Practical Transformation Using XSLT and XPath (XSL Transformations and the XML Path Language)

Crane Softwrights Ltd.
http://www.CraneSoftwrights.com

+//ISBN 1-894049::CSL::Courses::PTUX//DOCUMENT Practical Transformation Using XSLT and XPath 2004-02-03 23:50UTC//EN
Eleventh Edition - 2004-02-03 *http://www.CraneSoftwrights.com*
ISBN 1-894049-12-8

Practical Transformation Using XSLT and XPath (XSL Transformations and the XML Path Language)

Crane Softwrights Ltd.
http://www.CraneSoftwrights.com

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

Practical Transformation Using XSLT and XPath (Prelude) (cont.)

Preface

The main content of this book is in an unconventional style primarily in bulleted form

- derivations of the book are used for instructor-led training, requiring the succinct presentation
  - note the exercises included in instructor-led training sessions are not included in the book
- derivations of the book can be licensed and branded for customer use in delivering training
- the objective of this style is to convey the essence and details desired in a compact, easily perused form, thereby reducing the search for key words and phrases in lengthy paragraphs
- each chapter of the book corresponds to a module of the training
- each page of the book corresponds to a frame presented in the training
- a summary of subsections and their pages is at the back of the book

Much of the content is hyperlinked both internally and externally to the book in the 1-up full-page sized electronic renditions:

- (note the Acrobat Reader "back" keystroke sequence is "Ctrl-Left")
- page references (e.g.: Chapter 2 Getting started with XSLT and XPath (page 37))
- external references (e.g.: `http://www.w3.org/TR/1999/REC-xslt-19991116`)
- chapter references in book summary
- section references in chapter summary
- subsection references in table of contents at the back of the book
- hyperlinks are not present in the cut, stacked, half-page, or 2-up renditions of the material

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

Practical Transformation Using XSLT and XPath

Series: Practical Transformation Using XSLT and XPath

Reference: PTUX

Pre-requisites:

- knowledge of XML syntax
- knowledge of HTML

Outcomes:

- awareness of documentation
- introduction to objectives and purpose
- exposure to example scripts
- understanding of processing model and data model
- basic XSLT script writing for transformation
- an overview of every element and function in the recommendations
- introduction to XSL formatting semantics

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

Transforming structured information

This book is oriented to the stylesheet writer, not the processor implementer

- certain behaviors important to an implementer are not included
- objective to help a stylesheet writer understand the language facilities needed to solve their problem
    - a language reference arranged thematically to assist comprehension
    - a different arrangement than the Recommendations themselves

First two modules are introductory in nature

- overview of context of XSLT and XPath amongst other members of the XML family of Recommendations
- basic flow diagrams illustrate use of XSLT
- basic terminology and approaches are defined and explained

Third and fourth modules cover essential bases of understanding

- data model and processing model for document representation and behavior
- important to understand the models in order to apply the language features

Fifth through ninth modules address XSLT vocabulary

- every element, attribute and function not already covered when describing the models
- no particular order of the modules, but example code only uses constructs already introduced in earlier content

Transforming structured information (cont.)

First two annexes overview HTML and XSL-FO as related to using XSLT

- considerations of using XSLT features to address basic result vocabulary requirements

Third annex includes a number of handy summaries derived from the Recommendations

- alphabetical lists of elements and functions
- print-oriented summaries of all productions

Last annex addresses questions regarding tools

- lists of questions for processor implementers when assessing tool capabilities
- detailed XT documentation illustrating a real-world use of extension facilities
- summary example invocation information for XT, Saxon, Xalan and MSXML

External ZIP file included with the purchase of the book

- all of the complete scripts utilized in the documentation as stand-alone files ready for analysis and/or modification
- sample invocation scripts for Windows environments

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

Chapter 1 - The context of XSL Transformations and the XML Path Language

- Introduction - Overview
- Section 1 - The XML family of Recommendations
- Section 2 - Transformation data flows

Outcomes:

- an understanding of the roles of and relationships between the members of the XML family of Recommendations (related to XSLT and XPath)
- an awareness of available documentation and a small subset of publicly available resources
- an understanding of the data flows possible when using XSLT in different contexts and scenarios

Overview

Chapter 1 - The context of XSL Transformations and the XML Path Language

This chapter reviews the roles of the following Recommendations in the XML family and overviews contexts in which XSLT and XPath are used.

Extensible Markup Language (XML)

- hierarchically describes an instance of information
  - using embedded markup according to rules specified in the Recommendation
  - according to a vocabulary (a set of element types each with a name, a structure and optionally some attributes) described by the user
- optionally specifies a mechanism for the formal definition of a vocabulary
  - controls the instantiation of new information
  - validates existing information

XML Path Language (XPath)

- the document component addressing basis for XSLT and XPointer

Extensible Stylesheet Language Family (XSLT/XSL/XSL-FO)

- XSL Transformations (XSLT)
  - specifies the transformation of XML-encoded information into a hierarchy using the same or a different document model *primarily for the kinds of transformations for use with XSL*
- XSL (Formatting Semantics, a.k.a. XSL-FO)
  - specifies the vocabulary and semantics of the formatting of information for paginated presentation
  - colloquially referred to at times as XSL Formatting Objects

Namespaces

- disambiguates vocabularies when mixing information from different sources

Stylesheet Association

- names resources as candidates to be utilized as a stylesheet
  - does not modify the structural markup of the data
  - used to specify the rendering of an instance of information

- `http://www.w3.org/TR/REC-xml`

A Recommendation fulfilling two objectives for information representation:

- capturing information in hierarchical markup according to XML-defined constraints
- restricting and/or validating hierarchical markup according to user-specified constraints

XML defines basic constraints on physical and logical hierarchies

- the concept of well-formedness with a syntax for markup languages
  - the vocabulary and hierarchy of constructs in an instance of information is *implicit* according to the specified rules governing syntactic structures
- a language for specifying how a system can constrain the allowed logical hierarchy of information structures
- the semantics of the user's vocabulary are not formally defined using XML constructs
  - can be described in XML comments using natural language
  - are defined by the applications acting on the information

Physical hierarchy:

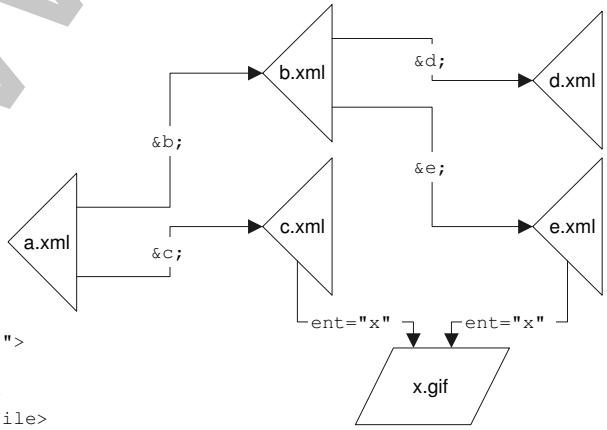
- single collection of information ("XML instance") from multiple physical resources ("XML entities")
  - an XML file is not required to be comprised of more than one physical entity
  - physical modularization typically used for larger information sets
  - inappropriately used for XML fragment sharing due to parsing context
- resource is nested syntactically using XML external parsed general entity construct
  - each physical resource has a well-formed logical hierarchy
- unparsed data entities in a declared notation are outside of the parsed hierarchy

Files:

```
adir/a.xml
adir/c.xml
adir/x.gif
bdir/b.xml
bdir/e.xml
bdir/ddir/d.xml
```

a.xml:

```
<!ENTITY b SYSTEM "../bdir/b.xml">
<!ENTITY c SYSTEM "c.xml">
<!ENTITY d SYSTEM "../bdir/ddir/d.xml">
<!ENTITY e SYSTEM "../bdir/e.xml">
<!NOTATION gif-file SYSTEM "gif-uri">
<!ENTITY x SYSTEM "x.gif" NDATA gif-file>
```



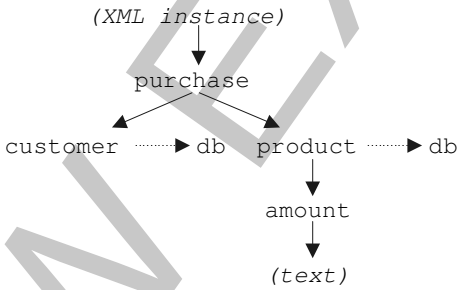
Logical hierarchy:

- single collection of information ("XML instance") comprised of multiple constructs ("XML elements, attributes, text, etc.")
- each piece is expressed using an XML construct at a user-defined granularity
- the breakdown of the information is hierarchical

```
01 <?xml version="1.0"?>
02 <purchase>
03   <customer db="cust123"/>
04   <product db="prod345">
05     <amount>23.45</amount>
06   </product>
07 </purchase>
```

The implicit document model exists by the mere presence of logical hierarchy

- the markup of the XML constructs demarcates the locations of the information in the hierarchy
- data model is comprised of family-tree-like relationships of parent, child, sibling, etc.



The physical hierarchy does not affect the logical hierarchy

- the syntactic boundaries of the physical hierarchy is not reflected in the information model of an XML document

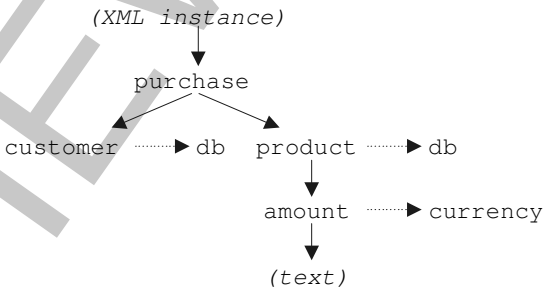
Extensible Markup Language (XML) (cont.)

XML allows user constraints on the logical hierarchy

- defines the concept of validity with a syntax for a meta-markup language
- Document Type Definition (DTD) describes the document model as a structural schema
  - the vocabulary and hierarchy of constructs of information is *explicit* according to user-specified rules governing the logical structure
- other structural and content schema languages exist for XML
  - validation constraints extend to values found within the document content in the document structure
  - different approaches to describing models provide different benefits
- constrains during generation and confirms during processing
- does not convey semantics of information being marked up

```
01 <?xml version="1.0"?>
02 <!DOCTYPE purchase [
03 <!ELEMENT purchase ( customer, product+ )>
04 <!ELEMENT customer EMPTY>
05 <!ATTLIST customer db CDATA #REQUIRED>
06 <!ELEMENT product  ( amount )>
07 <!ATTLIST product  db CDATA #REQUIRED>
08 <!ELEMENT amount   ( #PCDATA )>
09 <!ATTLIST amount   currency ( GBP | CAD | USD ) "USD"> ]>
10 <purchase>
11   <customer db="cust123"/>
12   <product db="prod345">
13     <amount>23.45</amount>
14   </product>
15 </purchase>
```

The DTD can supplement the data model with additional information:



Only declarations affecting the information set of the instance are significant to stylesheet processing that is focused on the implicit logical model of the instance:

- some attribute declarations in DTD are significant
  - attribute list declarations impact stylesheet processing by modifying the information set of the instance
    - supply of defaulted attribute values for attributes not specified in start tags of elements
    - declaration of `ID`-typed attributes (for `ID`/`IDREF` processing) that confer element identification uniqueness in an instance
    - attribute information does not affect the well-formed nature of an XML instance
- all content model declarations are not significant
  - what the logical model could contain does not affect what the actual logical model does contain

No respect of element content white space implied by the content models

- a content model is defined as either element content (a content model without `#PCDATA`) or mixed content (a content model with `#PCDATA`)
- the term "element content white space" is defined in `http://www.w3.org/TR/xml-infoset`
  - sometimes colloquially termed elsewhere as "ignorable white space"
- *all* white space is significant to an XSLT processor
- some recognition of white space can be influenced by the XSLT stylesheet

Extensible Markup Language (XML) (cont.)

XML 1.0 Recommendation describes behavior

- required of an XML processor
- how it must process an XML stream and identify constituent data
- the information it must provide to an application
- note that programming interfaces that have been standardized are separate initiatives and are *not* defined by the XML Recommendation
    - tree-oriented paradigm using DOM (Document Object Model)
    - stream-oriented paradigm using SAX (Simple API for XML)

No rendition or transformation concepts or constructs

- information representation only, not information presentation or processing
- XML only unambiguously identifies constituent parts of a stream of hierarchical information
- no inherent meanings or semantics of any kind associated with element types
- no defined controls for implying rendering semantics
- the `xml:space` attribute signals whether white space in content is significant to the data definition

XML information links

Links to useful information

- `http://www.xml.com/axml/axml.html` - annotated version
- `http://xml.coverpages.org/xml.html` - Robin Cover's famous resource collection
- `http://xml.coverpages.org/xll.html` - Extensible Linking Language
- `http://www.ucc.ie/xml/` - Peter Flynn FAQ
- `http://www.xmlbooks.com/` - a summary of available printed books
- `http://www.CraneSoftwrights.com/links/trn-20040203.htm` - training material
- `http://www.CraneSoftwrights.com/resources` - free resources
- `http://XMLGuild.info` - consulting and training expertise
- `http://xml.coverpages.org/elementsAndAttrs.html` - a summary of opinions

Related initiatives

- `http://www.w3.org/TR/xmlschema-0/` - W3C XML Schema
- `http://www.oasis-open.org/committees/relax-ng/` - RELAX NG (based on RELAX and TREX)
- `http://www.ascc.net/xml/schematron/` - Schematron
- `http://www.w3.org/TR/DOM-Level-2/` - Document Object Model Level 2
- `http://www.saxproject.org` - Simple API for XML

Examples of processors

- `http://www.jclark.com/xml/xp/` - XP (Java)
- `http://xml.apache.org/` - Xerces (Java, C++ and Perl Interfaces)
- `http://www.xmlsoft.org` - XML for Gnome
- `http://www.textuality.com/Lark/` - Lark (Java)
- `http://www.ltg.ed.ac.uk/software/xml/` - LT XML
- `http://msdn.microsoft.com/downloads/webtechnology/xml/msxml.asp` - Microsoft
- `http://www.a-dos.com` - XML processor and associated tools
- `http://www.xmlsoftware.com/` - list of tools

The above list is just some of the early or interesting processors of the very many that are available commercially and publicly.

Addressing structured information

- `http://www.w3.org/TR/xpath`
- a data model for representing an XML document as an abstract node tree
  - the original markup syntax is not preserved
  - the user constraints on the document model (e.g. DTD content models) are not germane
  - any logical or physical modularization (the use of entities) is not preserved
- a mechanism for addressing information found in the document node tree
  - the address specifies how to traversal the data model of the instance
- a core upon which extended functionality specific to each of XSLT and XPointer is added
  - an expression of boolean, numeric, string and node values as different data types
  - a set of functions working on the values

Addressing identifies a hierarchical position or positions

- common semantics and syntax for addressing a logical hierarchy
  - document order, a.k.a. parse order, a.k.a. depth first order
- no representation of the physical hierarchy of an XML document
- a compact non-XML syntax
  - for use in attribute values of XML documents
  - `select="id('start')//question[@answer='y']"`
    - select all question elements whose `answer` attribute is "`y`" that are descendants of the element in the current document whose unique identifier is "`start`"

XPath 1.0 is *not* a query language

- only based on XML 1.0 and Namespaces in XML 1.0
- one aspect of querying is addressing information that needs to be found
- other aspects of querying involve working with the information that is addressed before returning a result to the requestor
- XPath is used only to address components of an XML instance, and in and of itself does not provide any traditional query capabilities (though hopefully would be considered as the addressing scheme by those defining such capabilities)

Styling structured information

Styling is *transforming* and *formatting* information

- the application of two processes to information to create a rendered result
- the ordering of information for creation isn't necessarily (or shouldn't be constrained to) the ordering of information for presentation or other downstream processes
    - it is a common (though misdirected) first step for people working with these technologies to focus on presentation
    - the ordering should be based on business rules and inherent information properties, not on artificial presentation requirements
    - downstream orderings can be derived from constraints imposed upstream in the process
    - information created richly upstream can be manipulated into less-richly distinguished information downstream, but not easily the other way around
    - exception when the business rules are presentation or appearance oriented (e.g. book publishing)
- the need to present information in more than one ordering requires transformation
- the need to present information in more than one appearance requires formatting

W3C XSL Working Group

- chartered to define a style specification language that covers at least the formatting functionality of both CSS and DSSSL
- not intended to replace CSS, but to provide functionality beyond that defined by CSS
    - e.g. add element reordering and pagination semantics

Two W3C Recommendations

- designed to work together to fulfill these two objectives
- XSL Transformations (XSLT)
    - transforming information obtained from a source into a particular reorganization of that information to be used as a result
- Extensible Stylesheet Language (XSL/XSL-FO)
    - specifying and interpreting formatting semantics for the rendering of paginated information
    - the acronym XSL-FO is unofficial but in wide use, including at the W3C, for just the formatting objects, properties and property values
    - XSL normatively includes XSLT by reference in chapter 2
        - XSLT has specific features designed to be used for XSL-FO

XSLT and XSL-FO are endorsed by members of WSSSL

- an association of researchers and developers passionate about markup technologies

- `http://www.w3.org/TR/xsl`

Paginated flow and formatting semantics vocabulary

- capturing agreed-upon formatting semantics for rendering information in a paginated form on different types of media
- XSLT is normatively referenced as an integral component of XSL as a language to transform an instance of an arbitrary vocabulary into the XSL-FO XML vocabulary
- XSL-FO can be regarded simply as a "pagination markup language"
- flow semantics from the DSSSL heritage
    - e.g. headers, footers, page numbers, page number citations, columns, etc.
- formatting semantics from the CSS heritage
    - e.g. visual properties (font, color, etc.) and aural properties (speak, volume, etc.)

Target of transformation

- the stylesheet writer transforms a source document into a hierarchy that uses only the formatting vocabulary in the result tree
- stylesheet is responsible for constructing the result tree that expresses the desired rendering of the information found in the source tree
    - the XML document gets transformed into its appearance
- stylesheet cannot use any user constructs as they would not be recognized by an XSL rendering processor
    - for example, the rendering engine doesn't know what an invoice number or customer number is that may be represented in the source XML
    - the rendering engine does know what a block of text is and what properties of the block can be manipulated for appearance's sake
    - the stylesheet transforms the invoice number and customer number into two blocks of text with specified spacing , font metrics, and area geometry

Device-independent formatting constructs

- the XSL-FO vocabulary describes two media interpretations for objects and properties:
    - visual media
    - aural media
    - a further distinction is also made at times for interactive media
- the results of applying a single stylesheet can be rendered on different types of rendering devices, e.g.: print, display, audio, etc.
- may still be appropriate to have separate stylesheets for dissimilar media
    - device independence allows the information to be rendered on different media, but a given rendering may not be conducive to consumption

- `http://www.w3.org/TR/xslt`

Transformation by example

- a vocabulary for specifying templates of the result that are filled-in with information from the source
  - the stylesheet includes examples of each of the components of the result
  - the stylesheet writer declares how the XSLT processor builds the result from the supplied examples
- the primary memory management and manipulation (node traversal and node creation) is handled by the XSLT processor using declarative constructs, in contrast to a transformation programming language or interface (e.g. the DOM - Document Object Model) where the programmer is responsible for handling low-level manipulation using imperative constructs
- includes constructs to iterate over structures and information found in the source
- the information being transformed can be traversed in different ways any number of times required for the desired result
- straightforward problems are solved in straightforward ways without needing to know programming
  - useful, commonly-required facilities are implemented by the processor and can be triggered by the stylesheet
  - the language is Turing complete, thus arbitrarily complex algorithms can be implemented (though not necessarily in a pretty fashion)
- includes constructs to manage stylesheets by sharing components

Not intended for general purpose XML transformations

- designed for downstream-processing transformations suited for use with XSL formatting vocabulary
  - includes facilities for working with the XSL vocabulary easily
- still powerful enough for *most* downstream-processing transformation needs
  - an XSLT stylesheet can be (and is) called a transformation script
  - absolutely general purpose when the output from XSLT is going to be input to an XML processor
- does not include certain features appropriate for syntax-level general purpose transformations
  - unsuitable for original markup syntax preservation requirements

Extensible Stylesheet Language Transformations (XSLT) (cont.)

