediately following whatever it is that they are marking. The tags are usually only visible to the computer or processing software. If it

---

[3] There is a well-known amenity between Sanskrit and computer technology. Pāṇini has been a topic in the artificial intelligence (AI) community for some time, specifically with regard to his formal system of logic in the grammatical sutras. Also noted by Chomsky (1956, 1957), it is a frequent allusion by computer scientists (Russell and Norvig, 1995:15; Ingerman, 1967:137), and is identified by Chomsky's type 2 grammar (Knuth, 1964:736) as a syntactic origin for propositional logic (Russell and Norvig, 1995:256, 685, cf. Briggs, 1985). My special thanks to Patrick Durusau, the digital wizard of Scholar's Press at Emory University, for one of the Backus-Naur Form (BNF) references.

[4] The definitive discussion of the often-misused nomenclature for markup is found Goldfarb and Prescod's *XML Handbook*, 2nd ed. Specifically, "tag" is an acceptable reference for these markup items, and element refers to what is marked up. "Tag name" is technically not acceptable, though "element name" is—e.g., you can refer to "author" as the element name of the <author> tag; while <author>Shakespeare</author> in its entirety is an element. As you can see, this is indeed confusing for the beginner, so I refer to <author> as a tag, and the word "author" therein as the "tag, or element, name" in this paper (cf. Goldfarb and Prescod, 2000:71-72). Incidentally, I highly recommend this book, and Elliot Rusty Harold's *XML Bible* as the best resources for the beginner and even experienced markup worker. The first edition of the *XML Handbook* is not a safe bet as the technology has developed significantly beyond that stage.

sounds simple, it really is. In spite of the commercially-hyped changes in technology, this basic format has remained unchanged for over 15 years.

XML begins with this basic principle of markup tags, removes some of the more esoteric parts of SGML, and invokes a ruthlessly strict syntax. What is achieved through this "strict" set of rules for tags is, among other things, a wider range of software and a simpler set of applications for writing, working with, and storing XML documents.

The other advantage of XML lies in the word "extensible." This indicates that there is flexibility in the kinds of things that XML can identify. For Indologists, XML opens a panacea of possibilities by contrast. Tags can be matched with a tradition's terminology, or text-specific tags can be chosen to enable more precise marking, study, commentary, and reuse of an ongoing—even lifelong—text research project. Add to this the support for non-Roman-script languages, and you have an ideal tool for building a research resource. Of course, XML is also designed for storing and accessing all manner of media formats so that recordings and tapes from field study can be integrated into the research resource.

1. Terminology

Before going further, a couple of basic terminological clarifications will help. "Elements" are parts of the text marked with tags (cf.. note 4 above). Technically, in the following example "<author>Shakespeare</author>," the entire string of characters between the quote marks is an element. The word "author" is the element name. It is common to refer to the pair of <author></author> components as a "tag." Thus, you could say that "author" is the "tag name," but markup purists won't accept this when you hobnob with them.

Most such tags have a start tag, and an end tag (marked by the "/"), and they "wrap around"—or come before and after—the text that they are tagging. The end tag is also called the "closing tag" and when it is present, the complete wrapped element is often said to be "closed" as of the occurrence of the end tag. It is also possible to have a place-marker tag which doesn't contain anything, such as one to mark a note you are making on a text like <mynote type="here is where the original MSS ends" />. Note that instead of open/close tags, there is a single tag which "self-closes" with the "/" at the end of it. This is called an "empty tag."

It is possible to add more information to a tag, while keeping the same element name. Consider the following: "<author reference="last name">Shakespeare</author>" as opposed to "<author reference="full name">William Shakespeare</author>." The addition of "reference="last name"" is called an "attribute." The word "reference," then would be the attribute name and "last name" or "full name" would be the attribute value (sometimes this is called a "variable," but not as frequently). You can also add as many attributes to an element as required by your research task, or required level of detail (cf. Van Nooten and Holland, 1994:1, for the information below):

```
<mantra id="rv1.1.1a" meter="gaayatrii" family="vaizvaamitra" deity="agni">
agni'm iiLe puro'hitaM
</mantra>
```

These basic structures underlie all markup languages—for the most part—currently in use.

If you follow the basic XML syntax rules outlined below, you can begin adding your own notes to whatever primary sources you have in electronic form, or notes you've entered on disk over the years. Then, using the XSL/T techniques below and a little practice, you can begin spending less time **finding** what you've researched and more time reflecting on, adding to, and publishing your work.

2. Syntax Rules for XML

The syntax for XML is more strict than the usual HTML and other markup. These strict rules are based on a notion of what is called "well-formedness." Well-formedness is a new concept introduced by XML. Essentially this means that all tags must either have closing tags or be written in a special form (as described below), and that all the elements must nest one within the other.

It's sort of like having good manners: saying "please" before receiving something, and "thank you" afterward—or, use starting tags before the content you're marking, and closing tags afterward, for instance. Sets of tags should nest one inside the other, and values for attributes must have matching quote marks. For a clear explanation of these and several additional concepts, see the equivalent section on XML Syntax in the IJTS article. There is also a short

review I highly recommend at the World Wide Web Consortium's (W3C) web site.[5]

3. Additional XML Technologies

XML comes with a raft of related standards, some already completed, others close behind. These standards add to the functionality and truly exciting potential of X-nology, often without requiring expensive software. A great deal of electronic commerce and the wireless information devices such as web-smart cell phones use X-nology. To be sure, there are deluxe XML gadgets out there with some pretty deluxe prices, but there are just as many functional ones which are free or less than $100.

