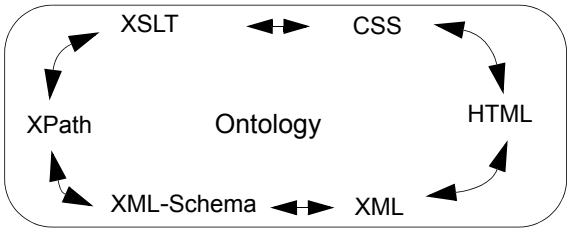
XML-Rings & Ontologies:

A Manual of Six XML Compatible Technologies



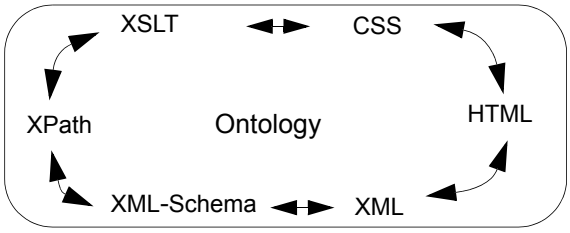
Rob Rucker

Aug. 2009

Mesa, Arizona

XML-Rings & Ontologies (*draft 2011-02*)

An XML-Ring



Why This Manual?

My reason for writing this manual is to show you how to understand, learn, and then use a core set of compatible XML (eXtensible Markup Language) applications. I will use these technologies together to develop some information domain(s) into a set of HTML (Hypertext Markup Language) web pages. The resultant pages will be styled by means of another technology, CSS (Cascading Style Sheets). To do a professional job in creating this set of HTML pages and its styling though, we will need to use four other technologies during page development. If you learn the total set of ring technologies presented in this manual, you will not only be able to produce the HTML pages shown here, but will also be well positioned to move to the next level of development, web applications. This will allow you to construct *web applications for your own purposes, within your own domain(s).* The web applications I am talking about are the bases of *Web Services* and will also become core elements of the modern SOA (*Service Oriented Architecture)* architectures. This is where I am ultimately guiding you, but for now we need to learn the infrastructure, the technologies illustrated by this manual. I must mention here that that is another emergent technology, **XQuery**, that is also critical to learn but, that's in another tutorial (see XPath&XQuery in this series)! The use of *JavaScript* is a valuable scripting tool for manipulating an HTML document, but its use will not be discussed in this tutorial.

I will call the set of XML technologies presented in this manual the *XML-Ring* (as in the diagram above) and the area of interest they are to be applied to as an *Ontology*. Briefly, an Ontolgy is simply a general shorthand word that covers the vocabulary of your area of interest, the meaning of these terms, their *inter* and *intra* relations, and any structure that they embody. The ontology provides the *context* and the reasons for development in the first place. The combination of an ontology and the 'ring' of XML technologies needed to represent, manipulate, transmit, store, and present it, are the focus of this manual and the reason I have titled it *XML-Rings & Ontologies*.

I have felt the need for this kind of manual for several years now, in order to show/explain to my colleagues, students, (and myself!) what they need to know in order to go from a domain of ideas and information, to web pages and web applications. In particular, what technologies and what procedures might be helpful in the web development cycle? In my role as a university instructor in Java, Object Oriented Analysis and Design (Software Engineering), XML, and e-Business, I have had a chance to try out different ways to introduce and then guide technology acquisition in these areas. This manual emphasizes the XML component of my experience, together with a way of learning XML applications that I have found effective.

Once you decide on the area (ontology) you want to work in, then you can use the technologies in this manual to create your own stylish web pages for personal pleasure, or go deeper and create more sophisticated web applications suitable for e-Business/e-Organizational deployment. As mentioned above, this manual provides the background for learning more advanced XML applica-

tions such as SOAP, WSDL, UDDI, REST, and BPEL. These XML applications are built upon the XML-Rings technologies and form the basis for *Web Services*. At this advanced level we are getting into business processes, which is probably where you want to go in the long run anyway, but first, let's establish a foundation.

In summary, whatever web-work you do, I am sure you will need the six technologies explained within these pages.