Document model and vocabulary independent

- an XSLT stylesheet is independent of any Document Type Definition (DTD) or schema that may have been used to constrain the instance being processed
- an XSLT processor can process well-formed XML documents without a model
  - behavior is specified against the presence of markup in an instance as the implicit model, not against the allowed markup prescribed by any explicit model
- one stylesheet can process instances of different document models
- multiple instances following different document models can be used in a single transformation
- different stylesheets can process a given single instance to produce different results

XML source and stylesheet

- all inputs must be conforming well-formed XML documents
  - one or more source files and one or more stylesheet files
  - starting with a single source file and a single stylesheet file
- Recommendation does not support SGML instances as input because of XML markup conventions
  - see `http://tidy.sourceforge.net/` for interpretation and conversion of instances of the HTML vocabulary into XHTML markup conventions
  - see `http://www.ccil.org/~cowan/XML/tagsoup` for interpretation and conversion of streams of arbitrary HTML constructs
  - see `http://www.jclark.com/sp/sx.htm` in the SP package `http://www.jclark.com/sp` for conversion of SGML instances to XML instances without document type declarations
  - see `http://www.CraneSoftwrights.com/resources/n2x` for conversion of SGML instances to XML instances with document type declarations

Validation unnecessary (but convenient)

- an XSLT processor need not implement a validating XML processor
- must implement at least a non-validating XML processor to ensure well-formedness
- validation is convenient when debugging stylesheet development
  - if the source document does not validate to the model expected by the stylesheet writer, then a correctly functioning stylesheet may exhibit incorrect behavior
  - time spent debugging the working stylesheet is wasted if the source was incorrect all along

Multiple source files possible

- one primary source file
- stylesheet may access arbitrary other source files (including itself as a source file)
    - names of resources hardwired within the stylesheet
    - names of resources found within source files
- multiple accesses to the same resource refer to a single abstract representation
    - one is not built for each access to a named resource

Stylesheet supplements source

- arbitrary information unrelated to the source can be injected into the result at any time from the stylesheet
- synthesis of primary file and any number of supplemental files
    - supplemental files can be included or imported
    - powerful for sharing and exploiting fragments of stylesheets
- entire stylesheet and all supplemental fragments read in and processed before processing source data

Extensible language design supplements processing

- an XSLT processor *may* support extensions specified in the stylesheet but is not obliged to do so
    - extended instructions
    - extended functions
    - extended serialization conventions
    - extended sorting schemes
- access to non-standardized extensions is specified in standardized ways

Abstract structure result

- the result of transformation is built from instantiated templates as an internal hierarchical tree that *may* be serialized externally as markup
- the XSLT processor may, but is not obliged to, externalize the result tree in XML or some other type of syntax if requested by the stylesheet writer
    - the stylesheet writer has little or no control over the constructs chosen by the XSLT processor for serialization
    - the stylesheet writer can request certain behaviors that the XSLT processor is not obliged to respect
    - final result is guaranteed to comply with lexical requirements of the output method (when not coerced by certain stylesheet controls)
    - another reason source tree markup syntax preservation cannot be implemented with XSLT
- the design of the language allows the XSLT processor to immediately serialize the result tree as markup while it is being built by the stylesheet, and not maintain the complete result in memory
- the stylesheet may request the XSLT processor emit the result tree using built-in available lexical conventions (XML, HTML or text-only conventions)

Result-tree-oriented objective

- result generated in result-tree parse order
    - stylesheet design is oriented to the production of the result
- the source trees can be traversed in any predictable order and not necessarily in parse order
- information in the source trees can be ignored or selectively processed
- the result tree is emitted as if produced internally in parse order
    - this is not an implementation constraint, but an implementation must act as if the tree were created in parse order
        - an important distinction for parallelism

Historical development of the XSL and XSLT Recommendations

Recommendation release history:

- first concept description floated in August 1997 with no official status within the World Wide Web Consortium (W3C)
  - `http://www.w3.org/TR/NOTE-XSL.html`
- the XSL Working Group officially chartered in early 1998
  - `http://www.w3.org/Style/XSL/`
- agreed upon requirements for XSL by the Working Group:
  - `http://www.w3.org/TR/WD-XSLReq`
- the XSL 1.0 Recommendation (XSL-FO) published October 15, 2001
  - `http://www.w3.org/TR/2001/REC-xsl-20011015/`
  - some software only supports Candidate Recommendation:
    - `http://www.w3.org/TR/2000/CR-xsl-20001121/`
- the XSLT/XPath 1.0 Recommendations published November 16, 1999
  - `http://www.w3.org/TR/1999/REC-xslt-19991116`
    - `http://www.w3.org/1999/11/REC-xslt-19991116-errata` - errata
  - `http://www.w3.org/TR/1999/REC-xpath-19991116`
    - `http://www.w3.org/1999/11/REC-xpath-19991116-errata` - errata
- XSLT 1.1 (work abandoned)
  - `http://www.w3.org/TR/2000/WD-xslt11req-20000825` - requirements
  - `http://www.w3.org/TR/2000/WD-xslt11-20001212`
  - no incompatible changes to XSLT 1.0 in XSLT 1.1, only additional functionality
  - too many interactions with plans for XSLT 2.0, so functionality to be folded into XSLT 2.0 release
- XSLT/XPath 2.0 (work in progress)
  - `http://www.w3.org/TR/2001/WD-xslt20req-20010214` - requirements
  - `http://www.w3.org/TR/xpath20/` - current XPath work
    - `http://www.w3.org/TR/query-datamodel/` - shared data model with XQuery
    - `http://www.w3.org/TR/xquery-operators/` - shared operators with XQuery
    - `http://www.w3.org/TR/xquery/` - standalone query work
  - `http://www.w3.org/TR/xslt20/` - current XSLT work
- XSL-FO 1.1 (work in progress)
  - `http://www.w3.org/TR/xsl11/`

XSL information links

Links to useful information

- `http://xml.coverpages.org/xsl.html` - Robin Cover
- `http://www.mulberrytech.com/xsl/xsl-list/` - mail list
- `http://www.dpawson.co.uk` - an XSL/XSLT FAQ
- `http://www.zvon.org/HTMLonly/XSLTutorial/Books/Book1/index.html` - numerous example XSLT scripts and fragments
- `http://www.nag.co.uk/projects/OpenMath/corecd/` - OpenMath project work by David Carlisle
- `http://www.CraneSoftwrights.com/links/trn-20040203.htm` - comprehensive XSLT/XPath and XSL-FO training material
- `http://www.CraneSoftwrights.com/resources`- free XSLT and XSL-FO resources
- `http://incrementaldevelopment.com/xsltrick/` - "Stupid XSLT Tricks"
- `http://www.xmlsoftware.com/` - list of tools
- `http://xml.coverpages.org/xslSoftware.html` - list of tools
- `http://www.exslt.org/` - community effort for XSLT extensions
- `http://exslfo.sf.net` - community effort for XSL-FO extensions
- `http://foa.sourceforge.net/` - open source FO GUI authoring tool
- `http://www.xslfast.com/` - commercial FO GUI authoring tool
- `http://www.inventivedesigners.com/` - commercial FO GUI authoring tool
- `http://www.abisource.com/` - word processing with "Save As..." for XSL-FO
- `http://www.AntennaHouse.com/XSLsample/XSLsample.htm` - paginating XHTML
- `ISBN 1-56609-159-4` - "The Non-Designer's Design Book", Robin Williams, Peachpit Press, Inc., 1994
- `ISBN 0-8230-2121-1/0-8230-2122-X` - "Graphic design for the electronic age; The manual for traditional and desktop publishing", Jan V. White, Xerox Press, 1988 (out of print but worthwhile to search for as a used book)

XSL information links (cont.)

Examples of XSLT processors

- `http://www.jclark.com/xml/xt.html` - James Clark
- `http://saxon.sourceforge.net` - Mike Kay
- `http://msdn.microsoft.com/downloads/webtechnology/xml/msxml.asp` - updated web release of XML/XSLT processor for Internet Explorer 5 (IE6 follows the W3C specifications)
  - `http://www.netcrucible.com/xslt/msxml-faq.htm` - useful FAQ
- `http://technet.oracle.com/tech/xml/` - Oracle
- `http://xml.apache.org/xalan/index.html` - Apache Project JAVA-based implementation (originally from IBM/Lotus AlphaWorks)
- `http://alphaworks.ibm.com/tech/LotusXSL`- IBM/Lotus AlphaWorks wrapper for Xalan
- `http://www.xmlsoft.org` - XSLT for Gnome
- `http://www.DataPower.com` - XSLT-dedicated hardware
- `http://www.sarvega.com` - XSLT-dedicated hardware
- `http://www.ambrosoft.com/gregor.html` - XSLT compiler
- `http://www.infoteria.com` - iXSLT - commercial implementation
- `http://www.unicorn-enterprises.com/` - Unicorn XSLT Processor
- `http://www.a-dos.com` - XSLT processor and associated tools

The above list is just some of the early or interesting processors of the very many that are available commercially and publicly.

XSL information links (cont.)

Examples of XSL formatting object rendering processors

- `http://www.AntennaHouse.com/` - AntennaHouse Windows-based and multi-platform versions
- `http://www.RenderX.com/` RenderX - direct to PDF
- `http://www.3b2.com` - Advent 3B2 - direct to PDF
- `http://www.ArborText.com` - Epic composition tool
- `http://www.Adobe.com` - Adobe Document Server
- `http://www.xmlpdf.com/ibex` - Ibex C# .NET - direct to PDF
- `http://www.Lunasil.com` - Java/COM - direct to PDF
- `http://xml.apache.org/fop/` - FOP - direct to PDF, PCL, others
- `http://xmlroff.sourceforge.net/` - open source to PDF
- `http://www.hcu.ox.ac.uk/TEI/Software/passivetex` - Passive TeX - TeX to PDF
- `http://www.unicorn-enterprises.com/` - Unicorn UFO - TeX to PDF
- `http://www.alphaworks.ibm.com/tech/xfc` - IBM XFC - direct to PDF
- `http://www.xmlmind.com/foconverter` - Pixware XFC - XSL-FO to RTF
- `http://www.jfor.org/` - XSL-FO to RTF
- `http://www.xsmiles.org/` - XML browser using FOP

The above list is just some of the early processors of what is anticipated to be very many that will be available commercially and publicly.

- `http://www.w3.org/TR/REC-xml-names`
- `http://www.megginson.com/docs/namespaces/namespace-questions.html`

An important role in information representation:

- vocabulary distinction in a single XML document
  - mixing information from different document models
- possible use for resource discovery being considered
  - generalized associated information regarding information in an instance
  - possible access to document model, stylesheets, validation algorithms, access libraries, etc.

Vocabulary distinction

- specifies a simple method for qualifying element and attribute names used in XML documents
- allows the same element type name to be used from different vocabularies in a given document
  - consider two vocabularies each defining the element type named "`<set>`", each with very different semantics
    - in SVG (Scalable Vector Graphics) the element `<set>` refers to setting a value within the scope of contained markup
    - in MathML (Mathematical Markup Language) `<set>` refers to a collection of constructs treated as a set
  - any document needing to mix elements from the two vocabularies may need to use the same name
    - without namespaces an application cannot distinguish which construct is being used
  - a namespace prefix differentiates the element type name suffix in an instance
    - `<svg:set>`
    - `<math:set>`
  - composite name lexically parses as an XML name
    - the use of the colon is defined by the namespaces recommendation
- also used to uniquely distinguish identification labels in some Recommendations
  - e.g.: customized sort scheme label

Namespaces (cont.)

URI value association

- associates element type name prefixes with Universal Resource Identifier (URI) references *whether or not any kind of resource exists at the URI*
  - URI domain ownership under auspices of established organization
  - URI conflicts avoided if rules followed
- examples:
  - `xmlns:svg="http://www.w3.org/2000/svg-20000629"`
  - `xmlns:math="http://www.w3.org/1998/Math/MathML"`
  - `xmlns:ex1="urn:isbn:1-894049:example"`
  - `xmlns:ex2="ftp://ftp.CraneSoftwrights.com/ns/example2"`
- explicitly does not expect nor require to de-reference any kind of information from the given URI
  - note that the Resource Description Framework (RDF) recommendation does have a convention of looking to the URI for information, though this is outside the scope of the Namespaces recommendation
- according to the recommendation, the URI is *only* used to disambiguate otherwise identical unqualified members of different vocabularies

The choice of the prefix is arbitrary and can be any lexically valid name

- the name need not be consistent with the use (though this helps legibility)
- the name is never a mandatory aspect of any Recommendation
- the prefix is discarded by the XML namespace-aware processor along the lines of:
  - `<{http://www.w3.org/2000/svg-20000629}set>`
  - `<{http://www.w3.org/1998/Math/MathML}set>`
  - the above use of "`{`" and "`}`" are for example purposes only
  - note how the "`/`" characters of the URI would be unacceptable given the lexical rules of names, thus, the URI could never be used directly in the tags of an XML document
- the prefix is merely a syntactic shortcut preventing the need to specify long distinguishing strings

Namespaces (cont.)

Namespaces in XSLT and XSL-FO

- Recommendations utilize namespaces to distinguish the desired result tree vocabularies from the transformation instruction vocabularies
- `http://www.w3.org/1999/XSL/Transform`
  - XSL transformation instruction vocabulary
  - the use of any archaic URI values for the vocabulary will not be recognized by an XSLT processor
- `http://www.w3.org/1999/XSL/Format`
  - XSL formatting result vocabulary
  - the year represents when the W3C allocated the URI to the working group, not the version of XSL the URI represents

Extension identification

- XSLT processors are allowed to recognize other namespaces in order to implement extensions not defined by the Recommendation:
  - functions
  - instructions
  - sort methods
  - system properties
  - serialization methods
- e.g.: `http://www.jclark.com/xt`
  - extensions available when using XT
- e.g.: `http://icl.com/saxon`
  - extensions available when using Saxon

Stylesheet association

- `http://www.w3.org/TR/xml-stylesheet`

Relating documents to stylesheets

- associating one or more stylesheets with a given XML document
- same pseudo-attributes and semantics as in the HTML 4.0 recommendation elements:
  - `<LINK REL="stylesheet">`
  - `<LINK REL="alternate stylesheet">`

Ancillary markup

- not part of the structural markup of an instance, thus it is marked up using a processing instruction rather than first-class (declared or declarable in a document model) markup

Typical examples of use:

```
01 <?xml-stylesheet href="fancy.xsl" type="text/xsl"?>
```

```
01 <?xml-stylesheet href="normal.css" type="text/css"?>
```

Less typical examples provided for by the design:

```
01 <?xml-stylesheet alternate="yes" title="small"
02                  href="small.xsl" type="application/xslt+xml"?>
```

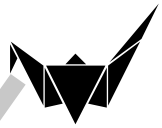
- provide the processor with an alternate stylesheet if some external stimulus triggers it by name

```
01 <?xml-stylesheet href="#style1" type="application/xslt+xml"?>
```

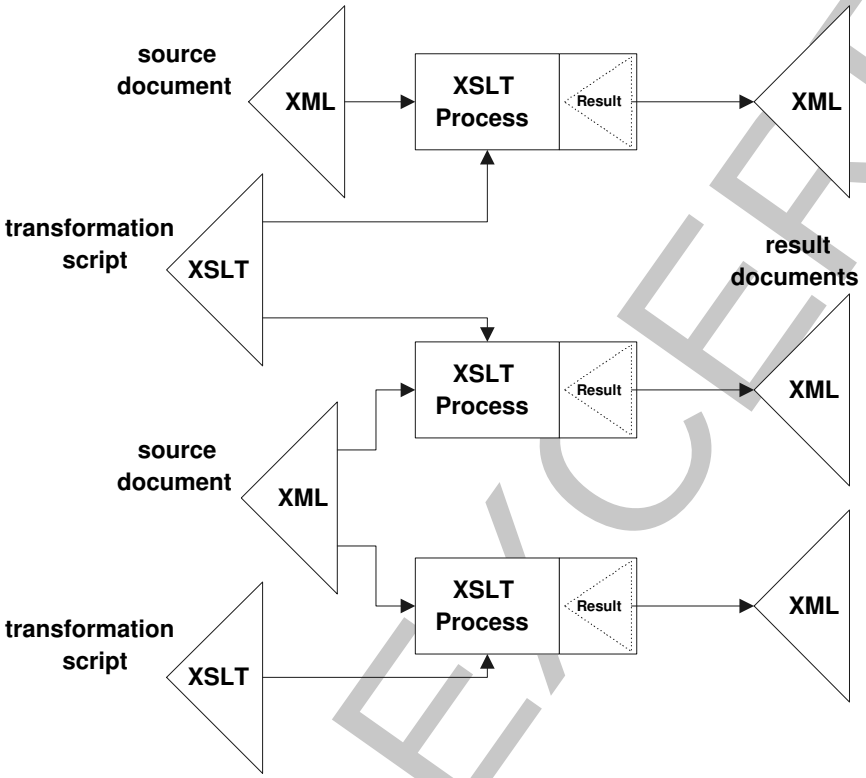
- instruct the processor to find the stylesheet embedded in the source document at the named location

Important note about `type=` values for associating XSLT:

- `type="text/xsl"` is not a registered MIME type
  - the only type recognized by IE for the use of XSLT
- `type="application/xslt+xml"` has been proposed in IETF RFC 3023
- `type="text/xml"` is reported to be supported by some processors



The normative behavior of XSLT is to transform an XML input into a hierarchical result tree; that result tree may be emitted as an XML instance:



Of note:

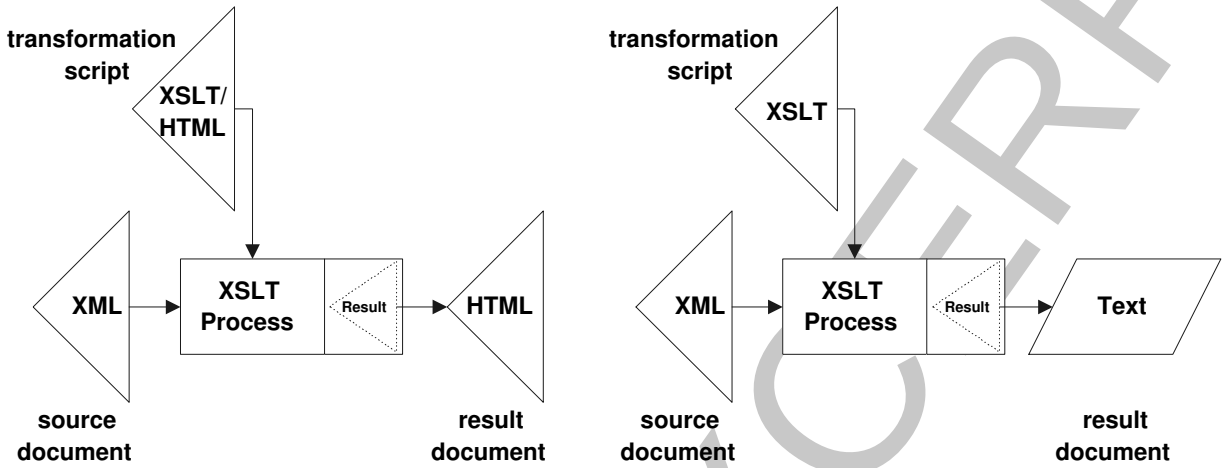
- a given XSLT stylesheet can be applied to more one XML document
- a given XML document can have more than one XSLT stylesheet applied
- the diagram depicts the target of transformation as the abstract result tree within the XSLT transformation process
- the serialization of the abstract result tree to the emitted XML is under the control of the XSLT processor

Diagram legend

- processes represented by rectangles
- hierarchical structures represented by triangles
    - a tree structure with the single root at the left point and the tree expanding and getting larger towards the leaves at the right edge
- unstructured files represented by parallelograms

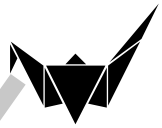
An XSLT processor may choose to recognize the stylesheet writer's desire to serialize a non-XML representation of the result tree:

- triggered through using an output serialization method supported by the XSLT processor