Two XML "technologies" that are recent standards (as of November, 1999)—Extensible Stylesheet Language for Transformations (XSL/T) and XPath (a system for describing where a piece of information is based on the tags and other content in an XML document)—are especially powerful and exciting. I will introduce some of the core functions of XSL/T below. To begin to do more, XPath will enable more precise selection of tags and ways of working with your files. These specifications enable sophisticated manipulation and extraction of data you've added to an XML file with a fairly human-friendly set of commands. Let's put it this way: if you can manage to research and read Sanskrit, Vedic, or Prakrit, these languages are destined to quickly be your favorite research tool.[6]

---

[5] The XML-conformant version of HTML is a good way to start, with Tidy (mentioned in note 9), getting yourself comfortable with XML, see: http://www.w3.org/TR/xhtml1/#diffs8.
There are additional rules and details, but most of these you will not or need not encounter. To introduce them here would detract from the fundamental simplicity of the basic essentials required for the majority of your work with XML. To learn more, follow the links from http://www.xml.com, or http://vedavid.org/xml/.

[6] If you want to begin tagging, you can use a plain text editor such as the Microsoft Wordpad in your Windows/Programs/Accessories menu. There are plenty of tools for specific work with XML which work much better, however. If you are adding your own notes and comments, marking passages, or organizing your own files of data, then you will want to use an XML editor such as those at the links provided in the Appendix. For Windows, WordPerfect 2000 is the best and easiest to use. Of all these tools, it is the most "costly"—but still quite reasonable at less than $100 educational price. On a Mac, at the same price is the Pro version of Media Design in-Progress's Emilé, which also has a free "Lite" edition to get you started (http://www.in-progress.com). For free, on Windows there is a wide variety. I

It is important to underscore that all of these technologies are standards. They are not "owned" by a corporation (though, predictably, Microsoft is trying to by implementing XML their own unique way). Therefore, to begin working with XML is not the same thing as upgrading in response to the commercial forces of the digital market place, only to be out of date before you've even received the bills.

But XML tagging is only part of the story. It is the indispensable core foundation, but the addition of a few XSL/T commands is where the magic happens. The more detail—with attributes or additional elements—that you add, the more you can do with your text.

Extensible Stylesheet Language for Transformations and XPath:

Manipulation and Matching

Now that you have a basic understanding of XML's fundamental rules, you are ready to begin to work with XSL Transformations. This is where the power and versatility of XML really becomes apparent. In fact, XSL/T's syntax is the same as that of XML. The examples begin simple to create conceptual understanding. It's important to note this because the astute reader will notice that some of the first examples could be done with find-and-replace in a word processor.

Fundamentally, consider XSL/T as you would *vidhāna* (*vi* + *-dhā* — to distribute, to put variously, sort), both literally and figuratively. *Vidhāna* is to *sūtra* as XSL/T is to XML document, specifically, that is—and for the references in this article—insofar as injunctions for mantras to be used in a ritual (for further discussion of *vidhāna*, see below).

Among other things, XSL/T is written according to XML rules. This makes certain parts of the otherwise difficult tedium in the syntax found in other mainstream programming languages less ambiguous and idiomatic. An XSL/T command has a beginning and end in a symmetric order just like XML's

---

prefer the simple approach which utilizes keystrokes rather than some fussy graphic interface, in a product called XED (find it and other tools at http://www.xmlsoftware.com). On the Mac, the Lite version of Emilé is the best free tool. The advantage of working with an XML-automating software such as these is that they make it easy to avoid typo's and errors which violate the XML rules.

matching opening and closing tags. Otherwise, it uses empty tags such as those we saw above.

XSL/T works based on a simple notion of matching. In other words, you tell it go and find such-and-such a tag, and once it does, do something to it, or do something to what it contains. A simple example assumes you to change my tagging of the Rig Veda in XML which uses mantra tags, and you want to change them to "paada." (this is in the "change_elem.xsl" file):[7]

```
<xsl:template match="mantra">
  <paada>
     <xsl:apply-templates />
  </paada>
</xsl:template>
```

In short, this finds—or matches—anything that has the tag or element name of "mantra" and changes it to "paada." In the example above, the basic components of XSL/T are all present. A simple command, in this case, "xsl:template"[8] is to make this model or template happen according to the pattern specified by the attribute value given for "match," in this case, all elements named "mantra" are to be templated.

---

[7] The Windows version operates differently for the Mac version, the Mac has automated applets on which you double-click to perform the transformation, and you edit the XSL/T file to change what you want to accomplish with the script. In the Mac version, you have this file change_elem.xsl, but it is activated by double-clicking the "jrg's_change_element" JAVA applet. The Mac applet will always have a name very similar to the accompanying XSL/T file (both of which must be in the same Jbindery folder per the instructions in the Appendix). In addition, for Windows and Mac, you have the option to convert to HTML for viewing in a browser. Not all scripts for the Mac have a matching "view" applet as those which only change element names or attributes will not "look" different. Those you should open in your Windows WordPad (see note 6 above), or Mac BBEdit.

[8] "Namespaces" is the specification at the Consortium which distinguishes tags from one DTD or tag set from another. Since you can have regular tags or elements—such as "mantra" in this example—in your XSL/T file along with the XSL/T command elements—such as "template"—it is important to make clear the difference so the XSL/T processor knows what to do. You might have a document where you use your own "template" tag. So, it is necessary to distinguish it from the XSL/T command of the same name. Namespaces are also used to distinguish tags from different DTD's in XML documents as well. See http://www.w3.org/TR/1999/REC-xml-names-19990114/.

Here the XML syntax can help us understand what's happening. Everything between <xsl:template match="mantra"> and </xsl:template> is what is supposed to happen, or is to be done, when the match of "mantra" is found. It's easy to tell the difference between the XSL/T command tags and the actual XML tags for the RV that we're working on, because all XSL/T command tags begin with "xsl:".

Simple enough, but what about that little <xsl:apply-templates /> esoterica? I confess, that this one often confounds me. The <xsl:apply-templates /> basically says "keep everything that was originally wrapped with <mantra> (technically called the 'children' of mantra), and pipe it on through to the resulting file, but wrap it in <paada>."