The Ring of Technologies

O.K., here is a list of the technologies that make up a ring (based on a designated Ontology):

- HTML - Hypertext Markup Language, [creates web page content, (this is the *Model* part of the MVC paradigm) descriptions for browser rendering]
- CSS2 - Cascading Stylesheets version 2, [styles web page descriptions, for browser rendering] This is the *View* part of the MVC paradigm.
- XML - eXtensible Markup Language [provides a syntax and grammar for document and data structure protocols]. This is another representatoin of the Model of MVC.
- XML-Schema - eXtensible Markup Language, Schema Definition [provides validation/quality control and quality assurance for XML document systems]
- XPath 2.0 - XML Path Language [provides an expression language for use within a *host* language, for XML document searching and (limited) processing]
- XSLT 2.0- eXtensible Stylesheet Language-Transformation [provides a way to transform an XML structure to a different structure, using XPath as a search sub-language]
- and back to HTML/CSS

Whew! Yes, I know that looks like a lot to digest, and you're right! Relax though, I'll do my best to introduce each technology in easy increments, and also show how one technology flows into another. I'll go around each ring in an *iterative* fashion, *implementing* each technology (and implicitly implementing the ontology) as needed, and aiming to produce HTML pages as a *deliverable*.When I go around the ring again, I will *increment* the ontology and each technology. I'll keep going around the ring until we are both satisfied! I'll lay out a series of steps to do all this next.

How I Will Present the Ring and the Ontology

Here are the steps I will go through so that you can learn these technologies in an effective and interesting way.

First: I will illustrate how to use this technology ring within an attractive and commonly understood domain of ideas, the *organic home garden*. I picked this domain since I am a gardener myself and hope that some of the information used to illustrate the technologies might also be of interest to other (potential) gardeners. Even if all you grow is a few plants in a window box, (or are even thinking about setting out a few container plants), I think you will be able to relate to these examples and perhaps reinforce or re-awaken your gardening sub-personality! I will refer to this domain of ideas as the *Garden Ontology*.

Second: I will treat the six technologies listed above as a *ring* of related technologies, where each is successively inter-related to the other, starting with HTML, through CSS, XML, XML-Schema, XPath, XSLT and back to HTML. The process of *going around the ring* represents an *iteration* that will result, from this manual's perspective, in an HTML set of pages. I will interpret the HTML

pages that are the result of a ring iteration as a *deliverable*, either to yourself, or a customer. Note that the purpose of going around the ring in the first place is to represent, manipulate/transform/transmit, and finally present ideas from a domain of interest, a garden domain in this case. (Other deliverables, focused on one of the other ring technologies will be produced as a matter of course, but I have chosen to focus on HTML as a target deliverable of each iteration.)

Third: I will "go around" four of these rings (MicroRing, Ring0, Ring1, and Ring2). Each time I go around, I will include more of the domain's information (that is, more of the garden ontology) which will in turn, require a deepening of each supporting technology. That is, I will introduce more of the domain's variety/complexity and call on more of each technology's capabilities in order to support it. I call these successive deepenings, the increments of both the ontology and each of the ring technologies. A picture might help at this point, so take a look below: (I will explain all of this as we go along, but for those who like an initial overview, here it is).

Fourth: For each of the technologies, I have provided a separate short chapter explaining the theory as implemented within the rings. For example, for HTML, there is a chapter called, "HTML"! Similarly, there is a chapter for CSS and so on for each of the other technologies.

In summary: I will consider the activity of going around a ring an *iteration* resulting in a deliverable of 'styled' HTML pages. Going around again will involve *incrementing* each of the technologies, as well as the ontology. This represents a two-dimensional pattern for learning: with one dimension going around the ring, and a second dimension projecting radially out through each individual technology. (Remember your analytic geometry - this would map roughly to a polar coordinate system with radii shooting out across circles). Finally, as mentioned above, I have provided a tutorial chapter for each technology.