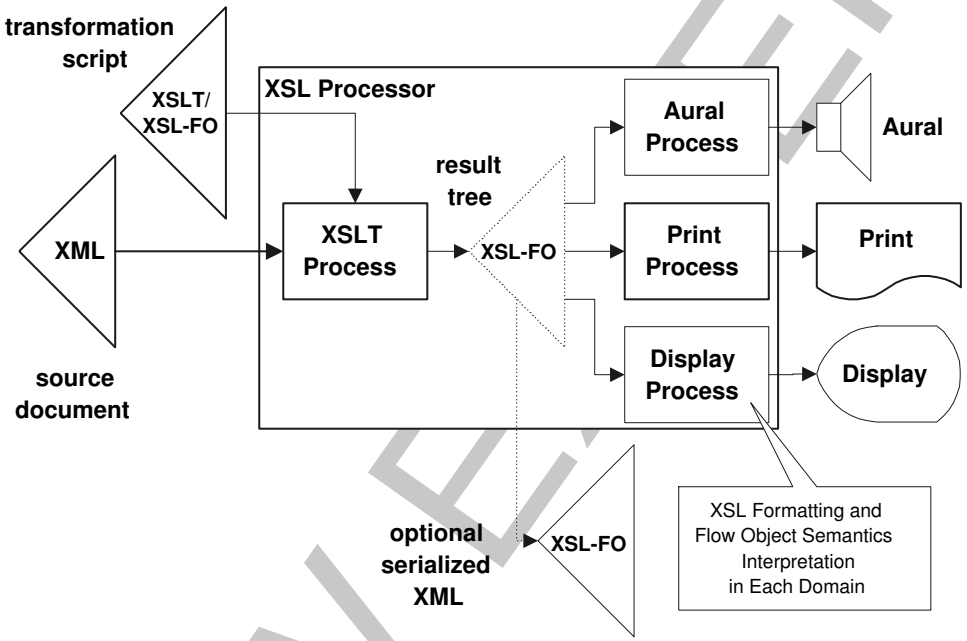
The XSLT Recommendation documents two non-XML tree serialization methods:

- `html`
  - HTML markup and structural conventions
    - some older HTML user agents (e.g. browsers) will not correctly recognize elements in the HTML vocabulary when the instance is marked up using XML conventions (e.g. `<br/>` must be `<br>`), thus necessitating the interpretation of HTML semantics when the result tree is emitted
    - using this will not validate the result tree output as being HTML
      - if the result is declared HTML but the desired output isn't HTML, the HTML semantics could interfere with the markup generated
  - HTML built-in character entities (e.g.: accented letters, non-breaking space, etc.)
- `text`
  - simple text content with all element start and end tags removed and ignored
  - none of the characters are escaped on output
  - example of use: creating operating system batch and script files from structured XML documents



When the XSLT result tree is specified to utilize the XSL-FO formatting vocabulary:

- the normative behavior is to interpret the result tree according to the formatting semantics defined in XSL for the XSL-FO formatting vocabulary
- an inboard XSLT processor can effect the transformation to an XSL-FO result tree
- the XSL-FO result tree need not be serialized in XML markup to be conforming to the recommendation
    - useful for diagnostics to evaluate results of transformation



Of note:

- the stylesheet contains only the XSLT transformation vocabulary, the XSL formatting vocabulary, and extension transformation or foreign object vocabularies
- the source XML contains the user's vocabularies
- the result of transformation contains exclusively the XSL formatting vocabulary and any extension formatting vocabularies
    - does not contain any constructs of the source XML vocabulary
    - may or may not be serialized as XML markup (useful for diagnostics)
- the rendering processes implement for each medium the common formatting semantics described by the XSL recommendation
    - for example, space specified before blocks of text can be rendered visually as a vertical gap between left-to-right line-oriented paragraphs or aurally as timed silence before vocalized content

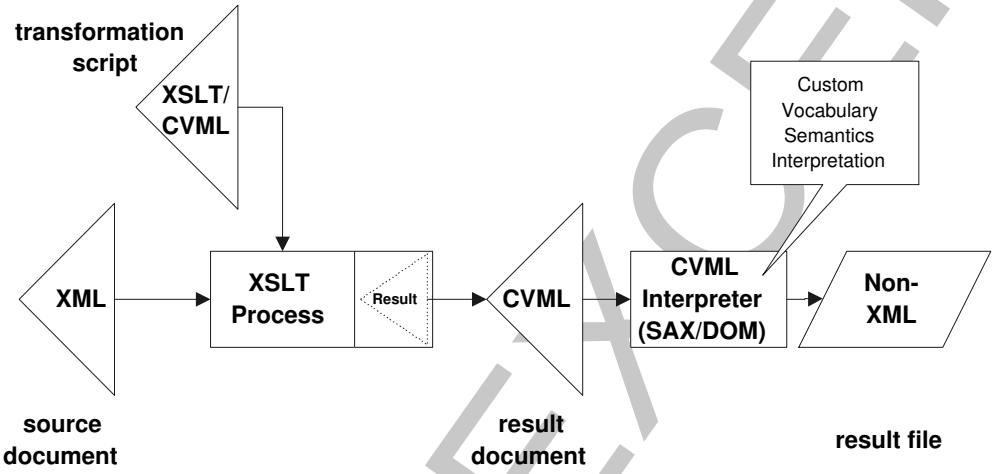
XML to binary or other formats

Some non-XML requirements are neither text nor HTML

- need to produce composition codes for legacy system
- binary files with complex encoding
- custom files with complex or repetitive sequences

One can capture the semantics of the required output format in a custom XML vocabulary

- e.g.: "CVML" for "Custom Vocabulary Markup Language"
- designed specifically to represent meaningful concepts for output



A single translation program (drawn as "CVML Interpreter"):

- can interpret all XML instances using the custom vocabulary markup language (e.g. CVML) to produce the output according to the programmed semantics
- is *independent* of the XSLT stylesheets used to produce the instances of the custom vocabulary
- allows any number of stylesheets to be written without impacting the translation to the final output
- divorces the need to know syntactic output details
  - output is described abstractly by semantics of the vocabulary
  - output is serialized following specific syntactic requirements

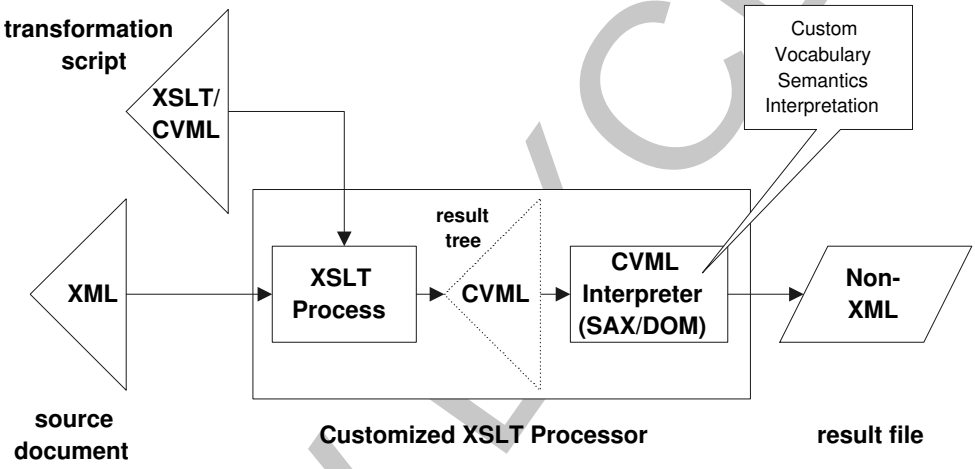
Note how this methodology is no different in concept than the methodology for rendition according to the standardized semantics of XSL formatting objects.

XML to binary or other formats (cont.)

The XSLT recommendation is extensible providing for vendor-specific or application-specific output methods:

- `xmlns:`*`prefix`*`="`*`processor-recognized-URI`*`"`
- *`prefix`*`:`*`serialization-method-name`*
  - vendors can choose to support additional built-in tree serialization methods
  - output can be textual, binary, dynamic process (e.g.: database load), auditory, or any desired activity or result

The ability to specify vendor-specific or implementation-specific output methods allows custom semantics to be interpreted *within* the modified XSLT processor, thus not requiring the intermediate file:



XT implements an extension serialization method:

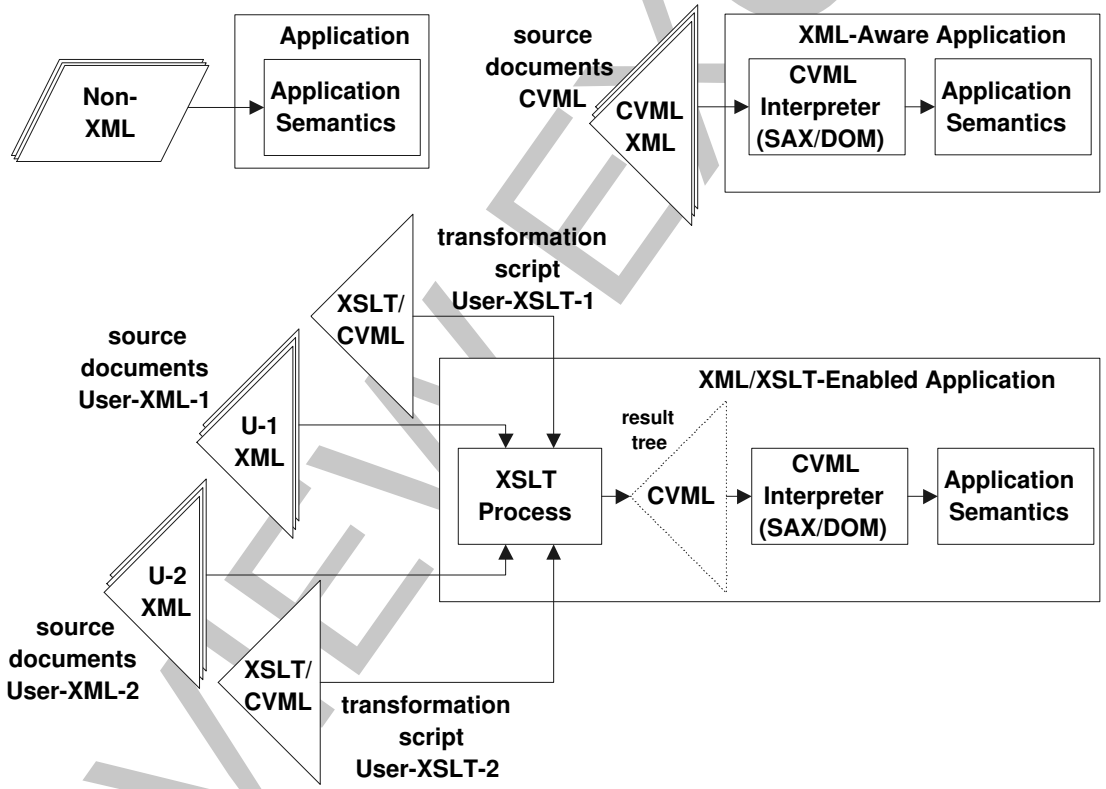
- NXML - Non-XML text environment (documented in detail in Extension method: Non-XML serialization (page 125))
  - `xmlns:`*`prefix`*`="http://www.jclark.com/xt"`
  - `method="`*`prefix`*`:nxml"`

XSLT as an application front-end

Chapter 1 - The context of XSL Transformations and the XML Path Language
Section 2 - Transformation data flows

A legacy application can utilize an XSLT processor to accommodate arbitrary XML vocabularies

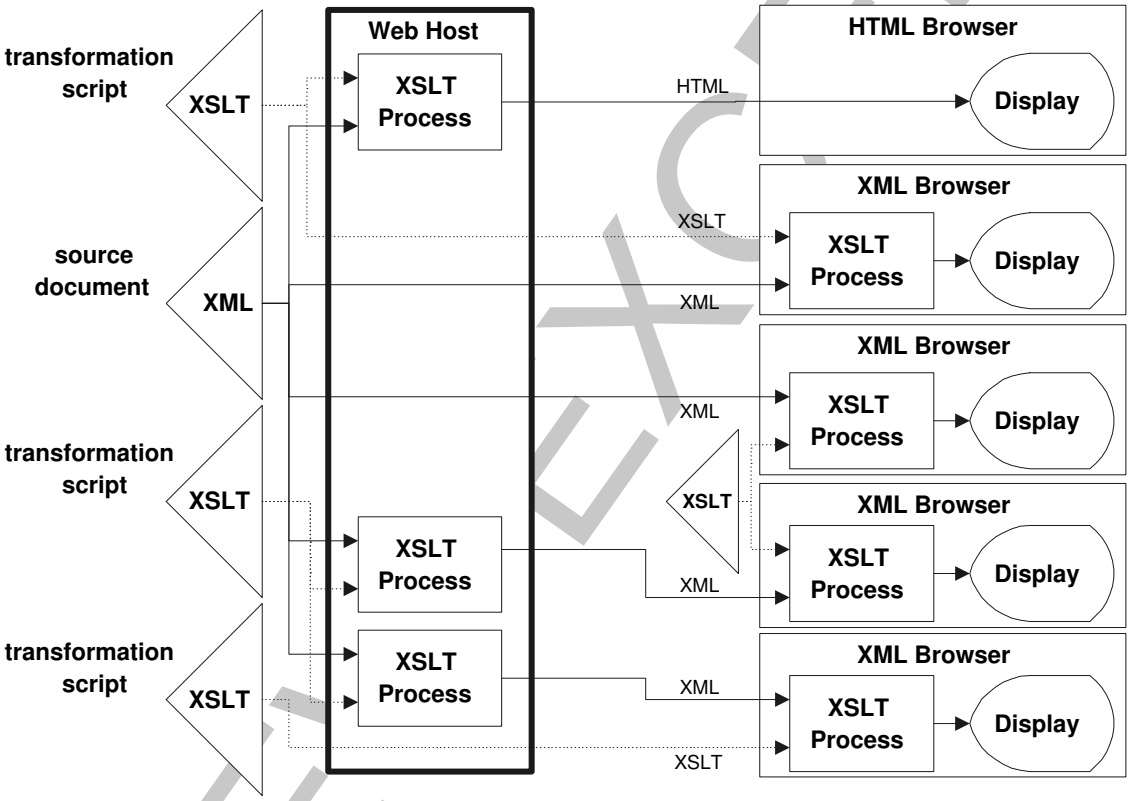
- making an application XML-aware involves using an XML processor to accommodate a vocabulary expressing application data semantics
  - event driven using SAX processing and programming
  - tree driven using DOM processing and programming
  - without XSLT, each different XML vocabulary would need to be accommodated by different application integration logic
- an application can engage an XSLT processor and directly access the result tree
  - single process programmed to interpret a single markup language
  - each different XML vocabulary is accommodated by only writing a different XSLT stylesheet
  - each stylesheet produces the same application-oriented markup language
- no reification of the result tree is required



Three-tiered architectures

To support a legacy of user agents that do not support XML:

- web servers can detect the level of support of user agents
- where XML and XSLT are not supported in a user agent:
  - the host can take on the burden of transformation
- where XML and XSLT are supported in a user agent
  - the burden of transformation can be distributed to the agent
  - the XML information can be massaged before being sent to the agent
- allows information to be maintained in XML yet still be available to all users



Three-tiered architectures (cont.)

Always performing server-side transformation:

- good business sense in some cases
  - even if technically it is possible to send semantically-rich information
- never send unprocessed semantically-rich XML
  - or only send it to those who are entitled to it
    - for security reasons
    - for payment reasons
- translation into a presentation-orientation
  - using a markup language inherently supported by the user agent (e.g. HTML)
  - using a custom, semantic-less markup language with an associated stylesheet
- "semantic firewall"
  - to protect the investment in rich markup from being seen where not desired
  - no consensus in the community that semantic firewalls are a "good thing"

Chapter 2 - Getting started with XSLT and XPath

- Introduction - Analyzing a simple stylesheet
- Section 1 - Stylesheet examples
- Section 2 - Syntax basics - stylesheets, templates, instructions
- Section 3 - More stylesheet examples

Outcomes:

- analyze the different components of a few example scripts
- introduce the concepts of instructions and templates

Analyzing a simple stylesheet

A few simple stylesheets:

- using Saxon
- using Internet Explorer 5 or greater
    - for IE5, the updated MSXML processor (at least the third Web Release of March 2000) is needed to support the W3C XSLT 1.0 Recommendation
    - the IE6 production release supports the W3C XSLT 1.0 Recommendation

Dissect example stylesheets

- identify stylesheet components as an introduction to basic concepts covered in more detail in the later chapters

Some simple examples

Consider the following XML file `hello.xml` obtained from the XML 1.0 Recommendation and modified to declare an associated stylesheet:

```
01 <?xml version="1.0"?>
02 <?xml-stylesheet type="text/xsl" href="hello.xsl"?>
03 <greeting>Hello world.</greeting>
```

Consider the following XSLT file `hellohtm.xsl` to produce HTML, noting how much it looks like an HTML document yet contains XSLT instructions:

```
01 <?xml version="1.0"?><!--hellohtm.xsl-->
02 <!--XSLT 1.0 - http://www.CraneSoftwrights.com/training -->
03 <html xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
04       xsl:version="1.0">
05  <head><title>Greeting</title></head>
06  <body><p>Words of greeting:<br/>
07    <b><i><u><xsl:value-of select="greeting"/></u></i></b>
08    </p></body>
09 </html>
```

Using an MSDOS command line invocation to execute the stand-alone processor explicitly with a supplied stylesheet, we see the following result:

```
01 C:\xslt\samp>saxon -o hellohtm.htm hello.xml hellohtm.xsl
02
03 C:\xslt\samp>type hellohtm.htm
04 <html>
05  <head>
06   <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
07   <title>Greeting</title>
08  </head>
09  <body>
10   <p>Words of greeting:<br><b><i><u>Hello world.</u></i></b></p>
11  </body>
12 </html>
13 C:\xslt\samp>
```

Note how the end result contains a mixture of the stylesheet markup and the source instance content, without any use of the XSLT vocabulary. The processor has recognized the use of HTML by the type of the document element and has engaged SGML markup conventions.

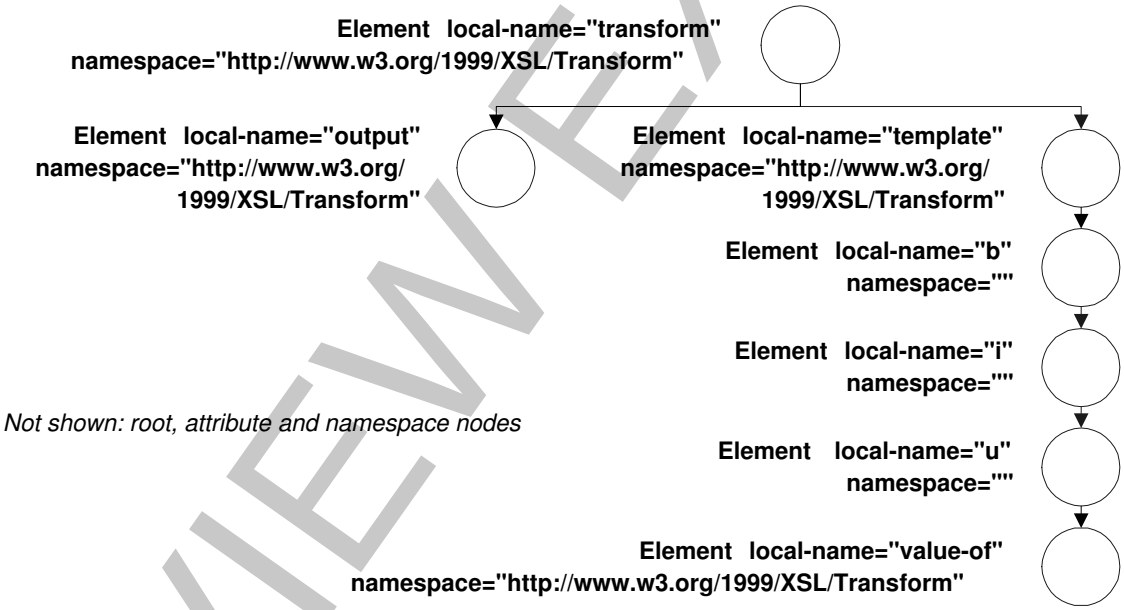
The `<meta>` element on line 6 added by Saxon is ensuring the character set of the web page is properly recognized by conforming user agents.

Some simple examples (cont.)