Suppose you decided that you wanted to identify the meter of each mantra as well. To add the attribute for meter is fairly simple. This particular process is very handy because you don't have to change element/tag names in order to add additional information for your research. To begin, we will only be adding the attribute name "meter" and a generic value of "vedic" as an example of the XSL/T attribute function (this is from the general_add_attribute.xsl file in the sample set, and it is assumed you are working with the original XML RV, not the one with the mantra's changed to pāda's):

```
<xsl:template match="mantra">
   <mantra>
      <xsl:attribute name="meter">
         <xsl:text>vedic</xsl:text>
      </xsl:attribute>
      <xsl:apply-templates />
   </mantra>
</xsl:template>
```

The XML syntax will again help us interpret what's going on in this script. The <xsl:template> tags specify what to do when a match of the "mantra" tag is found. In this case, the <xsl:attribute> command is to be applied when the match of "mantra" is found. When this happens, an attribute named "meter" is to be inserted into each <mantra> tag.

The value for this newly-created "meter" attribute has been set as "vedic" to keep this introductory example uncluttered. Note how the <xsl:attribute>

tags are wrapped around the identification of "vedic" as the text value for the meter attribute according to basic XML rules in order to effect this creation.[9]

Now, assume you know that all the mantra's of a particular hymn were *gāyatrī*. Rather than just insert a generic meter tag for every mantra with a meaningless "vedic" value, you want to identify a particular hymn's meter as *gāyatrī*.

So, we want to find the specific <hymn>, which is *gāyatrī*, one of which is identified as 3.62. In XSL/T, "attribute" is abbreviated by the "@" sign. Following the "@" comes the attribute's name. If you want to add something to this particular hymn's attributes, such as an attribute and value for meter, you can easily do this by specifying the "id" attribute's value (this is the deatil_attribute_create.xsl file, and you can select different hymns):

```
<xsl:template match="hymn[@id='rv3.62']">
   <hymn>
      <xsl:attribute name="meter">
         <xsl:text> gaayatrii </xsl:text>
      </xsl:attribute>
      <xsl:apply-templates select="*|@*" />
   </hymn>
</xsl:template>
```

This script is very much the same as what we did above, except that we are saying "match that particular hymn which has an attribute called 'id,' where the value of that id attribute is exactly 3.62." The reference to "hymn" specifies that element name, the [] indicate that, within any given <hymn> tag, find whatever it is that is inside the []'s. In this case, it says, "inside some hymn that has attribute/@ by the name of 'id' and value equal to 'rv3.62,' consider this a match and do whatever follows." You've just chosen RV 3.62 to act upon.

What comes next again follows from XML rules. As above, we want to keep our hymn tags, so we wrap them around the particular command we're doing, which is to use <xsl:attribute> as before to add an attribute named meter. This time, because we've chosen specifically RV 3.62, only that hymn will have an

[9] The <xsl:text> tags just state that this is the only text to put in, and not to add any extra spaces before or after "vedic." This is import as, in this example, that "vedic" line is indented and you don't want to stick all that indent space into every attribute or your resulting RV file gets very big and messy.

attribute with the value "gaayatrii." In fact, of course, only a few parts of RV 3.62 are *gāyatrī*, and it is possible to change this selectively for one or several hymns with XSL/T.

This is an easy way to look at how an attribute value—in this case the hymn's specific id, "rv3.62,"—can be used to identify this specific passage for a transformation with XSL/T. Of course, however, you know that in RV 3.62, verses 1-3 are *triṣṭubh* meter, while 4-18 are *gāyatrī*. I need to be able to "choose" specific mantra's for "when" to identify them as one meter or, "otherwise" another (namely, of course, for this hymn the balance are *gāyatrī*). Consider, then, the following adaptation of the rule above (this is the when_test.xsl file set):

```
<xsl:template match="hymn[@id='rv3.62']/verse">
   <xsl:choose>
      <xsl:when test="@id='rv3.62.1' or @id='rv3.62.2' or @id='rv3.62.3'">
         <verse>
            <xsl:attribute name="meter">
               <xsl:text>tristubh</xsl:text>
            </xsl:attribute>
            <xsl:apply-templates select="*|@*" />
         </verse>
      </xsl:when>
      <xsl:otherwise>
         <verse>
            <xsl:attribute name="meter">
               <xsl:text>gaayatrii</xsl:text>
            </xsl:attribute>
            <xsl:apply-templates select="*|@*" />
         </verse>
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>
```

The <xsl:template>, <xsl:attribute>, <xsl:apply-templates>, and repetition of the chosen <verse> element are all the same as what we've seen above. All we're doing is setting up a choice (xsl:when/xsl:otherwise) for what to do whenever the template "match" for verse containing the attribute "id" which has a value of rv3.62.1, "or" 3.62.2, "or" 3.63.3.

We can accomplish the function of the match, conditional test, and change using XML syntax, with an <xsl:choose> tag that contains an <xsl:when> test and an <xsl:otherwise> for those occasions which don't "pass" the <xsl:when> test. You can use as many "when's" as you want (just remember to repeat the complete XML syntax of opening and closing), but here we're able to combine them because we know that RV 3.62.1-3 are one meter, and everything else is another. In this xsl:when test, the criteria is 3.62.1-3, each verse distinguished with "or." If the "when" finds a match, then the element verse receives an xsl:attribute action wherein the attribute name of meter is set to the xsl:text value of "tristubh."

These examples have showed you how to add and change markup to an XML text. Of course, for scholarship, it is even more important to extract data. Remember that XSL/T is good for adding markup to a structured text like the RV which you already know well. It is even more powerful for extracting information based on that structure or based on tags you've added according to your research emphasis. One of the best ways to think of XSL/T is as a programming language for non-programmers. One of the greatest wizards of markup technology and lead developer and creator of related software as shareware, James Clark, notes "XSLT makes XML useful for non-programmers" (http://www.jclark.com/xml/xslt-talk.htm).

XSL Transformations and Vedic Research

For the following section, a set of texts and online tools are available for you to replicate the examples and begin making your own stylesheets by making small changes to the working scripts.[10] The first point about which the reader may have been wondering concerns the name of XSL/T itself: Extensible stylesheet Language for Transformations.