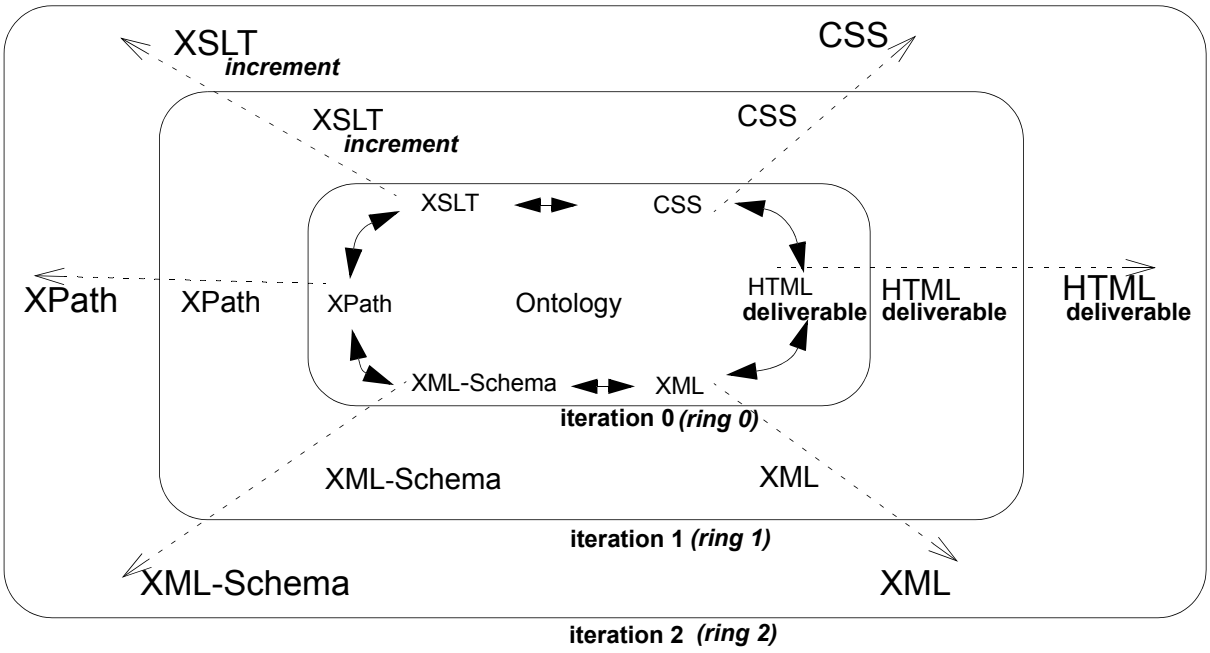


FIGURE 1. Iterative, Incremental Development Rings and Their HTML Deliverables

What you are seeing in the above diagram is my take on how to present and explain a complex set of technologies that are in service of implementing a given application domain. Notice that the "Ontology" is in the center of everything, since ultimately, the implementation of the ideas in the

ontology are the purpose of all this effort. The innermost ring shows a sequence progressing from an XML representation, to an XML-Schema validation, to an XPath/XSLT transformation that takes XML and produces HTML pages styled with CSS. I call this traversal of the ring an *iteration.* In this manual, I am using the HTML pages as the target *deliverable*, since that is what the user/customer will see. As you go around this ring you will be developing and releasing software *artifacts* (possibly deliverables in themselves) of each of the technologies. When you *increment* any of these technologies, or the ontology, I will consider that part of another ring (of course, you are free to group these increments any way you like). I will call each of these artifact releases an *increment* of that technology within that *iteration*. For example, in the diagram above, the next ring, Ring-1, would be identified because it featured increments from one or more of the technologies, and/or the ontology.

For example, for the inner, zeroth ring, the artifacts would be an XML document(s), an XML-Schema document(s) that validates that XML, an XPath/XSLT document(s) that transforms that XML to HTML pages, and a CSS style sheet(s) that styles the resultant HTML pages. Each successive iteration will usually require an increment of one or more of these technologies. (Depending on your customer, you may want to consider other technology increments as *your* target deliverables, or even the whole set of increments associated with an iteration).

On the diagram I have labeled only enough of the entries to suggest the overall pattern. For example, I have labeled the increments of the XSLT technology as it is progressively extended to support additional requirements, all the other technologies should be labeled similarly. The radial arrows are also intended to be inserted within each of the displayed rings, in the same pattern with solid arrows indicating circular iteration while the radial arrows indicate technology increments.

Cut to the Chase Description of the Rings Technology

Here is a preliminary look at the “ring” technologies that we will study and implement. For this first exposure, I have taken a (very) minimal example of each technology as well as a modest portion of the ontology, a single term, *garden*! I will explore each more deeply with you as we go along, but for now this will give you a taste. I will show each of the six technologies in very simple example format. Before each example, I will provide a snapshot description of the technology. Of course you will want to know more - and for sure more is coming as you continue to read and study this manual, including the tutorial chapters. The next diagram shows all of the technologies on one diagram. Following this diagram I will go through each file and explain it.

```
<html>
<!-- gardenMicro.html , part of  Micro-Ring 2005-07 -->
 <head>
   <title>Garden (browser window title)</title>
   <link href="gardenMicro.css" rel="stylesheet"
         type="text/css" />
 </head>
 <body>
 <h1>garden Ontology</h1>
   <p>Garden Ontology ideas go here. </p>
 </body>
</html>
```

HTML

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- gardenMicro.xml -->
<garden>Garden ontology ideas go here. </garden>
```

XML

```
/* gardenMicro.css part of Micro-Ring
* 2005-07
*/
body
{ margin:2em;
background-color: rgb(245,245,245);
color: black;
font-family:serif
}
h1 { font-family:sans-serif; color:teal; }
```

CSS

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- gardenMicro.xsd -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element name="garden" type="xs:string"/>
</xs:schema>
```