Consider next the following XSLT file `hello.xsl` to produce XML output using the HTML vocabulary, where the output is serialized as XML:

```
01 <?xml version="1.0"?><!--hello.xsl-->
02 <!--XSLT 1.0 - http://www.CraneSoftwrights.com/training -->
03
04 <xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
05                version="1.0">
06
07 <xsl:output method="xml" omit-xml-declaration="yes"/>
08
09 <xsl:template match="/">
10     <b><i><u><xsl:value-of select="greeting"/></u></i></b>
11 </xsl:template>
12
13 </xsl:transform>
```

Remember that the syntax of the stylesheet does not represent the syntax of the result, only the nodes of the result; the following is the node tree (not showing attribute and namespace nodes) of the stylesheet:

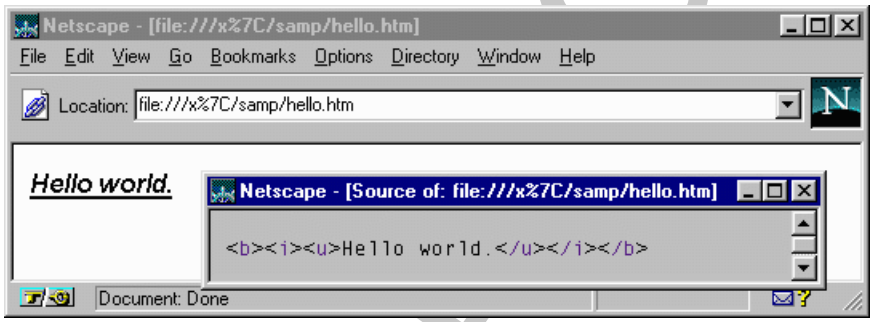


Some simple examples (cont.)

Using an MSDOS command line invocation to execute the Saxon processor we see the following result:

```
01 C:\xslt\samp>type hello.xml
02 <?xml version="1.0"?>
03 <?xml-stylesheet type="text/xsl" href="hello.xsl"?>
04 <greeting>Hello world.</greeting>
05
06 C:\xslt\samp>saxon -o hello.htm -a hello.xml
07
08 C:\xslt\samp>type hello.htm
09 <b><i><u>Hello world.</u></i></b>
10 C:\xslt\samp>
```

- the produced `hello.htm` file can be viewed with a browser to see the results using the menu selection View/Source to examine the contents of the document:



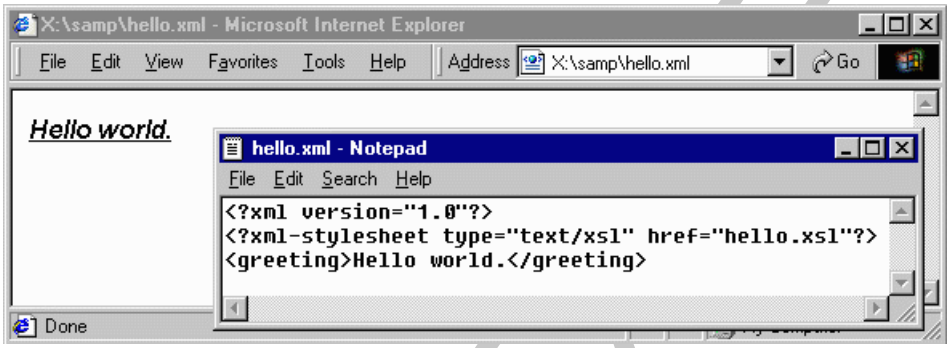
Some simple examples (cont.)

Using the `msxml.bat` invocation batch file (described in Invoking the Microsoft MSXML processor (page 134)) at an MSDOS command line to execute the MSXML processor:

```
C:\xslt\samp>msxml hello.xml hello.xsl hello-ms.htm

C:\xslt\samp>type hello-ms.htm
<b><i><u>Hello world.</u></i></b>
C:\xslt\samp>
```

Using IE to directly view the file will show the interpreted result on the browser canvas, in such a way that the menu function View/Source reveals the untouched XML:



Stylesheet requirements

An XSLT stylesheet:

- must identify the namespace prefix with which to recognize the XSLT vocabulary
  - a typical namespace declaration attribute declares a particular URI for a given prefix:
    - `xmlns:`*`prefix`*`="http://www.w3.org/1999/XSL/Transform"`
    - as a common practice the prefix "`xsl`" is used to identify the XSLT vocabulary, though this is not mandatory
    - the default namespace should not be used to identify the XSLT vocabulary
      - technically possible for elements of the vocabulary, but doing so prevents XSLT vocabulary attributes to be used wherever possible
      - not an issue for small stylesheets, but a maintenance headache if a large stylesheet needs to begin using XSLT attributes
  - extensions beyond the XSLT recommendation are outside the scope of the XSLT vocabulary so must use another URI for such constructs
- must also indicate the version of XSLT required by the stylesheet
  - in the start tag of an a element in the XSLT namespace
    - `version="`*`version-number`*`"`
    - attributes not in any namespace that are attached to an element in the XSLT namespace are regarded as being in the XSLT namespace
  - in the start tag of an element not in the XSLT namespace
    - *`prefix`*`:version="`*`version-number`*`"`

Instructions and literal result elements

An XSLT instruction:

- stylesheet file only
  - recognized in the stylesheet tree and not in the source tree
  - instruction defined by the XSLT recommendation and specified using the prefix associated with the XSLT URI
- is a control construct
  - the wrapper and top-level elements
  - procedural and process-control instructions
  - logical and physical stylesheet maintenance facilities
- is a value placeholder
  - any calculated value is replaced in-situ
  - `<xsl:value-of select="greeting"/>`
  - this example instruction calculates the concatenation of all text portions of all descendents of the first of the selected points in the source tree
  - the `select=` attribute is an expression specifying the point in the source tree or, more generally, the outcome of an arbitrary expression evaluation which in this case is a node set
  - the value "`greeting`" indicates the name of a direct element child node of the current source tree focus, which at the time of execution in this example is the root of the document (hence `<greeting>` must be the document element)
- may be custom extension
  - a non-standardized instruction implemented by the XSLT processor
  - implements extensibility
    - standardized fallback features allow any conforming XSLT processor to still interpret a stylesheet that is using extensions
  - specified using a namespace prefix associated with a URI known to the processor

A literal result element:

- any element not recognized to be an instruction
  - used in stylesheet file
  - any vocabulary that isn't a declared instruction vocabulary
- represents a result tree node
  - element and its associated attributes are to be added to the result

Templates and template rules

An XSLT template:

- specifies a fragment of the result tree as a tree of nodes
  - see the nodes in Some simple examples (page 40)
- expressed in syntax as a well-formed package of markup
  - may or may not include XSLT instructions

```
01 <b><i><u><xsl:value-of select="greeting"/></u></i></b>
```

- a representation of the desired nodes to add to the result tree
- the XSLT processor recognizes any constructs therein from the XSLT vocabulary as XSLT instructions and acts on them
  - regards all other constructs not from the XSLT vocabulary as literal result elements that comprise a representation of a tree fragment to add to the result tree

An XSLT template rule:

- a result tree construction rule associated with source tree nodes
  - specifies the template to add to the result tree when processing a source tree node
  - a "matching pattern" describes the nodes of the source tree

```
01 <xsl:template match="/">
02     <b><i><u><xsl:value-of select="greeting"/></u></i></b>
03 </xsl:template>
```

- prepares the XSLT processor for building a portion of the source tree for whenever the stylesheet asks the processor to visit the given source tree node
- uses the `match=` attribute as a "pattern" describing the characteristics of the source tree node associated with the given template
  - the pattern value "`/`" indicates the root of the source document (distinct from and the hierarchical parent of the document element of the source document, therefore, the very top of the hierarchy)
- a composite stylesheet must declare all the stylesheet writer's template rules to be used by the XSLT processor
- a simplified stylesheet defines in its entirety the one and only template rule for the stylesheet, that being for the root node

XSLT processor first visits the source tree root node:

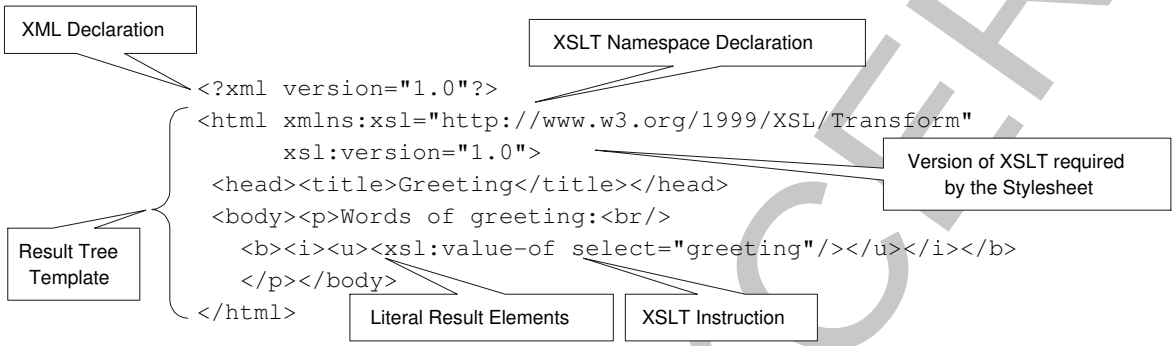
- the root node template rule is automatically processed
- *all* subsequent processing controlled by the stylesheet

Simplified stylesheets

Chapter 2 - Getting started with XSLT and XPath
Section 2 - Syntax basics - stylesheets, templates, instructions

A simplified declared XSLT stylesheet:

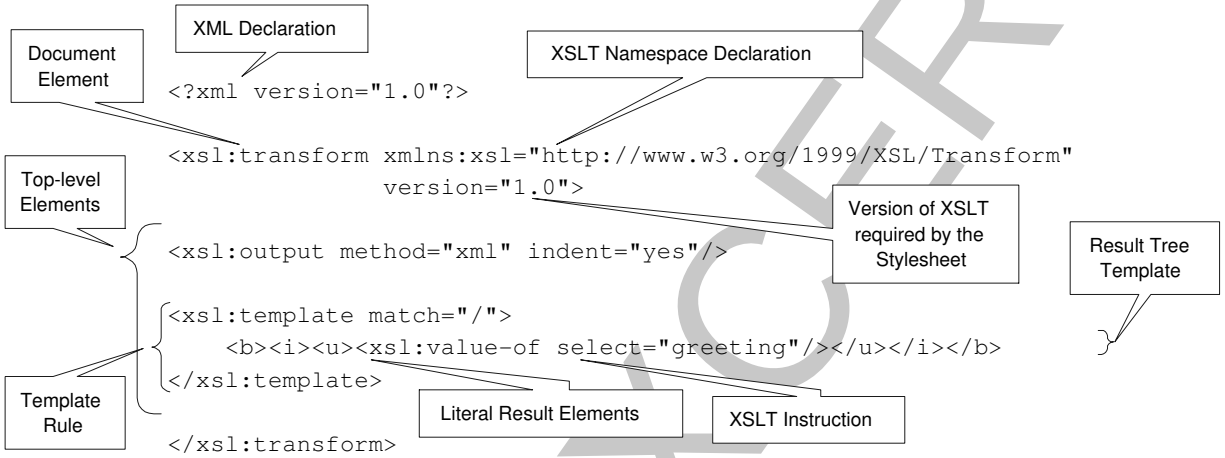
- can be declared inside an arbitrary XML document (e.g. an XHTML document) by using namespace declarations for XSLT constructs found within
- the entire stylesheet file is a template for the entire result tree
  - regarded as the template rule for the root node
- identifiable components of this implicitly declared XSLT script:



Composite stylesheets

A composite stylesheet:

- can be written as an entire XML document (or embedded fragment in an XML document) by using a stylesheet document element as the explicit container
- composite stylesheets can be utilized by other explicitly declared stylesheets
- identifiable components of this composite XSLT script:



Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

Approaches to stylesheet design

Pulling the input data

- known source structure
  - the hierarchy of the source file is known by the stylesheet writer
- the stylesheet "pulls" information:
  - from known locations in the source node tree
  - `<xsl:value-of select="`*XPath-expression*`"/>`
    - for extraction or calculation
  - `<xsl:copy-of select="`*XPath-expression*`"/>`
    - for wholesale copying of source tree nodes
  - `<xsl:for-each select="`*XPath-node-set-expression*`">`
    - for iteration over a collection of nodes
- stylesheet-determined result order
  - implements "stylesheet-driven" document processing
    - the result tree is built by the stylesheet obtaining each result component from the source, in result order, and framing each component as required with literal markup from the stylesheet
    - if the result can be described as a single template using only the "pull" instructions, the stylesheet can be simply declared as a single result template

For example

- consider a mixture of `<sale>` and `<purchase>` elements present in an arbitrary order
- to process all of the sales records together, they are pulled from the source tree:

```
<xsl:for-each select="//sale">
  ...template for the result tree for the sale...
</xsl:for-each>
```

- to process all of the purchase records together, they are pulled from the source tree:

```
<xsl:for-each select="//purchase">
  ...template for the result tree for the purchase...
</xsl:for-each>
```


Approaches to stylesheet design (cont.)

Pushing the input data

- arbitrary or unexpected source structure
  - the structure of the source file is not in either an expected or explicit order
  - source file order must be accommodated by stylesheet
- the stylesheet "pushes" information:
  - visits known (using names) or unknown (using a wild card) source tree nodes
  - `<xsl:apply-templates select="`*XPath-node-expression*`">`
- template rules prepare for source tree node visitations:
  - `<xsl:template match="`*XPath-pattern*`">`
- source-determined result order
  - implements "event-driven" document processing
    - the `<xsl:template>` template rule is the event handler
      - the type of event described as a qualification of the source information in the `match=` attribute
    - the `<xsl:apply-templates>` instruction is the event generator
      - selects the source information the XSLT processor is to visit

For example

- consider a mixture of `<sale>` and `<purchase>` elements present in an arbitrary order
- to process all of the records in the order they appear in the document, they are pushed through the stylesheet by specifying the union of all such nodes:

```
<xsl:apply-templates select="//sale |
                             //purchase"/>
```

- each construct being pushed must somehow be handled by the stylesheet:

```
<xsl:template match="sale">
  ...template for the result tree for the sale...
</xsl:template>
<xsl:template match="purchase">
  ...template for the result tree for the purchase...
</xsl:template>
```

Note it is not necessary to exclusively use one approach or the other

- stylesheets alternately push some of the input data through the XSLT processor (data driven) and pull the same or other input data where required (stylesheet driven)
- the pulling of data that is relative to the data being pushed can be done in the template catching the data being pushed

Processing XML data with multiple XSLT stylesheets

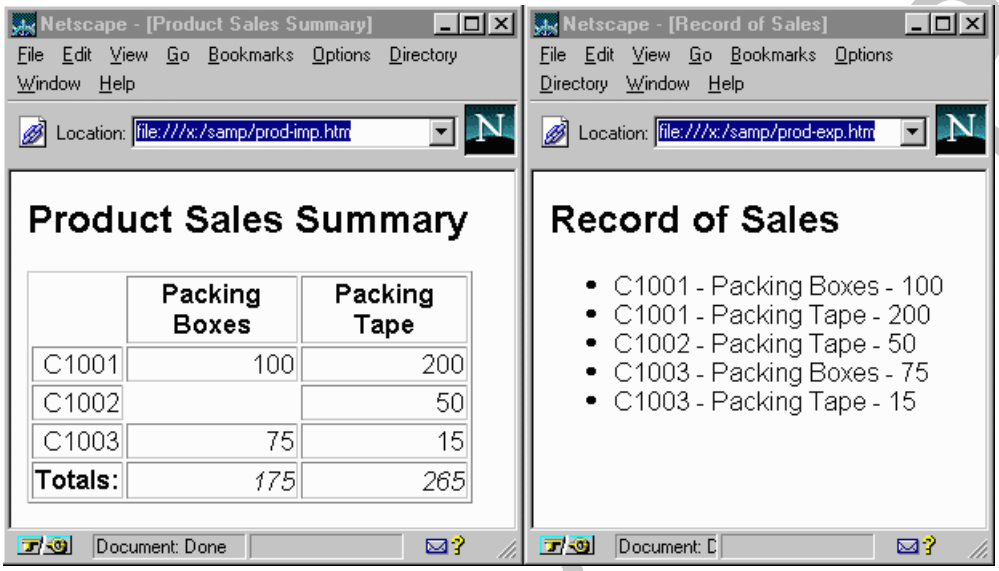
Consider the data file `prod.xml` containing some sales data information:

```
<?xml version="1.0"?><!--prod.xml-->
<!DOCTYPE sales [

<!ELEMENT sales ( products, record )> <!--sales information-->
<!ELEMENT products ( product+ )>         <!--product record-->
<!ELEMENT product ( #PCDATA )>      <!--product information-->
<!ATTLIST product id ID #REQUIRED>
<!ELEMENT record ( cust+ )>                <!--sales record-->
<!ELEMENT cust ( prodsale+ )>     <!--customer sales record-->
<!ATTLIST cust num CDATA #REQUIRED>     <!--customer number-->
<!ELEMENT prodsale ( #PCDATA )>     <!--product sale record-->
<!ATTLIST prodsale idref IDREF #REQUIRED>
]>
<sales>
  <products><product id="p1">Packing Boxes</product>
            <product id="p2">Packing Tape</product></products>
  <record><cust num="C1001">
            <prodsale idref="p1">100</prodsale>
            <prodsale idref="p2">200</prodsale></cust>
          <cust num="C1002">
            <prodsale idref="p2">50</prodsale></cust>
          <cust num="C1003">
            <prodsale idref="p1">75</prodsale>
            <prodsale idref="p2">15</prodsale></cust></record>
</sales>
```


Processing XML data with multiple XSLT stylesheets (cont.)

Very dissimilar reports could be generated for the one data file by using different stylesheets:

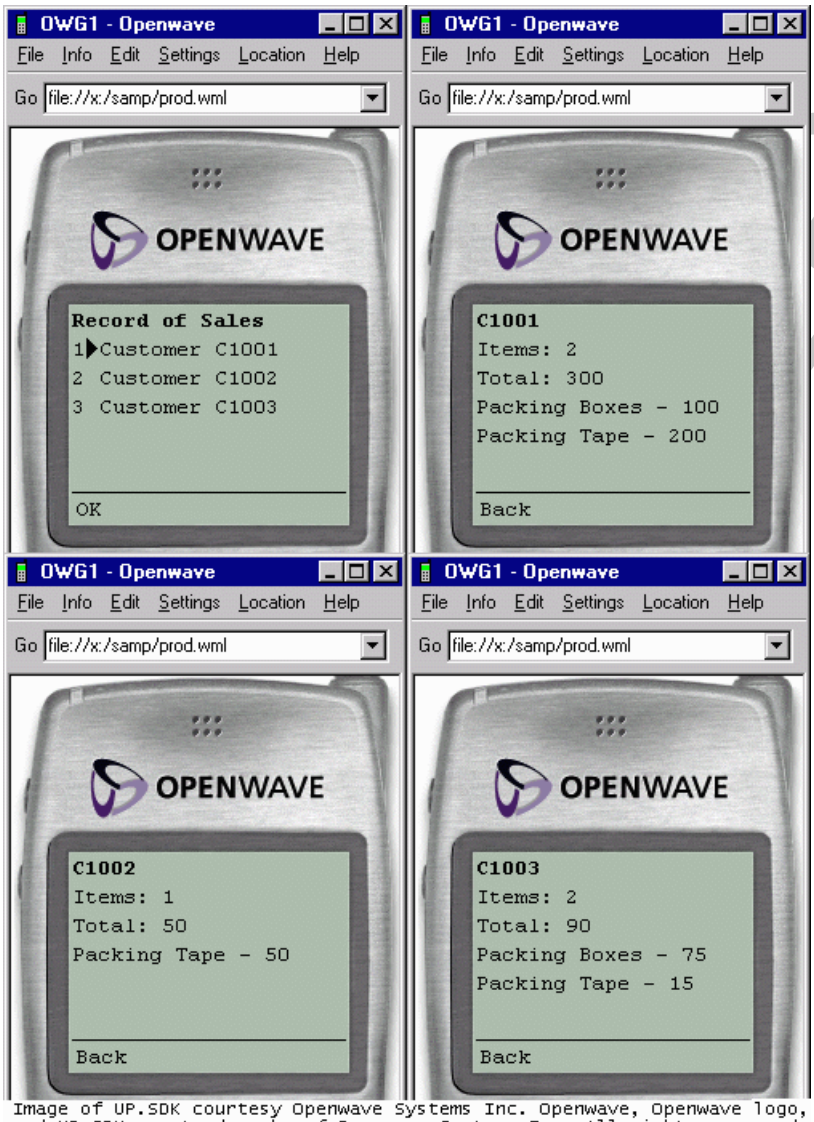


Of note:

- items are rearranged from one authored order to two different presentation orders
- transformation includes calculation of sum of marked up values

Processing XML data with multiple XSLT stylesheets (cont.)