[10] To work with these files, it does require Windows 95 or later, and Internet Explorer 5 (free) for easiest installation. If you are not technologically feint of heart, this can also be installed on UNIX. Subsequent to the publication of this article, I will also have a Mac-native environment for XSL/T (note, you can still work with XML on a Mac until then) which uses the MacIntosh Java Software Development Kit (MRJ SDK 2.1) platform (free) instead. You can also install Windows on a Mac with VirtualPC or SoftWindows for around $200 educational price. See the appendix for more details.

The use of the term "transformation" is specifically ironic in the context of mantra and its role in effecting ritual transformations. I have mentioned *vidhāna* above as a word signifying distribution or variously apportioning. Its formal role of designating the mantras to be employed in a ritual (e.g., F. Smith—*vidhi*—1987:23-24, 27, etc.; Gonda, 1980:4, 213f.; Bhat, 1998; Staal, 1989:48f.; etc.) is an optimal analogue of XSL/T's functional role with XML. Naturally, this is a different sense of transformation that is commonly associated with the role of mantra (cf. Wheelock, 1989:101f.; Gonda, 1980:345; Beyer, 1978; Santidev, 1999 – Vol. 2:113).

Along with several discussions of *vidhi* (Smith, Gonda, Bhat, etc.), there is an interesting discussion of a mantra's actual "transformation," though quite differently, in both Staal and Alper's offerings in the Mantra collection. Staal posits the transformation chronologically from mantra's to more discursive language. The simple rules of ritual which govern the mantra originating with it become complex as language develops from it (1989:71-72). This is, of course, an inverse of the relationship suggested here for the purpose of understanding XSL/T. However, it is worth noting that the relative simplicity—regardless of the immediate clarity or absence of meaning—in an XSL/T script definitely becomes complex and possibly unnatural in the requisite explanations, not unlike what Staal suggests!

Before the allowing the temptation to draw these inferences and accompanying metaphors beyond the scope of this article, a return to the primary focus upon deploying the technology will illustrate the relevant connection. Accordingly, contrary to the common sense of "transformation" being effected *by* mantra, we will refer to XSL/T more closely in the sense of *vidhi* to designate the transformation performed on a text to extract mantra (or any other selected portion of a text) for ritual or study. This use of XSL/T can facilitate the suggestion by Alper, for instance, of "using modern methods to gather together a large number of mantras" which could be drawn from an "inventory of rites where mantras occur" (1989: 311-312).

To be sure, such an undertaking assumes the existence of no small collection of texts in electronic form (some of the most systematic of which are found at TITUS).[11] Such resources would have to also be in XML, of course, to enable

[11] For resources from the TITUS collection, see http://titus.uni-frankfurt.de/; for other resources, see http://fas-www.harvard.edu/~clopez/indolink.html, or http://vedavid.org/methtech.html#anchor109532.

not only the collection but the systematic cross-referencing and multiple avenues of study enabled with XSL/T tools. As a starting model, we can begin with the XML RV supplied for this article. Many of the mantras inferred by Alper have been employed throughout history in various forms for various ritual purposes. Works such as Bhat's edition of *Ṛgvidhāna* provide ample lists of specific mantras to be extracted from the RV according to ritual applications (1998). In fact, one could arguably "translate" the *Ṛgvidhāna* into XSL/T insofar as its identifications of various mantras and the style of their recitation go.[12]

We've already seen above with the selection of RV 3.62 for adding an attribute how easy it is to select a specific mantra from an XML edition based upon your knowledge of how it is structured (e.g., id values of "rv1.1.1," etc., per verse and so forth). Often, however, the mantra's are not only extracted, but also re-formed. In order to examine both operations, we will move beyond the simple listing of mantras to their reworking in the following examples.

Returning to Staal, consider his discussion of *indra juṣasva* and the Soma ritual (1983 – Vol 1:661f., also 1989). The two mantras in this example are RV 1.16.1 (accents removed per Staal's purposes and those of this example):

ā tvā vahantu harayo vṛṣaṇaṃ somapītaye |
indra tvā sūracakṣasaḥ ‖

and RV 1.84.10:

svādor itthā viṣūvato madhvaḥ pibanti gauryaḥ |
yā indreṇa sayāvarīr vṛṣṇā madanti sobhase vasviir anu svarājyam ‖

Here they are in XML form building on the files you are provided (use the "viharanam.xml" file for trying out this example):[13]

```
<verse meter="gaayatrii" id="rv1.16.1">
```

[12] And yes, there is even a VoiceML for XML marking of oral texts, so the prescribed murmuring in the *Ṛgvidhāna* could also be stipulated! Learn more at http://www.oasis-open.org/

[13] Your RV does not have the mantra-level identified with the full id such as here, e.g., you have the verse identified with "id="rv3.62.10"," but the individual mantra's only specify if it is sub-section a, or c as in the Van Nooten and Holland edition. An RV tagged to this detail is currently retained under separate copyright, but you can also create your own by working with the files here, reviewing the recommended XML books (especially *XML Bible*) in the Bibliography, and discussing the process on the Indology list serve recommended in the appendix.

```
    <mantra id="rv1.16.1a">
      aa tvaa vahantu harayo
    </mantra>
    <mantra id="rv1.16.1b">
      vRSaNaM somapiitaye
    </mantra>
    <mantra id="rv1.16.1c">
      indra tvaa suuracakSasaH
    </mantra>
  </verse>
```

and the second mantra:

```
  <verse meter="gaayatrii" id="rv1.84.10">
    <mantra id="rv1.84.10a">
      svaador itthaa viSuuvato
    </mantra>
    <mantra id="rv1.84.10b">
      madhvaH pibanti gauryaH
    </mantra>
    <mantra id="rv1.84.10c">
      yaa indreNa sayaavariir
    </mantra>
    <mantra id="rv1.84.10d">
      vRSNaa madanti sobhase
    </mantra>
    <mantra id="rv1.84.10e">
      vasviir anu svaraajyam
    </mantra>
  </verse>
```

Staal discusses the use of *viharaṇam* (intertwining or transposition) with these mantras as part of his longer argument about the derivation of language from the pre-linguistic residual of ritual as something represented—or preserved—in the Vedic mantras (1989:52ff.). In fact, the intertwining of these two mantras includes additional verses from each hymn, undergoing *viharaṇam* in turn, as well as similar procedures with several other hymns (1983 –Vol 1: 661f).