XML-Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- gardenMicro.xslt -->
<xsl:transform version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:fn="http://www.w3.org/2005/
02/xpath-functions" xmlns:xdt="http://www.w3.org/2005/02/xpath-datatypes">
<xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"/>
<xsl:template match="/garden" >
 <html>
  <head>
 <title>Garden (browser window title)  </title>
  <link href="gardenMicro.css" type="text/css" rel="stylesheet" />
 </head>
  <body>
  <h1><xsl:value-of select="name()" />
  <xsl:text> Ontology</xsl:text>
 </h1>
  <p><xsl:value-of select="." /></p>
  </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

XPath 2.0/XSLT 2.0

FIGURE 2. All the Ring Files (except the resultant transformed HTML file)

Real World Comments & Caveats on Ring Development

Let me take a moment here and expand a little on the sequence of events that might actually take place within a development environment like I am proposing. I am again referring to the diagram above, "Iterative, Incremental Development Rings and Their HTML Deliverables" on page 4. First off, all diagrams like these are approximations to what actually goes on, which is the human capability to switch back and forth very rapidly between kinds of activities, and tasks within those activities. These changes consist of not only technology increments in the short run, but also changes in business and technological constraints over the longer term. Very few, if any, processes in life are 'linear', but that is often a way to start thinking about certain aspects of them. Software development is no different, even though I am starting out in a more sophisticated manner by proposing 'rings'! Anyway, what actually happens is that developers switch back and forth, often very rapidly (sometimes on the order of minutes or hours) between different levels within a technology, or even between technologies, depending on the emergence of ideas, interest, and time. For now though, bear with me and pretend that we can actually follow a defined sequence of activities, as in a ring fashion.

Take a look at the inner circle, that represents the zeroth ring. This is the initial iteration of a trip around the ring and the arrows are a possible sequence of development efforts. What is interesting about this perspective though is that a ring allows entry at *any point*. If you are just starting out a project, for example, then its often useful to work up some mock-ups of how the system will work/look when developed. Often, user interviews, use cases, or user stories are the basis for these system 'outlines'. This means that you might start off with just paper sketches, or screen shots, or a minimal 'prototype'. In other fields, such as the movie industry, there is great emphasis on 'storyboards' that visually and textually lay out the plot progression, and anticipated events.

In the modern software world, development methodologies such as Extreme Programming (XP) in particular, or Agile Development in general, proceed under the guidance of 'user stories'. In our case, this means that we might actually start off with short narratives of what the system is to present, paper sketches reflecting user requirements, or perhaps an ad-hoc set of HTML pages that are meant only to be suggestive of what the final pages will look like. These initial attempts may precede any of the other technology initiatives and go directly from parts of the ontology to HTML. This first set of HTML pages might well be our first, prototype, 'deliverable'.

After this first, preliminary set of HTML, you will usually have to step back and start to represent your ontology in a more general fashion, by using XML. This is the phase where you might start to consider the issues of scalability, extensibility, business to business inter/intra-transmission, and comprehensiveness within your domain. Reading, expanding, and 'refactoring' the ontology and then transferring that into XML notation would be a logical next step.

Next, or perhaps concurrently, you might develop the XML-Schema that precisely specifies the ontology so far, as represented by the XML document(s). Some people actually start out by developing the XML-Schema before any XML specific documents, on the grounds that this is the defining document as regards structure and content for any specific XML documents. This general schema activity then imposes 'quality assurance' on the XML documents that will be subsequently produced and the application of a specific schema to a document instance constitutes an example of "quality control".

XML is a great representation syntax, but right now the browsers don't know what to do with an XML file. That's where a transformation to a browser friendly format such as HTML/CSS comes in. This will be the role of XSLT, which will take an XML document and output an HTML document, under the control of a file of rules, called an XSLT stylesheet, that you set up. A bonus fea-

ture of XSLT is that we can include in the transformation from XML to HTML, an embedded link to a CSS file. The result is an HTML file that has embedded within it a linkage to a CSS file, thus allowing a more pleasing browser rendering.

So, that's a more realistic look at operating within the ring; now it is time to look at some actual code from each of these technologies. This will be a super 'light' introduction and I will deepen the experience of each as we go along. In addition, there is a separate tutorial chapter on each of the individual technologies that you can refer to for added detail.

HTML - the *Hypertext Markup Language*

HTML is an application of SGML (that is, it has a fixed set of tags conforming to a formal SGML syntax). Originally developed around 1992 to enable exchange of scientific reports by embedding coded tags (HTML markup tags) within the text, the simplicity of this approach spread to millions of developers and led to millions of the web pages we see today. One defining feature of HTML markup is the capability to link (called HyperLinking) one document to another, thus creating a web of information paths, a truly novel innovation. So, HTML is a simple tagging language that all browsers understand how to display. HTML allows anyone to quickly and easily construct "web pages" that can be served over the net just by using a simple text editor and typing a file of text with embedded markup tags. The example below shows an HTML program file. A browser would render (interpret and display) this program using all default typographic settings with the <title> text in the browser window and the <p> text in the browser's content area.