Any result vocabulary can be used; for example, WML rendered on a mobile device:



Processing XML data with multiple XSLT stylesheets (cont.)

The simplified stylesheet `prod-sim.xsl` for the table (page 51) for the XML (page 50):

```
<?xml version="1.0"?><!--prod-sim.xsl-->
<!--XSLT 1.0 - http://www.CraneSoftwrights.com/training -->
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xsl:version="1.0">
 <head><title>Product Sales Summary</title></head>
 <body><h2>Product Sales Summary</h2>
  <table summary="Product Sales Summary" border="1">
                                        <!--list products-->
   <tr align="center"><th/>
    <xsl:for-each select="//product">
     <th><b><xsl:value-of select="."/></b></th>
    </xsl:for-each></tr>
                                       <!--list customers-->
   <xsl:for-each select="/sales/record/cust">
    <xsl:variable name="customer" select="."/>
    <tr align="right">
     <td><xsl:value-of select="@num"/></td>
     <xsl:for-each select="//product">  <!--each product-->
      <td><xsl:value-of select="$customer/prodsale
                                [@idref=current()/@id]"/>

      </td></xsl:for-each>
    </tr></xsl:for-each>
                                            <!--summarize-->
   <tr align="right"><td><b>Totals:</b></td>
    <xsl:for-each select="//product">
     <xsl:variable name="pid" select="@id"/>
     <td><i><xsl:value-of
                 select="sum(//prodsale[@idref=$pid])"/></i>
     </td></xsl:for-each></tr>
  </table>
 </body></html>
```

Information from the stylesheet and pulled from the source document:

- the header and body title are hardwired content from stylesheet
- the table's header row comes from each product name in source
- the customer information is iterated over to produce rows (note use of variable)
  - sale information produces columns
- total information is generated by stylesheet using `sum()` built-in function (no custom node-traversal programming needed)

Processing XML data with multiple XSLT stylesheets (cont.)

The composite stylesheet `prod-com.xsl` for the list (page 51) for the XML (page 50):

```
<?xml version="1.0"?><!--prod-com.xsl-->
<!--XSLT 1.0 - http://www.CraneSoftwrights.com/training -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<xsl:template match="/">                              <!--root rule-->
  <html><head><title>Record of Sales</title></head>
    <body><h2>Record of Sales</h2>
      <xsl:apply-templates select="/sales/record"/>
    </body></html></xsl:template>

<xsl:template match="record">   <!--processing for each record-->
  <ul><xsl:apply-templates/></ul></xsl:template>

<xsl:template match="prodsale">   <!--processing for each sale-->
  <li><xsl:value-of select="../@num"/>   <!--use parent's attr-->
      <xsl:text> - </xsl:text>
      <xsl:value-of select="id(@idref)"/>      <!--go indirect-->
      <xsl:text> - </xsl:text>
      <xsl:value-of select="."/></li></xsl:template>


</xsl:stylesheet>
```

Source document is pushed through the stylesheet:

- the header and body title are hardwired content from stylesheet
- the root rule pushes all sales records through the stylesheet
- each record produces the `<ul>` unordered list wrapper for list items and pushes child elements through the stylesheet
- each child element pushed through produces a list item that pulls information from the parent and from an arbitrary place of the source

Processing XML data with multiple XSLT stylesheets (cont.)

The composite stylesheet `prod-wml.xsl` for the WML (page 52) for the XML (page 50):

```
<?xml version="1.0"?><!--prod-wml.xsl-->
<!--XSLT 1.0 - http://www.CraneSoftwrights.com/training -->
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output doctype-system="http://CRANE/wml13.dtd"/>

<xsl:template match="/">             <!--root rule-->
  <wml><card title="Record of Sales"> <!--index card-->
        <p><em>Record of Sales</em></p>
        <p><select name="cards">
             <xsl:apply-templates mode="head"
                  select="/sales/record/cust"/>
           </select></p></card>
   <xsl:apply-templates select="/sales/record/cust"/>
  </wml></xsl:template>

<xsl:template match="cust" mode="head"><!--index entry-->
  <option onpick="#{@num}">
    <xsl:text/>Customer <xsl:value-of select="@num"/>
  </option></xsl:template>

<xsl:template match="cust"><!--customer's card in deck-->
 <card id="{@num}" title="Customer {@num}">
  <p><em><xsl:value-of select="@num"/></em></p>
  <p>Items: <xsl:value-of select="count(prodsale)"/></p>
  <p>Total: <xsl:value-of select="sum(prodsale)"/></p>
  <xsl:apply-templates/></card></xsl:template>

<xsl:template match="prodsale"><!--proc for each sale-->
  <p><xsl:value-of select="id(@idref)"/> <!--indirect-->
     <xsl:text> - </xsl:text>
     <xsl:value-of select="."/></p></xsl:template>


</xsl:stylesheet>
```

Source document is pushed through the stylesheet:

- the document element of the result is `<wml>`
- the same source is visited twice using different template rules for processing to produce different results

Chapter 3 - XPath data model

- Introduction - The need for abstractions

Dealing with information, not markup

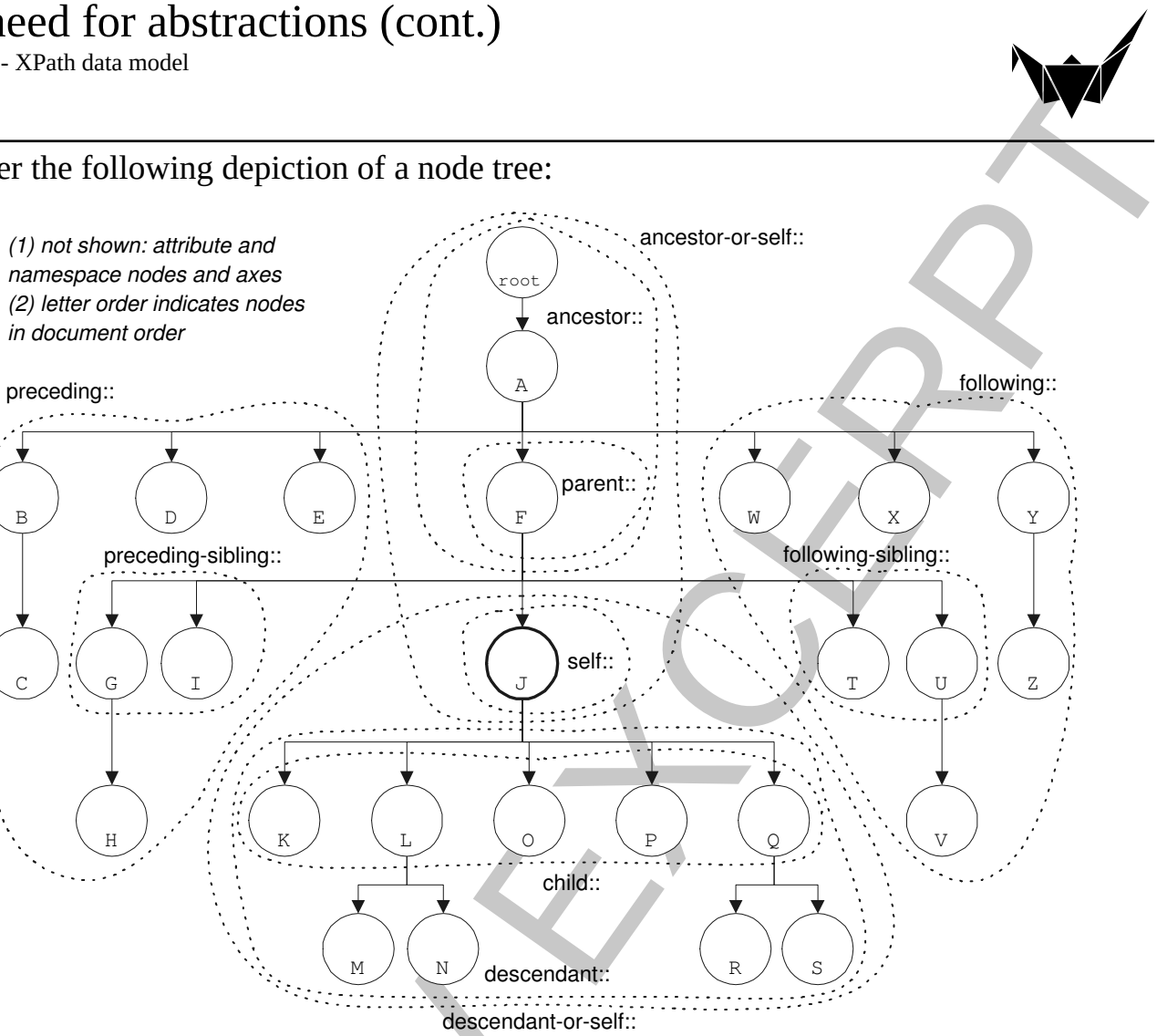
- all input and output information manipulated in an abstract fashion
  - minimum of three information structures: stylesheet, main source, result
    - multiple source documents may be read as separate structures
  - the result structure is constructed from the structures created from the stylesheet document and all source documents
- knowledge of input markup and control of output markup out of the hands of the stylesheet writer
  - the writer deals with nodes of information, not characters of markup

Traversing a source document or stylesheet document predictably

- interpreted as an abstract tree of nodes
  - the contents of the documents are described according to a data model for the XML markup
    - will be expressed in terms of a future version of the XML Information Set
  - all nodes created are typed, and have a value that can be used as a string of text
  - some nodes have an associated name, while other nodes are unnamed
  - the XSLT processor performs all operations using the node trees, not using the documents directly
    - the actual markup used in input instances is not preserved
    - there are no constraints or requirements of the XML source in that any well-formed markup chosen by the author of an XML document is represented abstractly in the tree
- XPath node tree navigation
  - the stylesheet can navigate around the source node tree in many directions
  - thirteen axes of direction that can be traversed relative to the context (current) node
  - the stylesheet is responsible for specifying which source nodes get processed when and how

The need for abstractions (cont.)

Chapter 3 - XPath data model



Consider the following depiction of a node tree:

- the root of the tree is at the top
- the leaves of the tree are towards the bottom
- the context node in this example is in the center with the bold circle
- the dotted lines completely surround the nodes of the tree that are members of each axis relative to the context node

The need for abstractions (cont.)

Building a result predictably

- created as an abstract tree of nodes
  - the content of the desired result is described according to the same data model for XML markup as is used for input
  - the processing of a template builds a portion of the result as sub-tree of nodes reflecting the output information
  - the interpretation of a result template must be reliable and reproducible
    - the XSLT processor acts on instructions the same way every time
- serialization instantiates markup syntax
  - the emission of the result node tree (if so desired)
  - in XML markup according to the standard
    - the stylesheet writer does `not` control which markup constructs are used for representing XML information
    - the XSLT processor can make arbitrary decisions as long as the end result is well formed
  - using alternative markup or syntax conventions if made available by the XSLT processor (e.g.: recognition of the desire to emit HTML 4.0 markup)

The need for abstractions (cont.)

The XSLT instructions covered in this chapter are:

- `<xsl:strip-space>`
  - Indicate those source tree nodes in which white-space-only text nodes are not to be preserved.
- `<xsl:preserve-space>`
  - Indicate those source tree nodes in which white-space-only text nodes are to be preserved

The functions covered in this chapter are:

- `last()`
  - the number of nodes in the current node list
- `position()`
  - the number of the current node in the current node list

Chapter 4 - XSLT processing model

- Introduction - A predictable behavior for XSLT processors

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

A predictable behavior for XSLT processors

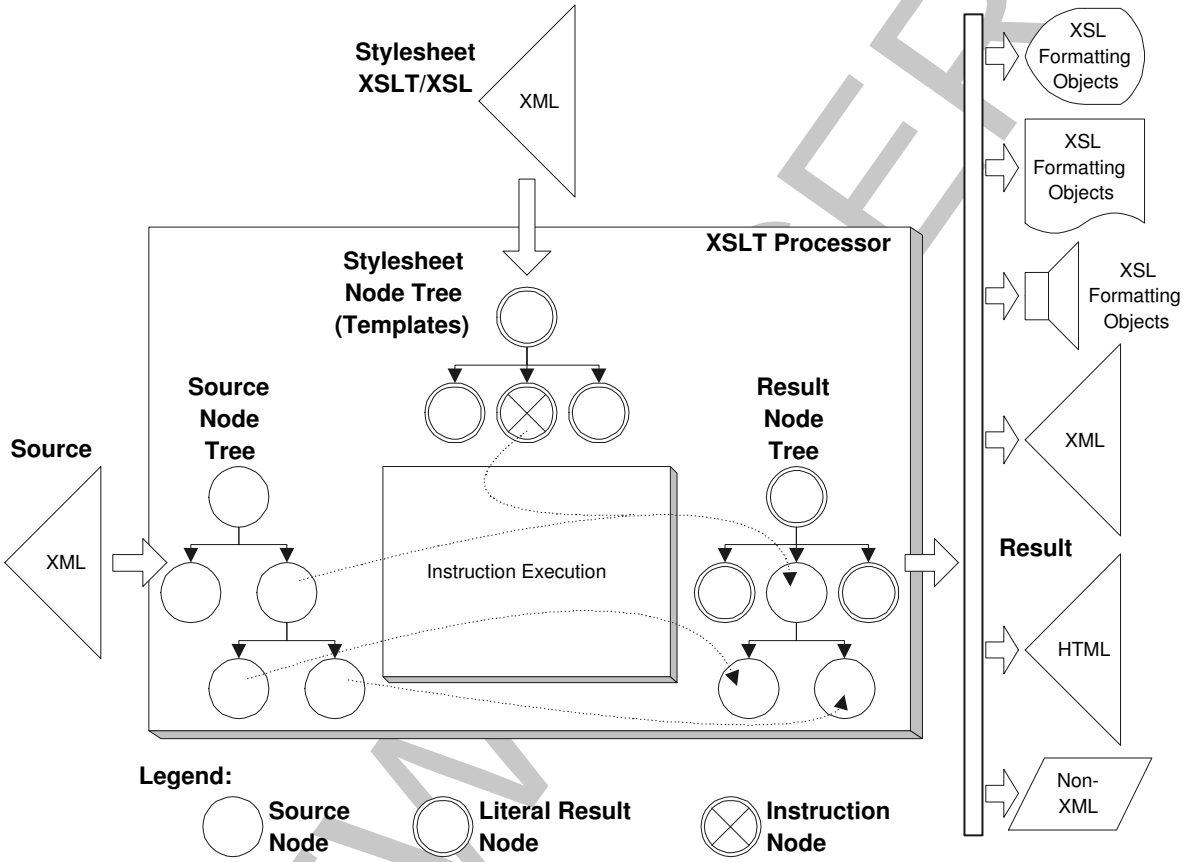
XSLT is designed to ensure predictability

- predictable processing behavior every time
  - all aspects of the processing are well-defined
- processor sees stylesheet as a tree of nodes
  - some nodes of which are from the XSLT instruction vocabulary
  - some nodes of which may be from an extension instruction vocabulary
  - the remainder of which are literal result elements that are to be used in the result
- processor sees the primary source file as a tree of nodes
  - the markup of the XML source document is not material
  - the vocabulary used in the source is not material to the XSLT processor
- processing starts with the template rule for the root node
  - the only action presumed by the XSLT processor is that the template for the root node be found and added to the result tree
- the stylesheet shapes the content of the result tree
  - the stylesheet writer must plan the flow of the stylesheet process according to the document order of the result
  - other source trees are created from other source files on request by the stylesheet
  - components from the source trees are obtained where required when executing instructions found in the stylesheet
  - once a portion of the result tree is completely generated there is no method of returning to modify the result tree in any way
- the result tree of nodes may be serialized
  - interpreted XSL formatting objects (e.g.: display, print, aural, etc.)
  - XML markup
  - HTML markup
  - simple text
  - syntax conventions recognized by the particular implementation of the processor (binary or text)
  - remember the XSLT processor is *not* required to support any particular `method=` and may choose to output the tree as XML only if at all

A predictable behavior for XSLT processors (cont.)

A simple illustration of the basic process

- the stylesheet includes a single instruction that obtains information from a particular point in the source tree
- not shown in this illustration are built in instructions and other templates copying the child nodes of the target point in the source tree



A predictable behavior for XSLT processors (cont.)

The XSLT instructions covered in this chapter are as follows.

Instructions to pull information from the source tree or to calculate values:

- `<xsl:value-of>`
  - add to the result tree the evaluation of an expression or the value of a source tree node
- `<xsl:copy-of>`
  - add to the result tree a copy of nodes from the source tree
- `<xsl:for-each>`
  - iterate over a selection of source tree nodes using a supplied template

Instructions to push information from the source tree through the stylesheet:

- `<xsl:apply-templates>`
  - supply a selection of source tree nodes to push through template rules
- `<xsl:template>`
  - define a template rule

Chapter 5 - The XSLT transformation environment

- Introduction - The XSLT transformation environment

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

The XSLT transformation environment

Different ways are available to communicate to and from an XSLT processor

- some aspects of transformation are under stylesheet control
- others cannot be manipulated under stylesheet control

The XSLT instructions covered in this chapter are as follows.

Wrapping the content of a stylesheet:

- `<xsl:stylesheet>`
    - encapsulate a stylesheet specification
- `<xsl:transform>`
    - encapsulate a stylesheet specification

Serializing the result tree:

- `<xsl:namespace-alias>`
    - specify a result tree namespace translation
- `<xsl:output>`
    - specify the desired serialization of the result tree

Communicating with the operator:

- `<xsl:message>`
    - report a stylesheet condition to the operator
- `<xsl:param>`
    - supply a parameterized value from the operator

The functions covered in this chapter are as follows.

Environment functions:

- `system-property()`
    - accessing system-defined property strings

Chapter 6 - XSLT stylesheet management

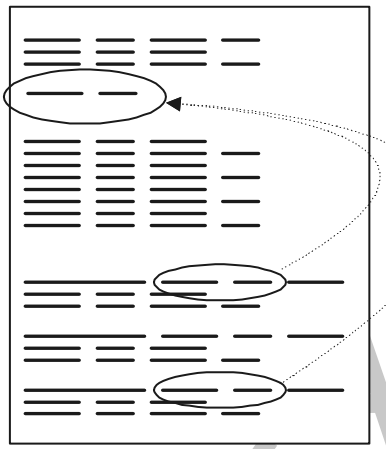
- Introduction - Why modularize logical and physical structure of stylesheets?

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

Why modularize logical and physical structure of stylesheets?

Modularizing the logical structure supports development

- manipulating or reusing stylesheet fragments within a given stylesheet
- declaration and reuse of syntactic packages of stylesheet markup
- parameterization of a template
- writing a template once and using it many places
- defining a value and referencing it many places



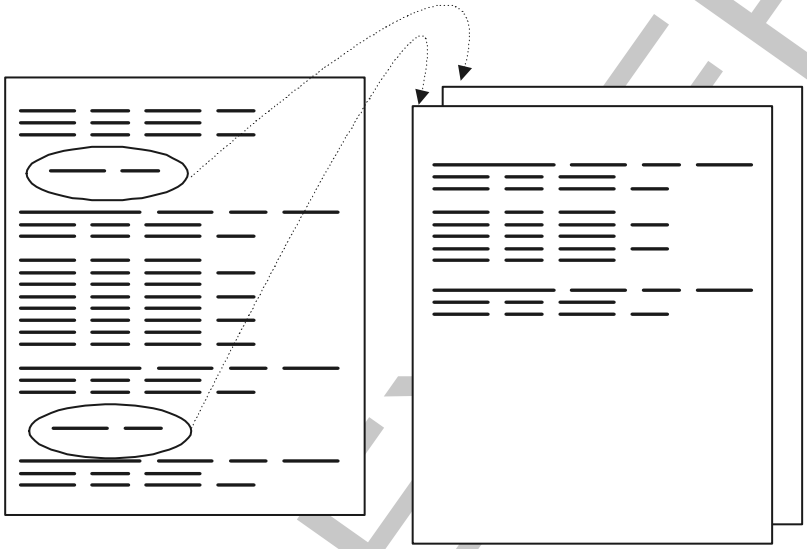
This chapter overviews logical modularization using:

- XML internal general entities
- XML internal general entities in marked sections in external parameter entities
- XSLT variable bindings
- XSLT named templates

Why modularize logical and physical structure of stylesheets? (cont.)