In this example, I'm addressing only the mechanics of the syntax as an applicable instance of extraction and reconstruction of mantra facilitated by

XSL/T. Working with the files provided, the matter of actually extracting the mantras and then reconstructing them can be done in one XSL/T script as the specification enjoins that the commands are to be enacted by the software in the order of the script's composition. The entire example serves not only as an analogy for XSL/T in the Vedic ritual tradition, but also as an excuse to provide you with a very powerful XSL/T script.[14]

I will divide this into two parts for clarity. The first involves the actual extraction of the mantras to be intertwined. This is a very simple operation, much like the one in which we applied a value of "gaayatrii" for the meter attribute of a hymn tag above, only we are "pulling out" the specific passage(s) chosen (this is the "select_hymn.xsl" file, only revised for the attribute id level—it's preset for 1.162, 1.164, so for practice you can edit it as below—and can be used with the larger RV file, or your own files once you tag them in XML):

```
<xsl:template match="hymn">
   <xsl:copy-of select="*[@id='rv1.16.1']" />
   <xsl:copy-of select="*[@id='rv1.84.10']" />
   <xsl:apply-templates select="." />
</xsl:template>
```

As you can see, if you need to pull a whole range of mantra's from a text, this can be repeated over and over. The script is run once, and you get a file with only the items you want to work with. Obviously, if you were only getting one mantra it would be as easy to retype it or cut and paste it as write all this code, but this short example is for didactic rather than pragmatic illustration. To display them, run the appropriate "view" file and open the resulting HTML file in your browser.

We're going to use XML syntax to perform the *viharaṇam* technique with XSL/T, but at a more complex level than you might expect (use the

[14] Many thanks for the debugging and discussion of the various scripts in this article should go to David Carlisle—especially for his patience in working through my efforts to explain a Vedic concept in XSL/T development terms—and Wendell Piez of the XSL discussion list (xsl-list@mulberrytech.com, "Re: Bug in XT? was [RE: Change Attribute Value: Search-n-replace is better?" and "String Match," December, 1999; at archive: http://www.mulberrytech.com/xsl/xsl-list).

"viharanam.xml" and the "viharanam_transform.xsl" files to try out this example):

```
<xsl:template match="sample">
   <xsl:for-each select="verse/mantra">
      <xsl:sort select="substring-after(@id,../@id)"/>
      <xsl:sort select="../@id" order="ascending"/>
      <xsl:copy-of select="."/>
   </xsl:for-each>
</xsl:template>
```

I included this script to show how sophisticated XSL/T can be—even in just a few lines—not to show how easy every function is. We're matching the main element or tag of the sample file (called "sample" in viharanam.xml), and asking that for each mantra within a verse, a sorting is done. The select is as above with the other occasions of "select," but we're using a string function which asks for whatever string comes after the attribute/@ "id", and the attribute of the other verse (selected by requesting its parent with ".."—see Appendix 2 on XPath in the IJTS article if you want to learn more—and then looking for the attribute and placing it in order after the first one). The second "sort" just says, count up (you can also supply "descending," which makes the rv1.84.10 mantras come before the rv1.16.1 mantras—try it with the sample files). Finally, the "copy-of" just says to output whatever the result of this sorting is.

But XML and XSL/T can obviously do much more, otherwise the world's biggest corporations and consulting firms wouldn't be embracing and supporting it wholesale. One of the most frequent needs is to find all portions of some level of a text which contains one thing or another. It is important to emphasize that, for the RV, it requires that you get the TITUS CD which has the excellent electronic edition of Lubotsky's painstaking per-word padapātha of the RV (it can be converted to XML). Many other electronic texts are based upon primary sources where the words are not elided as in the RV. It is up to the scholar to identify the resources s/he needs. In addition, I encourage those who wish to go further in this work to participate in a discussion of this on the Indology list serve for the benefit of other scholars and the discipline.

For another possibility, let's select all verses of the RV which also contain the word "I;dyo" (note, the ";" is the *udātta* accent mark. You may want to delete all of these from the sample RV). We're going to use the matches you've

learned above and another example of how intuitive the XSL/T language is, the "contains" function, and the "xsl:copy-of" operator (this script is "search.xsl" in your sample files):

```
<xsl:template match="//verse//mantra">
   <xsl:if test="contains(. , 'I;Dyo')">
      <xsl:copy-of select='ancestor::verse'/>
   </xsl:if>
</xsl:template>
```

Yes, it's really that simple. All you need to do is change the text value for the word you're seeking. This command says, match all tags called "mantra" (remember, XSL/T assumes you are asking for a tag or element name in the "=" statement unless you use "@" to say attribute) which contains the word "I;Dyo." You will notice the (., 'I;Dyo') syntax says a little more. The "." says "find itself and." Rather than go into all of why, I'll leave it that XSL/T is matching "mantra," but once it does, the "[]" statement says what it is of mantra that is sought. Of course, you want the mantra, not just the word, so you have to say "itself" (or ".") and the word sought (also the specification requires two arguments).

The xsl:template command is wrapped around a single "empty" (no separate closing tag to wrap around something) command called "xsl:copy-of." The thing to be copied, of course, is what is found in the match, the "itself" (or ".") located by the match command. Imagine, once you've added tags for meter, or chronology, or other topics you're studying, you can revise this script and seek matches on a word or words according to whether they are found in one context or another.

Conclusion

This exploration provides a beginning point for working with tools that are not only readily available but low in cost for both time to learn and finances. Most any Pentium or PowerMac will run them. On a Pentium, you'll need InternetExplorer 5 which is freely available. On a Mac, you'll need Apple's Java kit, MRJ SDK 2.1.[15] The other tools are explained briefly in the appendix.