To learn more, go to the HTML and CSS chapters.

Ex. A1: An (ad-hoc) HTML Program

Assume you typed this into a file named **gardenMicro.html.** When opened in a browser, it will be displayed using the browser's built-in typographic defaults. (Go ahead and try it!)

```
<html>
<!-- gardenMicro.html, part of Micro-Ring 2005-07 -->
 <head>
   <title>Garden (browser window title)</title>
   <link href="gardenMicro.css" rel="stylesheet" type="text/css" />
 </head>
 <body>
 <h1>Garden Ontology</h1>
   <p>Garden Ontology ideas go here. </p>
 </body>
</html>
```

Here is a short description of this file and its features:

- `html` - this is the overall containing element, called the root element (outermost element) and is the signal to the browser to treat this file as an 'HTML' file. Note that everything is between the start tag <html> and the ending tag </html>. This is the 'container' idea with start and end tags delimiting content. Note that the start and end tags, plus all their content in between, is a concrete representation of what is called the `html element.`
- <!-- this starts a comment that terminates with the symbols "-->"

- `head` - this is the container element for meta-data. Meta-Data is data that is contained within the `head` tags, and can include the browser window title, key words, links to other files, text content type, and various other instructions to the browser. In this case, I have only included two such tags. `title` meta-data, and `link` meta-data.
  1. `title` - this element's content is rendered in the browser window's title bar and is considered meta-data. The content of this element is: "Garden (browser window title)".
  2. `link` - this element's attributes link up a CSS file that will provide additional styling to the HTML file.
- `body` - this is the container tag for all of the content data that will appear in your browser's content window. (Remember, the `title` content appeared in the browser title bar, not the content window).
  1. `h1` - this is a first level head, that is, it is accorded most importance by the browsers typographical assignments.
  2. `p` - usually indicates paragraph text. In this case the `p` tags enclose simple text.

Here is a browser rendering of this HTML file, **gardenMicro.html.** Note that this rendering shows the effect of the CSS linked file. The details will be discussed next.

CSS - the *Cascading Style Sheet* I

The CSS language is a rule based, pattern matching language that can give HTML documents a professional and attractive look. The rules consist of typographic instructions to the browser, *keyed to matching HTML tag names*. This language allows the *separation* of presentation from content. CSS provides the presentation instructions, and HTML is used to structure the content of what is to be conveyed to the target agent (user). Linking a CSS file of style rules to an HTML document, allows a browser to use the CSS typographic rules to override its (browser's) usual default rendering. (CSS can also be used to style XML documents and XHTML documents as well as HTML docs). CSS is made for the web and is a lightweight styling language that can run on the client side of a processing chain, such as within a browser rather than on a web server. The CSS program below, when linked to the HTML program above, instructs the browser to use the CSS rules whenever they conflict with the browser's built in rules.

To learn more, go to the CSS chapter.

Ex. A2 The CSS Program that styles the HTML

Assume you typed this code into the file named **gardenMicro.css**

```
/* gardenMicro.css part of Micro-Ring
* 2005-07
*/
body
{     margin:2em;
      background-color: rgb(245,245,245);
      color: black;
      font-family:serif
}
h1
{
      font-family:sans-serif; color:teal;
}
```

An abbreviated explanation is:

- the first line is a comment. Note that this syntax is the same as programming languages like C, C++, C#, and Java.
- body - this is a 'selector' that will match an HTML tag of the same name, and apply the typographic instructions/rule found within its braces. In this case, the instructions tell the browser to do the following (that is, typographic events):
  1. insert space (margins) on all four sides of the content area. The amount of space inserted is "2em". This is a printers measure of space and is equal to the width of two capital "M" letters. This sets off the text from the edge and looks a lot better than having it jammed up to the edges, don't you think?
  2. make the background of the page light gray, and the color of the text 'black'.
  3. Also, use the font family with 'serifs' for all text. The browser will then use its built-in defaults for all other typographic events, such as font size, margins, line space, and many others.
- h1 - this is a first level heading and the browser treats it accordingly. In this case, I have overridden the default color of the heading text, which would be black, and changed it to "teal". Additionally, I have instructed the browser to use one of its "sans-serif" fonts.
- p - this is a 'selector- that will match an HTML tag of the same name. Because p is nested inside of the body element in the HTML document, it inherits all of the rules that were called out in `body` rule within the CSS document. So, by inheritance, the CSS rules say that text is to be set black on a white background, in the font 'Times New Roman'. This cascading of the rules from parent to descendent, is the origin of the *Cascade* part of the name CSS. Ah, but in this case the `p` rule does override one of the `body` rules, and so the color attribute is overridden and the result in the browser is red text rather than black.