Modularizing the physical entity structure supports reuse

- compartmentalization of code
- sharing and reuse of stylesheet fragments across an organization
- support for organizational rules for source code control and management
- access to any built-in custom extension function
    - if available in the XSLT processor implementation



This chapter overviews physical modularization using:

- XML external parsed general entities
- XSLT included files
- XSLT imported files
- XSLT extension functions
- XSLT extension elements (instructions)

Why modularize logical and physical structure of stylesheets? (cont.)

The XSLT instructions covered in this chapter are as follows.

Instructions related to logical modularization:

- `<xsl:call-template>`
  - process a stand-alone template on demand
- `<xsl:template>`
  - declare a template to be called by name
- `<xsl:variable>`
  - declare a non-parameterized variable and its bound value
- `<xsl:param>`
  - declare a parameterized variable and its default bound value
- `<xsl:with-param>`
  - specify a binding value for a parameterized variable

Instructions related to physical modularization:

- `<xsl:include>`
  - include a stylesheet without overriding stylesheet constructs
- `<xsl:import>`
  - import a stylesheet while overriding stylesheet constructs
- `<xsl:apply-imports>`
  - bypass the importation of template rules
- `<xsl:fallback>`
  - accommodate the lack of implementation of an instruction element

The functions covered in this chapter are as follows.

Availability functions:

- `element-available()`
  - determine the availability of an instruction element
- `function-available()`
  - determine the availability of a function

Chapter 7 - XSLT process control and result tree instructions

- Introduction - Managing the creation of result nodes

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

Managing the creation of result nodes

Source tree not always processed one-to-one for the result:

- may need to skip source nodes
- may need to select between alternative results based on source
    - arbitrary algorithm depending on the content
- may need to reflect the positions of source tree nodes in different contexts
- may need to create arbitrary result tree nodes
    - independent of the source tree nodes

XSLT supports:

- selective template selection
- numbering of information found in the source tree
- synthesis of nodes and text in the result tree

Managing the creation of result nodes (cont.)

The XSLT instructions covered in this chapter are as follows.

Instructions related to process control:

- `<xsl:if>`
  - single-state conditional inclusion of a template
- `<xsl:choose>`
  - multiple-state conditional inclusion of one of a number of templates
- `<xsl:when>`
  - single-state conditional inclusion of a template within a multiple-state condition
- `<xsl:otherwise>`
  - default-state conditional inclusion of a template within a multiple-state condition

Instructions related to building the result tree:

- `<xsl:number>`
  - add a string to the result tree representing the position of the current node
- `<xsl:attribute>`
  - instantiate an attribute node in the result tree
- `<xsl:attribute-set>`
  - declare a set of attribute nodes for use in the result tree
- `<xsl:comment>`
  - instantiate a comment node in the result tree
- `<xsl:element>`
  - instantiate an element node in the result tree
- `<xsl:processing-instruction>`
  - instantiate a processing instruction node in the result tree
- `<xsl:text>`
  - instantiate a text node in the result tree
- `<xsl:copy>`
  - instantiate a copy of the current node in the result tree
- `<xsl:copy-of>`
  - instantiate a complete copy of a specified node in the result tree

Chapter 8 - XPath and XSLT expressions and advanced techniques

- Introduction - XPath and XSLT functions and advanced techniques

XPath and XSLT functions and advanced techniques

Powerful functions and expression support

- this chapter describes functions and expressions to manipulate variables and values of the XPath and XSLT data types
    - the XSLT processor implements the algorithms for publishing-oriented facilities so that the stylesheet writer doesn't have to
- value manipulation
    - boolean functions and operators
    - number functions and operators
    - string functions
    - node set functions and operator
- access to the source node tree
    - `id()`
        - returns nodes with unique identifiers
    - `key()`
        - returns nodes that have stylesheet-declared relationships
    - `current()`
        - returns the current node of the start of expression evaluation
- access to document model information
    - `unparsed-entity-uri()`
        - returns URI associated with declared entity
- access to multiple node trees
    - `document()`
        - returns root nodes of source node trees from external files

Advanced techniques

- this chapter also describes an approach to walking the source node tree in search of information in such a way that is impossible through available pattern matching techniques.

XPath and XSLT functions and advanced techniques (cont.)

The XSLT instructions covered in this chapter are as follows.

Instruction related to string formatting:

- `<xsl:decimal-format>`
  - control the formatting of numbers when added to the result tree

Instruction related to advanced access to the source node tree:

- `<xsl:key>`
  - declare key nodes in the source tree node for bulk processing

Instruction related to advanced algorithmic techniques:

- `<xsl:call-template>`
  - use named templates with subroutine-like control

The functions covered in this chapter are as follows.

Functions related to boolean data types:

- `boolean()`
  - casting an argument to a boolean value
- `false()`
  - a fixed boolean value
- `lang()`
  - finding the in-scope language as specified by `xml:lang=`
- `not()`
  - inverting the boolean value of the argument
- `true()`
  - a fixed boolean value

XPath and XSLT functions and advanced techniques (cont.)

Functions related to number data types:

- `ceiling()`
  - rounding a number up
- `floor()`
  - rounding a number down
- `format-number()`
  - adding punctuation and controlling number display
- `number()`
  - casting an argument to a number
- `round()`
  - rounding a number
- `sum()`
  - summing the numeric values of nodes

Functions related to string data types:

- `concat()`
  - string concatenation
- `contains()`
  - string detection
- `normalize-space()`
  - normalizing extraneous spaces in a string
- `starts-with()`
  - establishing the presence of a string
- `string()`
  - casting an argument to a string
- `string-length()`
  - finding the length of a string
- `substring()`
  - returning a portion of a string
- `substring-after()`
  - returning a portion of a string
- `substring-before()`
  - returning a portion of a string
- `translate()`
  - translating characters found in a string

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

XPath and XSLT functions and advanced techniques (cont.)

Functions related to node data types:

- `count()`
  - node counting
- `generate-id()`
  - establishing uniqueness in source node trees
- `local-name()`
  - obtaining the local part of a node name
- `name()`
  - obtaining a node name
- `namespace-uri()`
  - obtaining the namespace URI for a node

Other functions:

- `current()`
  - current node access
- `document()`
  - access to multiple source documents
- `id()`
  - accessing ID values in source node trees
- `key()`
  - accessing key nodes from key tables
- `unparsed-entity-uri()`
  - finding the URI of an unparsed entity

Chapter 9 - Sorting and grouping

- Introduction - Sorting and grouping

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

Sorting and grouping

Chapter 9 - Sorting and grouping

Sorting

An important part of many transformations is the need to re-order the source tree nodes into a sorted order for processing into result tree nodes:

- designed for sorting the current node list using a single value for each node
  - value may be simple node value or may be any XPath evaluation relative to node
- multiple sort criteria when processing a set of nodes
  - items being sorted are selected from the tree and the selection itself is sorted, not the tree
- sorting can be language based, numeric based or based on custom semantics

This chapter covers how to specify sort criteria when writing XSLT stylesheets.

Grouping and uniqueness

Another important part of many transformations is the need to infer structure from the results of sorting information, which is a process often called "grouping":

- composite information that must be separated and grouped by a common component
  - selecting a single piece of composite information obtains all components
  - a simple sort doesn't partition the composite information into constituent pieces
- no explicit support yet to group together a sequence of duplicate source tree nodes
  - this is noted as a future consideration for XSLT
  - this is a common requirement that exists today in many information processing situations
- specific application of the generalized problem of finding unique values from a set
  - often necessary to find unique values in a set of values
  - unique values make up the group headings

This chapter covers two techniques of using XSLT to group constructs when processing:

- using reverse-document-order axes
- the "Muenchian Method" of using `<xsl:key>`

Sorting and grouping (cont.)

The XSLT instruction covered in this chapter is:

- `<xsl:sort>`
  - specify a criterion with which to sort a set of nodes

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

Annex A - XML to HTML transformation

- Introduction - Historical web standards for presentation

Historical web standards for presentation

Annex A - XML to HTML transformation

Recognizing that the purpose of many XSLT transformations will be to render information over the World Wide Web, it is important to understand what different user-agent technologies are currently available to be used:

- user agents do not inherently understand the presentation semantics associated with our custom XML vocabularies
- can translate instances of our vocabularies into instances of a user agent vocabulary (e.g. HTML)
- can annotate instances of our vocabularies with formatting properties recognized by a user agent (e.g. CSS)

Hypertext Markup Language (HTML)

- a language for sharing text and graphics
- a hyperlinking facility for relating information

Extensible Hypertext Markup Language (XHTML)

- modularization of HTML
- reformulating HTML as XML
- support of arbitrary XML in HTML

Cascading Stylesheets (CSS)

- getting away from the built-in user agent rendering semantics
- describes document tree ornamentation with formatting properties

User Agent Screen Painting

- direct control of the user agent canvas

This annex overviews considerations for producing different flavors of HTML to support different user agents. As well, stylesheet fragments illustrating common requirements to mark up images and links are described.

Issues of compatibility between different user agent implementations and recommended markup practices are not reviewed in this material. A discussion of such issues can be found at `http://www.w3.org/TR/xhtml1/#guidelines`.

Annex B - XSL formatting semantics introduction

- Introduction - Formatting objectives

Outcomes:

- awareness of the formatting objectives of the XSL development committee

Formatting objectives

Annex B - XSL formatting semantics introduction

The Extensible Stylesheet Language (XSL)

A catalogue of formatting objects and flow objects (each with properties controlling behavior) for rendering information to multiple media.

- addresses basic word-processing-level pagination
- semantic model for formatting
  - expressed in terms of which XSL concepts can be described
  - described as a vocabulary that can be serialized as XML markup

Sophisticated pagination and support for layout-driven documents

- DSSSL
  - Document Style Semantics and Specification Language ISO-10179
- W3C Common Formatting Model
  - effort initially based on CSS
- vocabulary accommodates both heritages
  - some constructs can be specified different ways with different names
  - writing-direction-independent (absolute) and writing-direction-dependent (writing mode relative) properties

Well-defined constructs

- express formatting intent
  - according to the XSL formatting model
- available to the stylesheet writer
  - for specification of a layout using the XSL formatting vocabulary
- managed and interpreted by the formatter
  - that process responsible for rendering
  - in response to a description of the layout

Formatting objectives (cont.)

Annex B - XSL formatting semantics introduction

Effecting the formatting of XML with XSL formatting semantics

- transformation stylesheet
  - it is the stylesheet writer's responsibility to write an XSLT transformation of the XML source file into a result node tree composed entirely of formatting and flow objects using the XSL vocabulary
- semantics interpretation
  - an XSL processor implementing XSL formatting semantics recognizes the vocabulary and renders the result
- unlike CSS
  - the user's vocabulary is not supplemented with formatting properties

Intermediate result of rendering

- the XSL processor may, but need not, emit the result node tree as XML markup
  - formatting and flow objects are XML elements
  - properties are attribute/value pairs specified in the XML elements
- very useful for debugging stylesheets

This chapter briefly introduces concepts and basic constructs used in the XSL-FO 1.0 Recommendation, without going into the details of the vocabulary or markup required to support these concepts. The topic of formatting objects and their semantics and markup warrants an entire tutorial on its own and is thus separate from this tutorial.

Annex C - Instruction, function and grammar summaries

- Introduction - Quick summaries
- Section 1 - Vocabulary and functions
- Section 2 - Grammars

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

Quick summaries

Annex C - Instruction, function and grammar summaries

The XSLT 1.0 Recommendation at `http://www.w3.org/TR/1999/REC-xslt-19991116` and XPath 1.0 Recommendation at `http://www.w3.org/TR/1999/REC-xpath-19991116` are excellent and rigorous references to the facilities and function of these two Recommendations.

This annex lists alphabetized references to the XSLT elements and the XSLT and XPath built-in functions. Each entry notes the chapter in this book where the construct is primarily described.

Also included is a summary of the grammars for patterns and string expressions from XSLT and XPath, rendered with cross-references not found in the recommendations themselves.

XSLT element summary

Annex C - Instruction, function and grammar summaries
Section 1 - Vocabulary and functions

All elements in the XSLT vocabulary in alphabetical order follow. Note that the Kleene operators '?', '*' and '+' (respectively zero or one, zero or more, and one or more) are used to denote the cardinality of attributes and contained constructs. The content model operators ',' and '|' (respectively sequence and alternation) are also used. The brace brackets '{' and '}' denote the use of an attribute value template. This information is mechanically derived from the XSLT 1.0 Recommendation.

`apply-imports` (instruction) - Why modularize logical and physical structure of stylesheets? (page 70)

- XSLT 5.6 Overriding Template Rules
- ```
  <xsl:apply-imports/>
  ```

`apply-templates` (instruction) - A predictable behavior for XSLT processors (page 64)

- XSLT 5.4 Applying Template Rules
- ```
  <xsl:apply-templates mode="qname"?
                       select="node-set-expression"?>
    (<xsl:sort>|<xsl:with-param>)*
  </xsl:apply-templates>
  ```

`attribute` (instruction) - Managing the creation of result nodes (page 73)

- XSLT 7.1 Creating Elements and Attributes
- ```
  <xsl:attribute name="qname|{string-expression}"
                 namespace="uri-reference|{string-expression}"?>
    template
  </xsl:attribute>
  ```

`attribute-set` (top level element) - Managing the creation of result nodes (page 73)

- XSLT 7.1 Creating Elements and Attributes
- ```
  <xsl:attribute-set name="qname"
                     use-attribute-sets="qnames"?>
    <xsl:attribute>*
  </xsl:attribute-set>
  ```

`call-template` (instruction) - Why modularize logical and physical structure of stylesheets? (page 70)

- XSLT 6 Named Templates
- ```
  <xsl:call-template name="qname">
    <xsl:with-param>*
  </xsl:call-template>
  ```

`choose` (instruction) - Managing the creation of result nodes (page 73)

- XSLT 9.2 Conditional Processing with `xsl:choose`
- ```
  <xsl:choose>
    (<xsl:when>+,<xsl:otherwise>?)
  </xsl:choose>
  ```

`comment` (instruction) - Managing the creation of result nodes (page 73)

- XSLT 7.4 Creating Comments
- ```
  <xsl:comment>
    template
  </xsl:comment>
  ```

`copy` (instruction) - Managing the creation of result nodes (page 73)

- XSLT 7.5 Copying
- ```
  <xsl:copy use-attribute-sets="qnames"?>
    template
  </xsl:copy>
  ```

`copy-of` (instruction) - Managing the creation of result nodes (page 73)

- XSLT 11.3 Using Values of Variables and Parameters with `xsl:copy-of`
- ```
  <xsl:copy-of select="expression"/>
  ```

`decimal-format` (top level element) - XPath and XSLT functions and advanced techniques (page 76)

- XSLT 12.3 Number Formatting
- ```
  <xsl:decimal-format decimal-separator="char"?
                      digit="char"?
                      grouping-separator="char"?
                      infinity="string"?
                      minus-sign="char"?
                      name="qname"?
                      NaN="string"?
                      pattern-separator="char"?
                      per-mille="char"?
                      percent="char"?
                      zero-digit="char"?/>
  ```

`element` (instruction) - Managing the creation of result nodes (page 73)

- XSLT 7.1 Creating Elements and Attributes
- ```
  <xsl:element name="qname|{string-expression}"
               namespace="uri-reference|{string-expression}"?
               use-attribute-sets="qnames"?>
    template
  </xsl:element>
  ```

`fallback` (instruction) - Why modularize logical and physical structure of stylesheets? (page 70)

- XSLT 15 Fallback
- ```
  <xsl:fallback>
    template
  </xsl:fallback>
  ```


`for-each` (instruction) - A predictable behavior for XSLT processors (page 64)

- XSLT 8 Repetition
- ```
  <xsl:for-each select="node-set-expression">
    (<xsl:sort>*,template)
  </xsl:for-each>
  ```

`if` (instruction) - Managing the creation of result nodes (page 73)

- XSLT 9.1 Conditional Processing with `xsl:if`
- ```
  <xsl:if test="boolean-expression">
    template
  </xsl:if>
  ```

`import` - Why modularize logical and physical structure of stylesheets? (page 70)

- XSLT 2.6 Combining Stylesheets
- ```
  <xsl:import href="uri-reference"/>
  ```

`include` (top level element) - Why modularize logical and physical structure of stylesheets? (page 70)

- XSLT 2.6 Combining Stylesheets
- ```
  <xsl:include href="uri-reference"/>
  ```

`key` (top level element) - XPath and XSLT functions and advanced techniques (page 76)

- XSLT 12.2 Keys
- ```
  <xsl:key match="pattern"
           name="qname"
           use="expression"/>
  ```

`message` (instruction) - The XSLT transformation environment (page 66)

- XSLT 13 Messages
- ```
  <xsl:message terminate="yes|no"?>
    template
  </xsl:message>
  ```

`namespace-alias` (top level element) - The XSLT transformation environment (page 66)

- XSLT 7.1 Creating Elements and Attributes
- ```
  <xsl:namespace-alias result-prefix="prefix|#default"
                       stylesheet-prefix="prefix|#default"/>
  ```

`number` (instruction) - Managing the creation of result nodes (page 73)

- XSLT 7.7 Numbering
- ```
  <xsl:number count="pattern"?
              format="string|{string-expression}"?
              from="pattern"?
              grouping-separator="char|{string-expression}"?
              grouping-size="number|{string-expression}"?
              lang="nmtoken|{string-expression}"?

  letter-value="alphabetic|traditional|{string-expression}"?
              level="single|multiple|any"?
              value="number-expression"?/>
  ```

`otherwise` - Managing the creation of result nodes (page 73)

- XSLT 9.2 Conditional Processing with `xsl:choose`
- ```
  <xsl:otherwise>
    template
  </xsl:otherwise>
  ```

`output` (top level element) - The XSLT transformation environment (page 66)

- XSLT 16 Output
- ```
  <xsl:output cdata-section-elements="qnames"?
              doctype-public="string"?
              doctype-system="string"?
              encoding="string"?
              indent="yes|no"?
              media-type="string"?
              method="xml|html|text|qname-but-not-ncname"?
              omit-xml-declaration="yes|no"?
              standalone="yes|no"?
              version="nmtoken"?/>
  ```

`param` (top level element) - Why modularize logical and physical structure of stylesheets? (page 70)

- XSLT 11 Variables and Parameters
- ```
  <xsl:param name="qname"
             select="expression"?>
    template
  </xsl:param>
  ```

`preserve-space` (top level element) - The need for abstractions (page 60)

- XSLT 3.4 Whitespace Stripping
- ```
  <xsl:preserve-space elements="tokens"/>
  ```

`processing-instruction` (instruction) - Managing the creation of result nodes (page 73)

- XSLT 7.3 Creating Processing Instructions
- ```
  <xsl:processing-instruction name="ncname|{string-expression}">
    template
  </xsl:processing-instruction>
  ```

`sort` - Sorting and grouping (page 81)

- XSLT 10 Sorting
- ```
  <xsl:sort case-order="upper-first|lower-first|{string-expression}"?
  data-type="text|number|qname-but-not-ncname|{string-expression}"?
            lang="nmtoken|{string-expression}"?
            order="ascending|descending|{string-expression}"?
            select="string-expression"?/>
  ```

`strip-space` (top level element) - The need for abstractions (page 60)

- XSLT 3.4 Whitespace Stripping
- ```
  <xsl:strip-space elements="tokens"/>
  ```



  ```

`stylesheet` - The XSLT transformation environment (page 66)

- XSLT 2.2 Stylesheet Element