[15] Special thanks goes to Chuck White for pioneering the Mac territory for XT use (http://www.javertising.com). His initial notes are available as part of an excellent

For your research, it is worthwhile to take these examples and practice with the RV in order to begin seeing what you will be able to do for your work. In closing, I would suggest that electronic text technology, or e-textnology, is only now beginning to "catch up" to the range of sophisticated tools which took form long ago with *smṛti* and *vedāṅga*. Most famous of these, of course, is the frequently-noted origins of propositional logic—upon which most basic machine language has been based—by the Artificial Intelligence community as a whole (cf. note 3). In one noteworthy exchange of the hallowed proceedings of the ACM, none other than Donald E. Knuth, creator of the TeX word processing system for the technical sciences and author of books on software engineering which have shaped the entire industry prompted a reference to Pāṇini when forwarding the now-standard name for the BNF or Backus Naur Form of annotation for propositional logic (Ingermann, 1967, cf. recent Indology discussion).[16]

Bibliography

Alper, Harvey P. "The Cosmos as Śiva's Language-Game: 'Mantra' According to Kṣemarāja's Śivasūtravimarśinī," in *Mantra*, Harvey P. Alper, ed. Albany: SUNY Press, 1989.

Arnold, E. Vernon. *Vedic Metre in its Historical Development*. Cambridge: Cambridge University Press, 1905.

Beyer, Stephan. *The Cult of Tārā: Magic and Ritual in Tibet*. Berkeley: University of California Press, 1973.

Bhat, M. S. Vedic Tantrism: A Study of the Ṛgvidhāna of Śaunaka with Text and Translation. Delhi: Motilal Banarsidass, 1998.

Bhattacharyya, Narrend. History of the Tantric Religion: A Historical, Ritualistic, and Philosophical Study. Delhi: Motilal Banarsidass, 1982.

Q & A source by Dave Pawson at http://freespace.virgin.net/b.pawson/xsl/sect4.html#N6860.

[16] See the Indology archives (and other e-textnology for Indology resources) via http://www.ucl.ac.uk/~ucgadkw/indology.html, performing a "Subject" search on "Pāṇini vs John Backus."

Briggs, R. “Knowledge representation in Sanskrit and artificial intelligence.” *AI Magazine,* 6(1):32-39.

Chomsky, Noam. “Three models for the description of language.” *IRE Transactions on Information Theory,* 2(3):113-124.

_____. *Syntactic Structures*. Mouton: The Hague and Paris, 1957.

Gardner, John Robert. *Vedavid Indological Research Site*. 1996-1998. http://vedavid.org/

_____. *The Developing Terminology for the Self in Vedic Indi*a. Dissertation. 1998, University of Iowa. Available online for free, http://vedavid.org/diss/.

Goldfarb, Charles and Paul Prescod. *The XML Handbook,* Second Edition. Upper Saddle River: Prentice Hall, Inc., 2000.

Gonda, J. *Vedic Ritual: The Non-Solemn Rites.* Leiden-Köln: E. J. Brill, 1980.

Harold, Elliot Rusty. *The XML Bible*. Online sections, http://metalab.unc.edu/xml/books/bible/.

Ingerman, Peter Zilahy. “‘Pāṇini-Backus Form’ Suggested.” *Communications of the ACM,* 10(3), March, 1967:137.

Lanman, Charles R. “A Statistical account on Noun-Inflection in the Veda,” *Journal of the American Oriental Society*, 10, 1880: 581.

Knuth, Donald E. “Backus Normal Form vs. Backus Naur Form.” *Communications of the ACM,* 7(2), December, 1964:735-736.

Narten, J. “Das altindische Verb in der Sprachwissenschaft,” *Sprache* 14, 1968: 114-115.

Oldenberg, Hermann. *Die Hymnen des Rigveda: Metrische und textgeschichtliche Prolegomena*. Berlin: Verlag von Wilhelm Hertz, 1888.

Russell, Stuart and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Upper Saddle River: Prentice-Hall, Inc., 1995.

Santideva, Sadhu, ed. *The Encyclopedia of Tantra*, Vol. II. New Delhi: Cosmo Publications, 1999.

Smith, Frederick M. *The Vedic Sacrifice in Transition: A Translation and Study of the Trikāṇḍamaṇḍana of Bhāskara Miśra*. Poona: Bhandarkar Oriental Research Institute, 1987.

Staal, Frits. *Agni: The Vedic Ritual of the Fire Altar*, Vol. I. Berkeley: Asian Humanities Press, 1983.

_____. "Vedic Mantras," in *Mantra*, Harvey P. Alper, ed. Albany: SUNY Press, 1989.

Witzel, Michael. "Ṛgvedic history: poets, chieftains and polities." In *The Indo-Aryans of Ancient South Asia: Language, Material Culture and Ethnicity*, edited by George Erdosy, Berlin: Walter de Gruyter, 1995.

Appendix: Tools and Resources for Working with XML and XSL/T on Windows and Mac

1. To Use the Example Files on Windows:

The "engine" that makes XSL/T files do what they do, which is free, is XT by James Clark. Clark offers it for free, working on Unix and Windows.

The best way to do this is to have a Pentium II or better and 20mb of hard disk space. The easiest way to run XSL/T files is to first get a copy of Internet Explorer 5 for Windows (to see which version you have, start up IE, go under "Help" to "About . .. " and it will tell you). You can do this by selecting IE5 and your operating system at http://www.microsoft.com/downloads/.

After you've installed it (follow all the pre-set switches so that the install happens transparently without affecting your system), then get the version of James Clark's XT for Windows 95 or 98 (often referred to as "Windows32" which specifies the new versions, 95 and 98, which only run on 32-bit processors), called xt.exe from http://www.jclark.com/xml/xt.html, or direct anonymous FTP from ftp://ftp.jclark.com/pub/xml/xt-win32.zip. Once it's downloaded (be sure to remember where you told your browser to put it), open that folder and double-click on the xt-win32.zip file. The WinZip application already on your system should open up, and follow the auto prompts to unzip

XT, setting a new directory for RV work. If WinZip doesn't open, you can get your own free by searching for "WinZip" at http://www.zdnet.com/downloads/specials/free.html. You may be able to short-cut the whole process by going directly to http://hotfiles.zdnet.com/cgi-bin/texis/swlib/hotfiles/info.html?fcode=000015.