XML - the *eXtensible Markup Language*

XML is the web's meta-language. It is a subset (lightweight version) of a more comprehensive lan-

guage called SGML (Standard Generalized Markup Language). XML is designed for the internet, e-Business (electronic business) and nowadays, used for e-EverythingElse. This is the foundation language for the capability for representing *structured*, *semi-structured*, and *unstructured* data, simultaneously. (Please carefully reread that last sentence since it mainly accounts for the spread of XML throughout the modern technology world). XML based applications enable representation, manipulation, and transmission of web documents and data with a minimum of technical knowledge and maximum flexibility between sender and receiver. Further, it is human readable and relatively obsolescence-proof, since it is expressed in natural language as opposed to a numeric protocol. The example below is a simple example of how you might start to represent parts of the Garden Ontology.

To learn more, go to the XML chapter *** TBD*.

Ex. A3 An XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<!--gardenMicro.xml -->
<garden>Garden Ontology ideas go here. </garden>
```

- the first line is a 'declaration' that characterizes this file as conformant to the XML syntax, `version` 1.0 (finalized in 1998). The next attribute, `encoding,` refers to mapping between characters and their numeric identifiers (code points). UTF-8 is often called 'compressed' Unicode since it supports a variable number of bytes to represent Unicode characters.
- the second line is a comment that is useful for documentation, generally recommended.
- `garden` - this is a tag that represents part of the Garden Ontology, namely, the general name for the ontology. This is, of course, an extremely simple encoding of the ontology and will be expanded in subsequent rings, but it is easy enough to understand this first time around.The content of this `garden` element (the element being represented by the starting and ending tags plus their included content,) is a simple text data type. That is, the material between the tags is called the element content. In this case it is a simple type consisting of just characters (no embedded elements or attributes), but in other cases will be more complex.

XML-Schema - the *eXtensible Markup Language - Schema*

XML-Schema is the validation technology that ensures that your XML documents mean what you intend. An XML-Schema is analogous to a Relational Database Table Schema; it describes both the structure and content (data types) of specific XML documents. XML-Schema is one of the replacements of the original validation language (Document Type Definition Language (DTD)) that came with XML itself. This original language DTD, was found to be insufficient to represent the many distinctions required in modern commerce. For example, where the DTD allowed you to specify one data-type for content, namely character data, XML-Schema allows some 44 primitive types plus you can define your own. XML-Schema allows an author to specify a structure for an ontology as well as the data-types for its content. The example below provides validation for XML documents that look like the above example. That is, this schema, if applied to any XML document, will reject (invalidate) any that don't start with a <garden> tag and consist solely of text content within that tag. That is, this schema separates all XML documents into two categories, those that conform (are valid) and those that don't!

To learn more, go to the XML-Schema chapter ***TBD.

Ex. A4 XML-Schema

Assume this is typed into the file **gardenMicro.xsd**

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- gardenMicro.xsd -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" ="qualified" attributeFormDefault="unqualified">
<xs:element name="garden" type="xs:string" />
</xs:schema>
```

- the first line is a 'declaration' that characterizes this file as conformant to the XML syntax, `version` 1.0 (finalized in 1998). The next attribute, `encoding,` refers to mapping between characters and their numeric identifiers (code points). UTF-8 is often called 'compressed' Unicode since it supports a variable number of bytes to represent Unicode characters.
- xs:schema - is a tag that represents the outermost element `schema`, that is part of the vocabulary set associated with XML-Schema. The prefix xs: is a nickname/shorthand for a unique identifying 'namespace' for the XML-Schema vocabulary, which is actually the XML-Schema ontology). Although namespaces can get tricky, they are simply a way to distinguish two or more terms that are spelled the same. For example the if the word 'table' (furniture table, chemical element table, numeric array table, water table, HTML table, . . .) appears multiple times in the same file, what is its meaning in each case? For natural text, the context of the word usually separates out the meaning as I did above. In XML documents though, it is convenient to use a unique text string to set the context and then further provide a shorthand nickname called a 'prefix' that refers to the unique string. For an even more common example, we use people's last names to distinguish between Kim Jones and Kim Smith. In this case, the first Kim belongs to the 'Jones' family namespace while the second Kim is in the 'Smith' namespace.
- xmlns:xs="http://www.w3.org/2001/XMLSchema" - this construction identifies the prefix `xs:` with a unique string, "http://www.w3.org/2001/XMLSchema". That string *is* the namespace for the XML-Schema vocabulary. (I talk more about namespaces in the XML chapter)
- `elementFormDefault="qualified" attributeFormDefault="unqualified"` is an advanced option that allows the schema author to constrain an individual XML document author to use namespace prefixes or not. This topic will be discussed in a later "ring" and in the XML chapter.
- the second line is a comment that is part of the documentation for the file.
- `<xs:element name="garden" type="xs:string" />` declares and defines the garden element, and says that its content must be a simple text string (no embedded elements or attributes).