```
<xsl:stylesheet version="number"
                exclude-result-prefixes="tokens"?
                extension-element-prefixes="tokens"?
                id="id"?>
  (<xsl:import>*,top-level-elements)
</xsl:stylesheet>
```

`template` (top level element) - A predictable behavior for XSLT processors (page 64)

- XSLT 5.3 Defining Template Rules

```
<xsl:template match="pattern"?
              mode="qname"?
              name="qname"?
              priority="number"?>
  (<xsl:param>*,template)
</xsl:template>
```

`text` (instruction) - Managing the creation of result nodes (page 73)

- XSLT 7.2 Creating Text

```
<xsl:text disable-output-escaping="yes|no"?>
  #PCDATA
</xsl:text>
```

`transform` - The XSLT transformation environment (page 66)

- XSLT 2.2 Stylesheet Element

```
<xsl:transform version="number"
               exclude-result-prefixes="tokens"?
               extension-element-prefixes="tokens"?
               id="id"?>
  (<xsl:import>*,top-level-elements)
</xsl:transform>
```

`value-of` (instruction) - A predictable behavior for XSLT processors (page 64)

- XSLT 7.6 Computing Generated Text

```
<xsl:value-of select="string-expression"
              disable-output-escaping="yes|no"?/>
```

`variable` (top level element) - Why modularize logical and physical structure of stylesheets? (page 70)

- XSLT 11 Variables and Parameters

```
<xsl:variable name="qname"
              select="expression"?>
  template
</xsl:variable>
```

`when` - Managing the creation of result nodes (page 73)

- XSLT 9.2 Conditional Processing with `xsl:choose`
- ```
  <xsl:when test="boolean-expression">
    template
  </xsl:when>
  ```

`with-param` - Why modularize logical and physical structure of stylesheets? (page 70)

- XSLT 11.6 Passing Parameters to Templates
- ```
  <xsl:with-param name="qname"
                  select="expression"?>
    template
  </xsl:with-param>
  ```


XPath and XSLT function summary

Annex C - Instruction, function and grammar summaries
Section 1 - Vocabulary and functions

All functions of both XPath and XSLT in alphabetical order follow. This information is mechanically derived from the XPath 1.0 and XSLT 1.0 Recommendations.

`boolean` - XPath and XSLT functions and advanced techniques (page 76)

- XPath 4.3 Boolean Functions
- *`boolean`* `boolean(` *`object`* `)`

`ceiling` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.4 Number Functions
- *`number`* `ceiling(` *`number`* `)`

`concat` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.2 String Functions
- *`string`* `concat(` *`string`*`,` *`string`*`,` *`string*`* `)`

`contains` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.2 String Functions
- *`boolean`* `contains(` *`string`*`,` *`string`* `)`

`count` - XPath and XSLT functions and advanced techniques (page 78)

- XPath 4.1 Node Set Functions
- *`number`* `count(` *`node-set`* `)`

`current` - XPath and XSLT functions and advanced techniques (page 78)

- XSLT 12.4 Miscellaneous Additional Functions
- *`node-set`* `current( )`

`document` - XPath and XSLT functions and advanced techniques (page 78)

- XSLT 12.1 Multiple Source Documents
- *`node-set`* `document(` *`object`*`,` *`node-set?`* `)`

`element-available` - Why modularize logical and physical structure of stylesheets? (page 70)

- XSLT 15 Fallback
- *`boolean`* `element-available(` *`string`* `)`

`false` - XPath and XSLT functions and advanced techniques (page 76)

- XPath 4.3 Boolean Functions
- *`boolean`* `false( )`

`floor` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.4 Number Functions
- *`number`* `floor(` *`number`* `)`

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

`format-number` - XPath and XSLT functions and advanced techniques (page 77)

- XSLT 12.3 Number Formatting
- *`string`* `format-number(` *`number`*`,` *`string`*`,` *`string?`* `)`

`function-available` - Why modularize logical and physical structure of stylesheets? (page 70)

- XSLT 15 Fallback
- *`boolean`* `function-available(` *`string`* `)`

`generate-id` - XPath and XSLT functions and advanced techniques (page 78)

- XSLT 12.4 Miscellaneous Additional Functions
- *`string`* `generate-id(` *`node-set?`* `)`

`id` - XPath and XSLT functions and advanced techniques (page 78)

- XPath 4.1 Node Set Functions
- *`node-set`* `id(` *`object`* `)`

`key` - XPath and XSLT functions and advanced techniques (page 78)

- XSLT 12.2 Keys
- *`node-set`* `key(` *`string`*`,` *`object`* `)`

`lang` - XPath and XSLT functions and advanced techniques (page 76)

- XPath 4.3 Boolean Functions
- *`boolean`* `lang(` *`string`* `)`

`last` - The need for abstractions (page 60)

- XPath 4.1 Node Set Functions
- *`number`* `last( )`

`local-name` - XPath and XSLT functions and advanced techniques (page 78)

- XPath 4.1 Node Set Functions
- *`string`* `local-name(` *`node-set?`* `)`

`name` - XPath and XSLT functions and advanced techniques (page 78)

- XPath 4.1 Node Set Functions
- *`string`* `name(` *`node-set?`* `)`

`namespace-uri` - XPath and XSLT functions and advanced techniques (page 78)

- XPath 4.1 Node Set Functions
- *`string`* `namespace-uri(` *`node-set?`* `)`

`normalize-space` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.2 String Functions
- *`string`* `normalize-space(` *`string?`* `)`

`not` - XPath and XSLT functions and advanced techniques (page 76)

- XPath 4.3 Boolean Functions
- *`boolean`* `not(` *`boolean`* `)`

`number` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.4 Number Functions
- *number* `number(` *object?* `)`

`position` - The need for abstractions (page 60)

- XPath 4.1 Node Set Functions
- *number* `position( )`

`round` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.4 Number Functions
- *number* `round(` *number* `)`

`starts-with` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.2 String Functions
- *boolean* `starts-with(` *string*`,` *string* `)`

`string` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.2 String Functions
- *string* `string(` *object?* `)`

`string-length` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.2 String Functions
- *number* `string-length(` *string?* `)`

`substring-after` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.2 String Functions
- *string* `substring-after(` *string*`,` *string* `)`

`substring-before` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.2 String Functions
- *string* `substring-before(` *string*`,` *string* `)`

`substring` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.2 String Functions
- *string* `substring(` *string*`,` *number*`,` *number?* `)`

`sum` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.4 Number Functions
- *number* `sum(` *node-set* `)`

`system-property` - The XSLT transformation environment (page 66)

- XSLT 12.4 Miscellaneous Additional Functions
- *object* `system-property(` *string* `)`

`translate` - XPath and XSLT functions and advanced techniques (page 77)

- XPath 4.2 String Functions
- *string* `translate(` *string*`,` *string*`,` *string* `)`

`true` - XPath and XSLT functions and advanced techniques (page 76)

- XPath 4.3 Boolean Functions
- *`boolean`* `true( )`

`unparsed-entity-uri` - XPath and XSLT functions and advanced techniques (page 78)

- XSLT 12.4 Miscellaneous Additional Functions
- *`string`* `unparsed-entity-uri(` *`string`* `)`

XPath grammar productions

Annex C - Instruction, function and grammar summaries
Section 2 - Grammars

Location Paths (2)

```
[1] LocationPath              ::= RelativeLocationPath[3]
                                | AbsoluteLocationPath[2]
[2] AbsoluteLocationPath      ::= '/' RelativeLocationPath[3]?
                                | AbbreviatedAbsoluteLocationPath[10]
[3] RelativeLocationPath      ::= Step[4]
                                | RelativeLocationPath[3] '/' Step[4]
                                | AbbreviatedRelativeLocationPath[11]
```

Location Steps (2.1)

```
[4] Step                      ::= AxisSpecifier[5] NodeTest[7] Predicate[8]*
                                | AbbreviatedStep[12]
[5] AxisSpecifier             ::= AxisName[6] '::'
                                | AbbreviatedAxisSpecifier[13]
```

Axes (2.2)

```
[6] AxisName                  ::= 'ancestor'
                                | 'ancestor-or-self'
                                | 'attribute'
                                | 'child'
                                | 'descendant'
                                | 'descendant-or-self'
                                | 'following'
                                | 'following-sibling'
                                | 'namespace'
                                | 'parent'
                                | 'preceding'
                                | 'preceding-sibling'
                                | 'self'
```

Node Tests (2.3)

```
[7] NodeTest                  ::= NameTest[37]
                                | NodeType[38] '(' ')'
                                | 'processing-instruction' '(' Literal[29]
                                ')'
```

Predicates (2.4)

```
[8] Predicate                 ::= '[' PredicateExpr[9] ']'
[9] PredicateExpr             ::= Expr[14]
```

Abbreviated Syntax (2.5)

```
[10] AbbreviatedAbsoluteLocationPath
                              ::= '//' RelativeLocationPath[3]
[11] AbbreviatedRelativeLocationPath
                              ::= RelativeLocationPath[3] '//' Step[4]
```

```
[12] AbbreviatedStep        ::= '.'
                                | '..'
[13]                        ::= '@'?
AbbreviatedAxisSpecifier
```

Expressions (3)

Basics (3.1)

```
[14] Expr                   ::= OrExpr[21]
[15] PrimaryExpr            ::= VariableReference[36]
                                | '(' Expr[14] ')'
                                | Literal[29]
                                | Number[30]
                                | FunctionCall[16]
```

Function Calls (3.2)

```
[16] FunctionCall           ::= FunctionName[35] '(' ( Argument[17] ( ','
                                Argument[17] )* )? ')'
[17] Argument               ::= Expr[14]
```

Node-sets (3.3)

```
[18] UnionExpr              ::= PathExpr[19]
                                | UnionExpr[18] '|' PathExpr[19]
[19] PathExpr               ::= LocationPath[1]
                                | FilterExpr[20]
                                | FilterExpr[20] '/' RelativeLocationPath[3]
                                | FilterExpr[20] '//' RelativeLocationPath[3]
[20] FilterExpr             ::= PrimaryExpr[15]
                                | FilterExpr[20] Predicate[8]
```

Booleans (3.4)

```
[21] OrExpr                 ::= AndExpr[22]
                                | OrExpr[21] 'or' AndExpr[22]
[22] AndExpr                ::= EqualityExpr[23]
                                | AndExpr[22] 'and' EqualityExpr[23]
[23] EqualityExpr           ::= RelationalExpr[24]
                                | EqualityExpr[23] '=' RelationalExpr[24]
                                | EqualityExpr[23] '!=' RelationalExpr[24]
[24] RelationalExpr         ::= AdditiveExpr[25]
                                | RelationalExpr[24] '<' AdditiveExpr[25]
                                | RelationalExpr[24] '>' AdditiveExpr[25]
                                | RelationalExpr[24] '<=' AdditiveExpr[25]
                                | RelationalExpr[24] '>=' AdditiveExpr[25]
```

Numbers (3.5)

```
[25] AdditiveExpr           ::= MultiplicativeExpr[26]
                                | AdditiveExpr[25] '+' MultiplicativeExpr[26]
                                | AdditiveExpr[25] '-' MultiplicativeExpr[26]
[26] MultiplicativeExpr     ::= UnaryExpr[27]
```



```
                    | MultiplicativeExpr[26] MultiplyOperator[34]
                    UnaryExpr[27]
                    | MultiplicativeExpr[26] 'div' UnaryExpr[27]
                    | MultiplicativeExpr[26] 'mod' UnaryExpr[27]
[27] UnaryExpr      ::= UnionExpr[18]
                    | '-' UnaryExpr[27]
```

Lexical Structure (3.7)

```
[28] ExprToken      ::= '(' | ')' | '[' | ']' | '.' | '..' | '@' |
                    ',' | '::'
                    | NameTest[37]
                    | NodeType[38]
                    | Operator[32]
                    | FunctionName[35]
                    | AxisName[6]
                    | Literal[29]
                    | Number[30]
                    | VariableReference[36]
[29] Literal        ::= '"' [^"]* '"'
                    | "'" [^']* "'"
[30] Number         ::= Digits[31] ('.' Digits[31]?)?
                    | '.' Digits[31]
[31] Digits         ::= [0-9]+
[32] Operator       ::= OperatorName[33]
                    | MultiplyOperator[34]
                    | '/' | '//' | '|' | '+' | '-' | '=' | '!='
                    | '<' | '<=' | '>' | '>='
[33] OperatorName   ::= 'and' | 'or' | 'mod' | 'div'
[34] MultiplyOperator ::= '*'
[35] FunctionName   ::= QName[XML-Names-6] - NodeType[38]
[36] VariableReference ::= '$' QName[XML-Names-6]
[37] NameTest       ::= '*'
                    | NCName[XML-Names-4] ':' '*'
                    | QName[XML-Names-6]
[38] NodeType       ::= 'comment'
                    | 'text'
                    | 'processing-instruction'
                    | 'node'
[39] ExprWhitespace ::= S[XML-3]
```

XSLT grammar productions

Annex C - Instruction, function and grammar summaries
Section 2 - Grammars

Template Rules (5)

Patterns (5.2)

```
[1] Pattern                    ::= LocationPathPattern[2]
                                   | Pattern[1] '|' LocationPathPattern[2]
[2] LocationPathPattern        ::= '/' RelativePathPattern[4]?
                                   | IdKeyPattern[3] (('/' | '//')
                                   RelativePathPattern[4])?
                                   | '//'? RelativePathPattern[4]
[3] IdKeyPattern               ::= 'id' '(' Literal[XPath-29] ')'
                                   | 'key' '(' Literal[XPath-29] ','
                                   Literal[XPath-29] ')'
[4] RelativePathPattern        ::= StepPattern[5]
                                   | RelativePathPattern[4] '/' StepPattern[5]
                                   | RelativePathPattern[4] '//' StepPattern[5]
[5] StepPattern                ::= ChildOrAttributeAxisSpecifier[6]
                                   NodeTest[XPath-7] Predicate[XPath-8]*
[6]                            ::= AbbreviatedAxisSpecifier[XPath-13]
ChildOrAttributeAxisSpecifier      | ('child' | 'attribute') '::'
```

Annex D - Sample tool information

- Introduction - Sample questions for vendors
- Section 1 - Java-based XSLT processors
- Section 2 - Mike Kay's Saxon
- Section 3 - Apache Xalan
- Section 4 - James Clark's XT
- Section 5 - Microsoft Internet Explorer

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

Sample questions for vendors

Annex D - Sample tool information

Answers to the following questions may prove useful when trying to better understand an XSLT product offering from a vendor. The specific questions are grouped under topical questions. This by no means makes up a complete list of questions as you may have your own criteria to add, nonetheless, they do cover aspects of XSLT that may impact on the stylesheets and transformation specifications you write.

- how is the product identified?
    - what is the name of the XSLT processor in product literature?
    - what value is returned by the `xsl:vendor` system property?
    - what value is returned by the `xsl:vendor-url` system property?
    - what version of XSLT is supported (also returned by the `xsl:version` system property)?
    - to which email address or URL are questions forwarded for more information in general?
    - to which email address or URL are questions forwarded for more information specific to the answers to these technical questions?
- what output serialization methods are supported for the result node tree?
    - XML?
    - HTML?
    - text?
    - XSL formatting and flow objects?
        - in what ways are the formatting objects interpreted (direct to screen? HTML? PostScript? PDF? TeX? etc.)?
    - other non-XML text-oriented methods different than the standard text method (e.g. NXML by XT)?
        - what are the semantics and vocabulary for each such environment?
    - other custom serialization methods?
        - what are the semantics and vocabulary for each such environment?
    - what customization is available to implement one's own interpretation of result tree semantics?
        - is there access to the result tree as either a DOM tree or SAX events?
        - does such access still oblige serialization to an external file?

Sample questions for vendors (cont.)

Annex D - Sample tool information

- how does the processor differ from the W3C working drafts or recommendations?
    - upon which dated W3C documents describing XSL, XSLT and XPath is the software based?
    - which constructs or functions are not implemented at all?
    - which constructs or functions are implemented differently than in the W3C description?
    - what namespace URI values are used for those available constructs or functions described differently or not described in W3C version?
    - is the W3C recommended stylesheet association technique implemented for the direct processing XML instances?
        - if so, can it be selectively engaged and disengaged?
- are any extension functions or extension elements implemented?
    - what is the recognized extension namespace and the utility of the extension functions and elements implemented?
        - is there an extension function for the conversion of a result tree fragment to a node-set?
        - are there any built-in extension functions or extension elements for the writing of templates to an output URL?
            - is the XT-defined extension element `<document>` and associated namespace explicitly supported?
    - can additional extension functions or extension elements (beyond those supplied by the vendor) be added by the user?
        - how so?
    - are any extensions defined by `exslt.org` supported?

Sample questions for vendors (cont.)

Annex D - Sample tool information

- how are particular XSLT facilities implemented?
  - what is the implementation in the processor of `indent="yes"` for `<xsl:output>`?
  - is a method provided for defining top-level `<xsl:param>` constructs at invocation time?
  - how is the `<xsl:message>` construct implemented?
  - which UCS/Unicode format tokens are supported for `<xsl:number>`?
  - which `lang=` values are supported for `<xsl:sort>`?
- how are XSLT errors reported or gracefully handled?
  - regarding template conflict resolution?
  - regarding improper content of result tree nodes (e.g. comments, processing instructions)?
  - regarding invocation of unimplemented functions or features?
  - regarding any other areas?
  - can fatal error reporting (e.g. template conflict resolution or other errors) be selectively turned on to diagnose stylesheets targeted for use with other XSLT processors that fail on an error?

Sample questions for vendors (cont.)

Annex D - Sample tool information

- what are the details of the implementation and invocation of the XSLT processor?
  - which hardware/operating system platforms support the processor?
  - which character sets are supported for the input file encoding and output serialization?
  - what is the XML processor used within the XSLT processor?
    - does the XML processor support minimally declared internal declaration subsets with only attribute list declarations of `ID`-typed attributes?
    - does the XML processor support XML Inclusions (Xinclude)?
    - does the XML processor support catalogues for public identifiers?
    - does the XML processor validate the source file?
      - can this be turned on and off?
  - can the processor be embedded in other applications?
    - can the processor be configured as a servlet in a web server?
    - is there access to the result tree as either a DOM tree or SAX events?
  - is the source code of the processor available?
    - in what language is the processor written?
  - for Windows-based environments:
    - can the processor be invoked from the MSDOS command-line box?
    - can the processor be invoked from a GUI interface?
    - what other methods of invocation can be triggered (DLL, RPC, etc.)?
    - can error messages be explicitly redirected to a file using an invocation parameter (since, for example, Windows-95 does not allow for redirection of the standard error port to a file)?
  - does the processor take advantage of parallelism when executing the stylesheet, or is the stylesheet always processed serially?
  - does the processor implement tail recursion for called named templates?
  - does the processor implement lazy evaluation for XPath location path expression evaluation?

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

Working with Java-based processors

An environment for `.jar` Java object files

- `http://java.sun.com/j2se/`

An environment for Windows-based `.exe` compiled Java object files

- `http://www.microsoft.com/java`
- requires Internet Explorer 4 or above

Typical messages or indications when the Microsoft Java Virtual Machine is either not found or needs to be updated to a more current version are as follows:

- `The Microsoft virtual machine cannot be found.`
- `The parameter is incorrect.`
- `Invalid class.`
- the output generated using Saxon is always the empty file

When attempting to execute the Java runtime environment or the Windows `.exe` files, typical messages when the program cannot be found on the execution path are as follows:

- `Bad command or file name`
- `The name specified is not recognized as an...`

Recall that in any version of Windows the execution path can be temporarily changed in an MSDOS window by the command:

- `set path=`*`directory-with-exe-file`*`;%path%`
- note that when using Windows NT, the control panel setting of the system path environment variable does not impact any already-open command line windows and it is necessary to exit and restart the command line window

Invoking Saxon

Annex D - Sample tool information
Section 2 - Mike Kay's Saxon

Using Saxon as a Java program

A source version of the tool is available through links in the web-based documentation, which includes details on the invocation parameters:

- `http://saxon.sf.net`

Requires the following file to be found on the class path:

- `saxon.jar`

A sample invocation batch file using Java runtime environment JRE is as follows (a derivation of this file can be found in the `/prog` directory of the sample files):