When you unzip XT, tell WinZip to put it in a directory you can remember, such as "c:\RVwork\." Then all your files for this technology can be kept in one place. Get the ***link***sample files which are also zipped, and put them here, and unzip them with WinZip too (they are also at http://vedavid.org/xml/docs/).

Restart Windows, make sure that you have Internet Explorer 5 (IE5), and you should be ready. To run an example, you have to work in MSDOS. Go to Start, Programs, MSDOS Prompt. You should have a window with a simple DOS command-line prompt. Type:

```
cd\RVwork
```

And hit the "Enter" key (assuming you have an RVwork folder on your c:\ drive according to the instructions above). You're now in the directory for working with XSL/T and XT.

To see what all is there, type

```
dir
```

and hit "Enter." Choose the XSL/T script you want to use (remember, all files must be in your RVwork folder where xt.exe is), and run xt by typing as follows:

```
xt rv_ejvs.xml add_attr.xsl test.xml
```

Hit Enter and the process happens. You can check the results by opening the result file, test.xml, in a simple word editor like WordPad (Start, Programs, Accessories), choosing Open File, and in the Open File window, specifying at the bottom center pull-down "Files of Type" menu "All Documents," and then picking test.xml. This is also a good program to use for making your own changes to the scripts to begin learning more. Just always save as text only, no extra formatting.

In the command above, "xt" invokes the XSL/T processor called xt.exe. Then "rv_ejvs.xml" tells it to start with the RV supplied with this article for

performing the commands determined by "add_attr.xsl," the XSL/T instruction script described in this paper. You can give any name for the output file, called here "text.xml."

2. To Run XSL/T Scripts with XT on a Mac

MacIntosh is not a command line system, so for most of you, this will be probably one of the more complicated things you'll install, but follow these instructions and all should be well (comforting, yes?). Be certain that once set up, you won't have to fuss so much, but also know that this will be the toughest thing you've done with your mac in all probability (but it's worth it). In the future, better tools are coming from http://www.in-progress.com.

In addition, you should have at least OS 8.0, and no less than 64megs of of RAM (but with large files like the RV, this will run slow, so 128megs recommended and don't even both unless it's a PowerMac- G3+ preferred). Go to MacIntosh downloads and get Aladdin Expander from the download at http://hotfiles.zdnet.com/cgi-bin/texis/swlib/hotfiles/search.html?b=mac if you do not already have it (when you follow this link you'll see a message at the top requiring you to submit a query but just below you'll see a little window with a search button next to it, type "Aladdin Expander" there). First, you need to download the EJVS sample files Vedavid at http://vedavid.org/xml/docs, and get the Java for Mac that has JBindery.

You will need two MacIntosh Java software packages, both free. You may already have one of these, but you will need both of them. First get MRJ 2.2.2 Installer at http://developer.apple.com/java/text/download.html#software. Download and unstuff it (it has a file called .xmi, and when you double click it, a file is created by the same name on your desktop that looks something like a disk). Double click the little disk icon just created, which is the installer, and choose custom install (where it says "Easy Install" in the upper left) and make sure to click all three boxes. Then click install (it will probably put it on your hard drive, which is best).

Next, get MRJ SDK 2.2 at http://developer.apple.com/java/text/download.html#sdk. Once you have MRJ SDK 2.2 and have unstuffed it (it has a file called .xmi, and when you double click it, a file is created by the same name on your desktop that looks something like a disk), double-click the "Installer" icon (the little disk icon) and choose custom install (upper left pull-down menu in the install window where it says "Easy Install"), checking both options.

Now you need the files for XT, from James Clark's site. **Be sure to choose the Java versions, not the Windows.** You will need XT and XP. Get them from http://www.jclark.com/xml/xt.html, or direct anonymous FTP from ftp://ftp.jclark.com/pub/xml/xp.zip and ftp://ftp.jclark.com/pub/xml/xt.zip. Unzip these (if you got Aladdin Expander from above, just double-click on the .zip files xt and xp). On your hard drive, you will have an MRJ folder, inside that is a folder called Tools, and one called Application Builders. It is inside the Application Builders where you will find the JBindery folder. You will put all these files in one folder called "xt" inside the JBindery folder of the MRJ SDK 2.1 folder on your hard drive. Do the same with XP (except put its files in the JBindery folder in their own folder called "xp").

Now UnStuff the EJVS files (double-click on it and it should launch Stuffit, if not, get a copy here http://www.zdnet.com/mac/download.html by searching for "Stuffit Expander"). Inside the resulting EJVS_supplementary_files folder, open the "mac" folder and put all its contents *inside* the JBindery folder **you must do this for the scripts to work**. NOTE: JBindery is the core of what you'll use, so be sure you KNOW where you put it, otherwise your efforts will be wasted. All the other MRJ materials are support files, what you will use is MRJ, and you must work in this directory b/c the command line for Java on a Mac, currently, is very difficult to manage otherwise. Do not just put the whole "mac" folder itself in, but take its contents out of it and move its contents into JBindery (you'll not use the "mac" folder again, which should now be empty), in addition to doing the same with the rv_ejvs.xml file, putting it inside the main EJVS folder. When you look at the contents of Jbindery, you should now have a bunch of .xsl files, some little jrgardner applet files, and an rv_ejvs.xml file along with the xt and xp folders.