XPath - Expression Language for Searches

XPath is an expression language that is designed to manipulate values conformant to its data model. In practice, XPath is used mainly as a versatile *search* sublanguage within many other XML

languages such as: XSLT, XML-Schema, XForms, XPointer, and XQuery. XPath can *select* out sequences of items that are then worked on by the host language. XPath itself has some computational capabilities, especially in the latest specification of XPath 2.0. So, XPath is an *expression language* that is primarily targeted at selecting sequences of nodes from an XML tree, but has been extended to also handle sequences of mixed data types such as number, strings, and nodes. Further extensions to XPath now include comprehensive function libraries, regular expression support, standard programming expressions, and predicate logic. (Note: the XPath example is combined with the XSLT example below).

To learn more, check out the XPath chapter.

XSLT - the *eXtensible Stylesheet Language - Transformation*

The XSLT language enables us to map (transform) XML documents into browser friendly HTML, styled with CSS. (actually, XSLT can be used to restructure any XML document into practically any other structure, but we will concentrate on the mapping from XML to HTML). XSLT is a functional language that operates by pattern matching, targeted at selecting and processing nodes in an XML tree. This "tree" arises since whenever the XSLT processor works on an XML document, it first transforms it into a tree of nodes before it goes any further. XSLT uses XPath to select out nodes from the XML tree, and then processes those selected nodes.

The latest version of XSLT, incorporates capabilities such as processing multiple input documents as well as producing multiple output documents with comprehensive processing along the way. This is the technology that allows processing that results in a table of contents or output that can be rendered as Portable Document Format (PDF), HTML, text, or even CSV (comma separated values) flat files . The example below, when applied to "Ex. A3 An XML Document" on page 11, will produce an HTML page that matches the ad-hoc HTML of Example A1. The difference here is that the HTML was produced from XML, rather than directly from the ontology.

To learn more, go to the XSLT chapter.