Now you're just about ready to do a transformation, but you have to "tell" your mac where everything is. To run a Mac XSL file, double-click on the JBindery application in the JBindery folder and you will see a little window on your screen with some icons on the left (such as below, only without the filenames added in yet), and you will select the second one which says class path. First, choose a file to begin working with by selecting the File menu and Open, and browse to all your files in JBindery (which will ALWAYS be on your Hard Drive, in the MRJ Folder, in Tools, in Application Builder). You will need to do the following steps for each file that begins with "jrg's" (these are applets which, once edited, can be run by themselves by double-clicking on them INSIDE the JBindary folder). Click on the classpath icon and you will

see a window like the one below (but without the XP and XT files inserted). First, click on the "Add .zip File" button, and use the Mac finder window to click and select each .jar file after clicking on the "Add .zip Folder" button. You will need to get the xt.jar file, the sax.jar, and the xp.jar file into the classpath window by browsing to them and selecting as you did with the XT and XP folders. You will find sax.jar and xt.jar in the XT folder, and xp.jar in the XP folder. All files will ALWAYS—and must always—BE IN THE JBINDARY FOLDER.

When you're done (and ready to run an applet), your window should look as seen in the image below. You first want to click and save it (**BE SURE** that when you save, you save to the JBindary folder, where ALL files must always be) so that you don't have to re-enter these paths each time (just edit the XSLT files as described below once you see how the examples work). When you click on "Save Settings" you get a Mac finder window with the file name there—either keep this name by clicking save, or give it another name. You will need to do this for each applet converter before you run it as everyone's Mac has different names for hard drives and directory. But you only need to do this once for each. Then any changes, described further below, are only done to the XSLT scripts, these applets are just the little triggers that launch them. Now try it out! You can run this applet at this point from JBindary by clicking Run. Afterward, you can run it by double-clicking the "jrg's" applets from inside the JBindary folder. Note that it will take a few moments when you run it because the file is large and your system may not have the preferred 128 meg.s of RAM (64megs minimum). It produces a file in the JBindary folder called rv_att_detail.xml (if you run jrg's_attribute_detail). At any rate, the new file you have made will have a name DIFFERENT from the source file rv_ejvs.xml.

Doing this with each file before you begin, and saving it again, means you will always be ready to go when you are moving from one kind of RV transformation to another.

Use BBEdit Lite to open the file (get a copy at http://www.zdnet.com/mac/download.html by searching on BBEdit). When you choose Open from the File menu, be sure to set the "File Types" to "Any File." Then look for a .xml file with a name like that of the applet you ran, like—if you ran search, look for "rv_searched.xml" in JBindery (which will ALWAYS be on your Hard Drive, in the MRJ Folder, in Tools, in Application

Builder). After a little practice you'll learn how to rename and make new files by following these steps. More XSLT files will be available at Vedavid depending on what scholars request and discuss on Indology.

For viewing the results of the search and select_hymn actions in a browser, after you've run the "jrg" applet, then run the matching "view" applet, and open the HTML file which results in a browser. This is described in more detail in the following section.

If you want to make your own applets (see below) after you have more experience with XSL/T, and change the input and output file names, just run "JBindery" itself, choose "Open" under file, and select any "jrg" applet, and change the file names in the "Command" window such as seen here:

Then "Save as Application" in the JBindery folder with whatever name you wish, leaving the other selections as I have preset them. When you run it, if you don't see a new file with a .xml name appear in the directory, check the errors in "test.out" and report them to the Indology list, or to me at john.robert.gardner@sun.com. The list is better so others can benefit as they may have the same problems. To see your output as HTML in a browser, read below in #4 on display/view of output.

For Mac users, I also recommend a copy of VirtualPC so you can run Windows programs (yes, for real) on your PowerMac. In the long run, you'll benefit in multiple ways from having Windows on your Mac. And, the big secret is, Windows runs more stable on a Mac!

3. To Edit XSL Files

It is important to experiment to learn more. Change small things at first. Also, if you are technically inclined, bring your questions to the XSL list. To subscribe, go to Indology (if you haven't already), review the rules and how to join at: http://www.ucl.ac.uk/~ucgadkw/indology.html.

On a Mac, I recommend using BBEdit Lite which is free (http://www.zdnet.com/mac/download.html, and perform a search on "BBEdit", and select 4.6), and it assures that your files will be saved correctly. On Windows, open the file in Microsoft WordPad (go to Start, Programs, Accessories, WordPad). To open an XSL file, you'll need to tell the "Files of

Type" menu, which is at the bottom center of the window that pops up when you choose "Open" from the File menu, to show "All Files." You will want to be sure and only change those variables, indicated in the article above, in the parts of the script marked off between the rows of asterisks.

4. To Display or View Selected XML Output Files in a Browser

Sometimes you may not need to see just code, but want to view the files for easier reading and study, like in a browser.

On Windows, run XT with whatever filename of the XML you want to view in a browser, and the convert_to_browser_view.xsl file like this:

```
C:\xt mytextoutput.xml convert_to_browser_view.xsl viewit.html
```

Note –be sure and put ".html" at the end of the output file, or your browser may not read it in formatted form. You can open it in a browser by choosing "Open file" and/or Browse from your browser's file menu (browsers can open files on your hard drive just like files on the net).

For MacIntosh, just run the "view" applet that matches the "jrg" file name whose output you want to see, such as jrg's_select_hymn would have its output converted to a browser viewable html file with "view_selected-hymn." The resulting HTML file will be in your JBindery directory, and you can open it from the file menu of your browser by choosing "open file" or "open page."

5. To Add More XML Tags

If you are adding your own notes and comments, marking passages, or organizing your own files of data, then you will want to use an XML editor such as those at the links provided below. For free, on Windows there is a wide variety. I prefer the simple approach which concentrates on keystrokes rather than some fussy graphic interface, in a product called XED. On the Mac, the Lite version of Emilé is the best. You can get XED, or a range of other possible XML editors free to try and use, at http://www.xmlsoftware.com. You can get Emilé for the Mac at http://www.in-progress.com. They include instructions for their use.

For a few dollars extra, WordPerfect 9 in WordPerfect Office 2000 is great for XML tagging, and even doing your own work with DTD's. WordPerfect 2000 is the best "for cost" tool and easiest to use. Of all these tools, it is the

most “costly”—but still quite reasonable at less than $100 educational price. On a Mac, at a lesser price is the Pro version of Media Design in-Progress’s Emilé.