Ex A5 XPath/XSLT Stylesheet

Assume you have typed this code into the file named **gardenMicro.xslt**

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- gardenMicro.xslt -->
<xsl:transform version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:fn="http://www.w3.org/2005/
  02/xpath-functions" xmlns:xdt="http://www.w3.org/2005/02/xpath-datatypes">
  <xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:template match="/garden" >
   <html>
    <head>
   <title>Garden (browser window title)  </title>
    <link href="gardenMicro.css" type="text/css" rel="stylesheet" />
   </head>
    <body>
    <h1><xsl:value-of select="name()" />
    <xsl:text> Ontology</xsl:text>
   </h1>
    <p><xsl:value-of select="." />
   </p>
    </body>
   </html>
  </xsl:template>
  </xsl:stylesheet>
```

namespace declarations

results format

XPath expression

pass through literals (no processing)

XSLT instruction applied to an XPath expression - all of which are embedded inside of HTML tags

Xpath expression

- line one is the standard XML declaration I have described several times above.
- the second line is a comment
- the third line, `<xsl:transform version = "2" . . .` declares that this is the stylesheet version 2.0 (version 1.0 is still supported)
- `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"` - this sets up the prefix, `xsl:` that is short hand for the namespace string that follows. This namespace delimits the XSLT vocabulary set, that is, the XSLT ontology. So when you later see `xsl:` as a prefix on a name, it means that that name is part of the XSLT vocabulary. (The reader may note that this constraining of a set of names is the key reason to call something an *application*). For example, the names `stylesheet`, `output`, `template` are all part of the XSLT vocabulary since they are prefixed with `xsl:`. This means that the application XSLT is composed of these words, plus other components. The XSLT processor is aware of various standard namespaces, and treats those entries accordingly.
- the next lines set up three more namespaces: XML-Schema vocabulary words (with a prefix `xs:`), XPath functions (with a prefix `fn:`) and XPath data types (with a prefix `xdt:`). This means that if names in this file have these prefixes, then those words come from the vocabularies associated with those namespaces. The processor knows about these namespaces and assigns those word/instructions to the proper application.
- `xsl:output` - allows the developer some control about the format of the resultant output. In this case the *`method`* attribute suggests that the rendering software use XML syntax.

- `xsl:template match` - starts an XSLT rule that searches the source document tree, using the XPath search sublanguage, looking for a 'node' called `garden` The way this search path is specified is controlled by the syntax of XPath. This XPath expression, which is a path in this case, is the string: "/garden" and can be read as:
  1. "/" tells the processor to start at the top of the document tree. This 'virtual' top is called the "document root". This is analogous to the root of a file system in Unix where the top of the file system is denoted by "/".
  2. the next part of the path, `garden`, is called a step in the path, and tells the processor to look for all `child` nodes whose parent is that document root. In our particular case there is only one child, and that is the single `garden` element. Since there is such an element in the XML tree, the match succeeds, the element `garden` is selected for processing, the body of the template is instantiated, and processing continues.
- the processing of the `garden` element in this case consists of copying across, to the output, called a result 'tree', the literal text from <html>, all the way down to the <h1> tag. The reason this is so is because none of these tags is prefixed with any special synonym. For that reason, the processor doesn't recognize them as anything special, and therefore simply passes them through to the output format, which is a result tree.
- following the <h1> tag though is an instruction to the XSLT processor. <xsl:value-of select="name()" />. This instruction says to select the name of the node being currently processed, and return it. That name is "garden" and so that is what is written out to the result tree. Following that is another XSLT processing instruction, <xsl:text> that copies its content across to the result tree, maintaining white space as coded.
- further down the listing is the <p> tag. Embedded within it is a processing instruction:
  `<xsl:value-of select="." />`
  that says: select the text content of the current node you are processing, which is the `garden` node, and copy it out to the result tree. This results in the text string "Garden ontology ideas go here." being copied across to the result tree. This is a common way to get an element's content from the XML tree. Note that this text will be bracketed by the <p> tags and so will take on whatever characteristics are in force for this paragraph element, when it is rendered by the browser.

Here is the transformed xml file, **gardenMicroTransformed.html** as rendered in a Netscape browser. Notice that this output is (almost) the same as the original ad-hoc HTML file **gardenMicro.html**. This was deliberate as I wanted to illustrate how we could start with an XML base document and derive an HTML equivalent, rather than starting directly from the ontology in an ad-hoc manner. Generating our HTML from XML documents is the approach we will use for the rest of this manual except for the times when a mock-up or prototype HTML might be constructed in a stand-alone manner. (Note that the only difference is in the capitalization of the word "garden", although that could have been handled as well by the 'string' functions built-in to XPath and XSLT).

The Micro-Ring and What's Next?

O.K., that was our first encounter with the ring of technologies centered on a garden ontology. I hope you enjoyed seeing the 'big' picture that showed the main points and are eager to deepen your knowledge by going around the ring a few more times. So, in general, here's what happened within the Micro-Ring: I started off, as I usually do, with a few 'one-off' HTML files just to get going with something to show, in this case it was a single page. This first page was not intended to do more than generate some interest and suggest some fruitful directions to go in, and I constructed it directly from the vocabulary and structure of the garden vocabulary. At this point, I wasn't concerned with any of the production-level features that will be eventually required, such as validity, reliability, maintainability, reusability, readability, as well as other "ilities". That will come later, as we go around the rings multiple times.

To start on the path to designing a larger system, based on the garden ontology. I created an XML-Schema document that could be used to validate documents of the "garden" category. All this schema says is that such documents must have a `garden` tag and can contain only character data, no embedded elements or attributes, but this is important information. Now I create an XML file in accordance with that schema. It validates so I can then be confident that it conforms to my design.

To render my XML within a browser, I need to transform it to HTML. I do this with an XSLT stylesheet that also includes a callout to a CSS stylesheet that will improve the looks of the HTML.

Now we are ready to deepen our knowledge with the next round of ideas and code. Before launching into a new ring though, I would like to expand on the idea of an *Ontology*. I hope you keep in mind that the *purpose* of all this other technology is to expose/export/display the ideas inherent in the ontology.

Connecting up the XML- Schema with the XML Document

Recall the XML document was written with the intent of validating it with an XML-Schema. As it stands, there is no obvious connection. In some environments the connection is made by underlying software where the Schema and Document are automatically linked up. Let me show you an explicit way to do this though by embedding linking instructions within the XML document itself.

Ex. A3 - An XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!--gardenMicro.xml -->
<garden
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="gardenMicro.xsd"
>

Garden Ontology ideas go here. </garden>
```

Summary

This is the start of a series of XML documents that will detail more of the XML-Rings plus the emerging technologies of XQuery, XForms, and associated XML-Databases such as eXist. Happy XML-ing!