```
01 @echo off
02 REM xsltjavasaxon.bat
03 REM check arguments: %1=source XML, %2=script XSL, %3=result XML
04 if a%3 == a goto :noout
05
06 REM environment: SET saxon=p:\saxon\saxonjavacur\
07 echo Invoking Saxon jar....
08 java -cp "%saxon%;%saxon%saxon.jar" ^
09          com.icl.saxon.StyleSheet -o %3 %1 %2
10 REM post-process results
11 goto :done
12
13 :noout
14 echo Saxon Args:  input sytlesheet output
15 goto :done
16
17 :done
```

Invoking Saxon (cont.)

Note how on line 8 in the above example, the invocation file is expecting the environment variable named `saxon` to be set to the directory in which the required Java files are found (commented out in line 6):

- `set saxon=`*`directory-with-required-files`*
- note the trailing directory separator character must be included in the value
- this is not obligatory as the invocation file can be changed to specify the locations of the `.jar` files explicitly

Note also that the invocation file is not complete with respect to passing top-level parameter binding values.

Invoking Saxon (cont.)

Annex D - Sample tool information
Section 2 - Mike Kay's Saxon

Using Saxon as a Windows executable

An executable version of the tool named `saxon.exe` and using the Microsoft Java Virtual Machine is available through links in the web-based documentation:

- is described in the information file for "Instant Saxon" at `http://saxon.sf.net`
- requires `saxon.exe` file to be found on the execution path

A batch file can wrap the invocation of Saxon with preprocessing and post-processing commands, rather than invoking Saxon directly:

```
01 @echo off
02 REM xslsaxon.bat
03 REM arguments: %1=source XML, %2=script XSL, %3=result XML
04 echo Invoking Saxon executable....
05
06 REM do preprocessing here
07
08 if exist ..\prog\saxon.exe goto :usedirect
09
10 :usepath
11 saxon -o %3 %1 %2
12 goto :done
13
14 :usedirect
15 ..\prog\saxon -o %3 %1 %2
16 goto :done
17
18 :done
19
20 REM do postprocessing here
```

Invoking Saxon (cont.)

Annex D - Sample tool information
Section 2 - Mike Kay's Saxon

This indirect invocation appears to behave the same way as direct invocation:

```
01 C:\xslt\samp>..\prog\xslsaxon hello.xml hello.xsl hello.htm
02
03 C:\xslt\samp>type hello.htm
04 <b><i><u>Hello world.</u></i></b>
05 C:\xslt\samp>
```

Not shown above are two (of many) aspects of invocation not engaged by the example batch file:

- additional command line parameters following the source, stylesheet and output provide binding values for top-level parameters:
  - *parameter-name*=*parameter-value*
- using the `-a` command line option engages the recognition of the stylesheet association processing instruction

Invoking Xalan

Annex D - Sample tool information
Section 3 - Apache Xalan

Using Xalan as a Java program

A source version of the tool is available through links in the web-based documentation, which includes details on the use of the processor:

- `http://xml.apache.org/xalan-j`
- includes Xerces XML processor `.jar` file

Version 2.0D06 requires the following files to be found on the class path:

- `xalan.jar`
- `xerces.jar`

A sample invocation batch file using the Java runtime environment is as follows (this file can be found in the `/prog` directory of the sample files):

```
01 @echo off
02 REM xsltjavaxalan.bat
03 REM check arguments: %1=source XML, %2=script XSL, %3=result XML
04 if a%3 == a goto :noout
05
06 if not exist ..\prog\xalan.jar goto :noenv
07 if not exist ..\prog\xerces.jar goto :noenv
08
09 echo Invoking Xalan/Xerces jar: "%1" with "%2" to "%3"
10 java -cp "..\prog\xalan.jar;..\prog\xerces.jar"
org.apache.xalan.xslt.Process -in %1 -xsl %2 -out %3
11 goto :done
12
13 :noenv
14 if not exist ..\prog\xalan.jar echo "xalan.jar" does not exist in
..\prog\
15 if not exist ..\prog\xerces.jar echo "xerces.jar" does not exist in
..\prog\
16 goto :done
17
18 :noout
19 echo Xalan Args:  input stylesheet output
20 goto :done
21
22 :done
```

XT differences from W3C XSLT/XPath Recommendations

Annex D - Sample tool information
Section 4 - James Clark's XT

This is a Java-based implementation of XSLT and XPath that:

- is described in the information file `http://www.jclark.com/xml/xt.html` in detail
- follows the XSL Transformations recommendation very closely with few exceptions (noted in detail in the information file)
    - notably, there is no implementation of `<xsl:key>`
- does not interpret the XSL Formatting Object vocabulary
- does not report very many kinds of errors and will recover silently in many areas
- is acknowledged by the author to be only a beta release
- is documented regarding how to be used as a servlet
- is available with full source code
- is available as a Windows executable dependent only on the Microsoft Java Virtual Machine

Although a few of the more esoteric features are not implemented, the tool can be used for most typical uses of the XSLT recommendation.

XT includes implementations of element, function and serialization method extensions illustrating the extensibility of the XSLT language.

A support site dedicated to XT is at `http://www.4xt.org/`.

Extension element: Multiple output documents

Creating separate result trees

- `xmlns:`*`prefix`*`="http://www.jclark.com/xt"`
- `<`*`prefix`*`:document>`
- only the `href=` attribute is required, other attributes are optional
- `href="`*`output-URL-of-document`*`"` (an attribute value template) specifies the name and location of the document created by the processor
- other attributes are identical to and override any stylesheet use of `<xsl:output>` (e.g. `method=`, described in Chapter 5 The XSLT transformation environment (page 65)), and are *not* attribute value templates
- the template content of this extension instruction element is written to the specified URL

Consider the need to create a number of inter-linked HTML documents from a single XML source as in this following example:

- the content of each HTML file comes from portions of the source tree
- the XML use of `ID` and `IDREF` typed attributes ensure self-referentially correct linking in the source:
  - `<link idref="doc2"/>` is a link to document 2 found in the content portion of document 1
  - `<linkend id="doc2"/>` is the target of the link and is found in document 2
- the corresponding emission of the links must create HTML hyperlink (anchor) markup utilizing the document identifiers from ancestral elements:
  - `<a href="multhtm2.htm#doc2">` is needed in the generated `multhtm1.htm` as the source
  - `<a name="doc2">` is needed in `multhtm2.htm` as the target

Extension element: Multiple output documents (cont.)

The following XML source `multhtm.xml` describes two documents and links between them:

```
<?xml version="1.0"?>
<!DOCTYPE test [
<!ATTLIST linkend id ID #REQUIRED>
]>
<test>
<doc file="multhtm1.htm">
<head><title>Test Output 1</title></head>
<body><linkend id="doc1"/>
<p>This is a link to <link idref="doc2"/>
from document 1.</p>
</body>
</doc>
<doc file="multhtm2.htm">
<head><title>Test Output 2</title></head>
<body><linkend id="doc2"/>
<p>This is a link to <link idref="doc1"/>
from document 2.</p>
</body>
</doc>
</test>
```


Extension element: Multiple output documents (cont.)

The following XSL source `multhtm.xsl` utilizes XT's extension instruction and creates the described hyperlinked files:

```
01 <?xml version="1.0"?><!--multhtm.xsl-->
02 <!--XSLT 1.0 - http://www.CraneSoftwrights.com/training -->
03 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
04                 version="1.0"
05                 xmlns:xt="http://www.jclark.com/xt"
06                 extension-element-prefixes="xt">
07
08 <xsl:template match="/">
09   <xsl:for-each select="//doc"> <!--copy documents out as HTML-->
10     <xt:document method="html" href="{@file}">
11       <html>
12         <xsl:apply-templates select="*"/>
13       </html>
14     </xt:document>
15   </xsl:for-each>
16 </xsl:template>
17
18 <xsl:template match="link">  <!--create link to other document-->
19   <a href="{id(@idref)/ancestor::doc/@file}#{@idref}">
20     <xsl:value-of select="@idref"/>
21   </a>
22 </xsl:template>
23
24 <xsl:template match="linkend">     <!--create anchor for links-->
25   <a name="{@id}"/>
26 </xsl:template>
27                          <!--all other elements just copied-->
28 <xsl:template match="*|@*">
29   <xsl:copy>
30     <xsl:apply-templates select="*|@*|text()"/>
31   </xsl:copy>
32 </xsl:template>
33
34 </xsl:stylesheet>
```

Extension element: Multiple output documents (cont.)

Each output file is produced containing the link to the other, as in the first document:

```
<html>
<head>
<title>Test Output 1</title>
</head>
<body>
<a name="doc1"></a>
<p>This is a link to <a href="multhtm2.htm#doc2">doc2</a>
from document 1.</p>
</body>
</html>
```

And as in the second document:

```
<html>
<head>
<title>Test Output 2</title>
</head>
<body>
<a name="doc2"></a>
<p>This is a link to <a href="multhtm1.htm#doc1">doc1</a>
from document 2.</p>
</body>
</html>
```


Extension functions: Node set manipulation

Creating and manipulating node sets:

- `xmlns:`*`prefix`*`="http://www.jclark.com/xt"`
- *`prefix`*`:node-set(`*`result-tree-fragment-or-node-set`*`)`
  - used to convert a result tree fragment variable into a node set
  - the node set returned is a single unnamed node similar to a root node in that the child constructs of the variable declaration become child nodes of the unnamed node returned by the function
- *`prefix`*`:intersection(`*`first-node-set`*`,`*`second-node-set`*`)`
  - return as a node set the members of the first node-set argument that are also members of the second node-set argument
- *`prefix`*`:difference(`*`first-node-set`*`,`*`second-node-set`*`)`
  - return as a node set the members of the first node-set argument that are not members of the second node-set argument

Extension functions: Node set manipulation (cont.)

The stylesheet `xtnodes.xsl` illustrates the use of all three:

```
<?xml version="1.0"?><!--xtnodes.xsl-->
<!--XSLT 1.0 - http://www.CraneSoftwrights.com/training -->
<!DOCTYPE xsl:stylesheet [
<!ENTITY nl "&#xd;&#xa;">
]>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xt="http://www.jclark.com/xt"
                version="1.0">
<xsl:output method="text"/>

<xsl:template match="/">                              <!--root rule-->
  <xsl:variable name="data">    <!--a fragment with 4 elements-->
     <a>The A element</a><b>The B element</b>
     <c>The C element</c><d>The D element</d>
  </xsl:variable>
     <!--process children of the node returned by the function-->
  <xsl:for-each select="xt:node-set($data)/*">
    <xsl:value-of select="."/><xsl:text>&nl;</xsl:text>
  </xsl:for-each>
  <!--determine intersection and difference using two children-->
  <xsl:text>Intersection:</xsl:text><xsl:text>&nl;</xsl:text>
  <xsl:for-each select="xt:intersection(xt:node-set($data)/*,
                                        xt:node-set($data)/b|
                                        xt:node-set($data)/c)">
    <xsl:value-of select="."/><xsl:text>&nl;</xsl:text>
  </xsl:for-each>
  <xsl:text>Difference:</xsl:text><xsl:text>&nl;</xsl:text>
  <xsl:for-each select="xt:difference(xt:node-set($data)/*,
                                      xt:node-set($data)/b|
                                      xt:node-set($data)/c)">
    <xsl:value-of select="."/><xsl:text>&nl;</xsl:text>
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```


Extension functions: Node set manipulation (cont.)

The results when running the stylesheet with any file as input are as follows (note that since the stylesheet itself is an XML file, it is acceptable to be used as a source file; the stylesheet is not performing any operations on the source tree nodes):

```
C:\xslt\samp>xt xtnodes.xsl xtnodes.xsl
The A element
The B element
The C element
The D element
Intersection:
The B element
The C element
Difference:
The A element
The D element

C:\xslt\samp>
```

Extension functions: Java library access

The Java platform has a number of packages of classes that are formally described at:

- `http://java.sun.com/products/jdk/1.2/docs/api/index.html`
- each package has a name, e.g.: "`io`"
- each class in each package has a name, e.g.: "`File`"

XT provides access to the methods in a class by declaring a namespace prefix whose associated URI points to the class; access to the functions then uses the method names within that namespace as in the following example for the class above:

- `xmlns:`*`prefix`*`="http://www.jclark.com/xt/java/java.io.File"`
  - point to the `File` class in the `io` package
- *`prefix`*`:new(`*`string`*`)`
  - returns a `File` object with the given string as the file name
- *`prefix`*`:exists(`*`File-object`*`)`
  - returns `true` or `false` if the file exists or not

Extension functions: Java library access (cont.)

Consider a stylesheet that reads an instance of file names and reports whether or not the given files exist:

```
<?xml version="1.0"?><!--exists.xsl-->
<!--XSLT 1.0 - http://www.CraneSoftwrights.com/training -->
<!DOCTYPE xsl:stylesheet [ <!ENTITY nl "&#xd;&#xa;"> ]>
<xsl:stylesheet version="1.0"
         xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
         xmlns:file="http://www.jclark.com/xt/java/java.io.File">

<xsl:output method="text"/>

<xsl:template match="/">
  <xsl:if test="not( function-available('file:exists') and
                     function-available('file:new') )">
    <xsl:message terminate="yes">
      <xsl:text>Required Java file facilities </xsl:text>
      <xsl:text>are not available</xsl:text>
    </xsl:message>
  </xsl:if>
  <xsl:for-each select="//file">        <!--process each file-->
    <xsl:text>File </xsl:text><xsl:value-of select="."/>
    <xsl:text>: </xsl:text>             <!--display file name-->
    <xsl:choose>                         <!--report existence-->
      <xsl:when test="file:exists(file:new(string(.)))">
        <xsl:text>exists&nl;</xsl:text>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>does not exist&nl;</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

Extension functions: Java library access (cont.)

Consider executing the stylesheet on the `exists.xml` source file:

```
<?xml version="1.0"?>
<files>
<file>exists.xml</file>
<file>exists.junk</file>
<file>exists.xsl</file>
</files>
```

The following is the result of executing the stylesheet:

```
C:\xslt\samp>xt exists.xml exists.xsl
File exists.xml: exists
File exists.junk: does not exist
File exists.xsl: exists

C:\xslt\samp>
```


Extension method: Non-XML serialization

Annex D - Sample tool information
Section 4 - James Clark's XT

XT supports an extension output method to emit escapable non-XML text (different semantics than the XSLT text output method; available in XT in advance of the standardization of the text output method):

- named the "`nxml`" method and requested using the extension namespace URI for XT:
  - `xmlns:`*`prefix`*`="http://www.jclark.com/xt"`
  - `method="`*`prefix`*`:nxml"`
- expects the result tree to conform to a simple document model that follows

```
<!ELEMENT nxml (escape*, (control | data)*)> <!--document elem.-->
   <!--tell XT which characters to escape when found in <data>-->
<!ELEMENT escape (#PCDATA | char)*>
<!ATTLIST escape char CDATA #REQUIRED>     <!--character value-->
     <!--information in <data> is escaped by XT when necessary-->
<!ELEMENT data    (#PCDATA | data | control)*>
    <!--all information in <control> is passed untouched by XT-->
<!ELEMENT control (#PCDATA | char | data | control)*>
<!ELEMENT char    EMPTY>      <!--emit a single character by XT-->
<!ATTLIST char    number NMTOKEN #REQUIRED> <!--character value-->
```

To create raw text output from an XML file source, transform the source into a result tree instance according to the Non-XML document model and XT will interpret that instance into a stream of text characters using the character set indicated by the `encoding=` attribute (if present) used in the `<xsl:output>` element.

This is different than the raw `method="text"` output method that is part of XSLT in that it provides:

- a customizable escaping mechanism for characters that are sensitive to processes using the output
- control over which content is subject to the customized escaping and which content is written out in clear text
- the ability to specify the arithmetic value (possibly the result of a calculation) of a character code point for serialization

Extension method: Non-XML serialization (cont.)

Element type `nxml` (line 1):

- required document element of result tree

Element type `escape` (lines 2-4):

- optional and repeatable declaration of a sensitive character
  - declares each character of `data` elements that is considered sensitive and needs to be translated on output
    - the attribute `char=` specifies the sensitive character needing translation
    - the character data of the element is the sequence to emit in the output file in place of the sensitive character
  - only sensitive characters found in the `<data>` elements are translated; those found in the `<control>` elements are not touched

Element type `data` (lines 5-6):

- result content where sensitive characters are significant
  - sensitive characters are translated to their declared replacement sequences
  - all other data is written in clear non-escaped characters

Element type `control` (lines 7-8):

- result content where all data is written in clear non-escaped characters
  - sensitive characters are not recognized or treated specially

Element type `char` (lines 9-10):

- a numerically calculated character
  - an output character described by the numeric value of the `number=` attribute
  - useful when the character is unprintable
    - for example, most control characters below hex 20 are not valid in an XML file, so cannot be specified in either a source file or stylesheet file
  - useful when the character is generated algorithmically
  - is an attribute value template

Extension method: Non-XML serialization (cont.)

Consider the following stylesheet using the XT NXML serialization method:

```
<?xml version="1.0"?><!--nxml.xsl-->
<!--XSLT 1.0 - http://www.CraneSoftwrights.com/training -->
<!--XT (see http://www.jclark.com/xml/xt.html)-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

 <!--encoding attribute from suffix of sun.io.ByteToCharLatin1-->
<xsl:output method="xt:nxml"
            encoding="Latin1"
            xmlns:xt="http://www.jclark.com/xt"/>

<xsl:template match="/">       <!--kind of text being produced-->
  <nxml>
    <escape char="\">\\</escape>   <!--escape any back slashes-->
    <data><xsl:apply-templates/></data>  <!--translate <data>-->
  </nxml>
</xsl:template>

<xsl:template match="Name|Address">     <!--prefix information-->
  <xsl:value-of select="name(.)"/>: <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="charValue"> <!--don't translate <control>-->
  <control><xsl:text>\</xsl:text>
           <xsl:value-of select="@val"/>-<char number="{@val}"/>
           <xsl:text>\</xsl:text>
  </control>
</xsl:template>

</xsl:stylesheet>
```

Extension method: Non-XML serialization (cont.)

Consider an XML file of information to be translated to simple text with the "\" character used as an escape character and delimiter for explicit character numbers:

```
01 <?xml version="1.0"?>
02 <test><Name>Crane Softwrights Ltd.</Name>
03 <Address>Box 266, Kars, ON Canada</Address>
04 <info>Test with chars:
05   &lt; \ &amp; <charValue val="38"/> end.</info>
06 </test>
```

The following is the output, with "\\" representing a single "\" and a single "\" delimiting special characters:

```
01 Name: Crane Softwrights Ltd.
02 Address: Box 266, Kars, ON Canada
03 Test with chars:  < \\ & \38-&\ end.
```


Character encoding for the serialized result

Serialization method `encoding=` attribute of `<xsl:output>` control element

- only for HTML, text and NXML serialization methods
  - not supported for XML serialization, which is fixed as UTF-8
- the character set used when serializing the result tree (described in Chapter 5 The XSLT transformation environment (page 65)) can be specified
  - when not specified, this defaults to that supported by the Java virtual machine
  - a specified attribute value is prefixed by the string "`sun.io.ByteToChar`" to determine the Java virtual machine encoding class to be utilized
    - for example, on the JRE `encoding="ISO8859_1"` will engage the "`sun.io.ByteToCharISO8859_1.class`" when serializing the result tree
    - for example, on both the JRE and the Microsoft Java Machine `encoding="Latin1"` will engage the "`sun.io.ByteToCharLatin1.class`" when serializing the result tree
    - to determine which encoding values are supported by a particular Java virtual machine, review a listing of the methods in the class libraries in alphabetical order to group all available classes contiguously

Invoking XT

Annex D - Sample tool information
Section 4 - James Clark's XT

Using XT as a Java program

A source version of the tool is available through links in the web-based documentation, which includes details on the invocation parameters:

- `http://www.jclark.com/xml/xt.html`

One can use any Simple API for XML (SAX) driver, though such a driver is supplied for use with the XP processor (XP itself is not supplied with XT); for example, the following files would need to be found on the class path:

- `xt.jar` - from `http://www.jclark.com/xml/xt.html`
- `xp.jar` - from `http://www.jclark.com/xml/xp/index.html`
- `sax.jar` - from `http://www.saxproject.org`

A sample invocation batch file using the Java runtime environment JRE is as follows (this file can be found in the `/prog` directory of the sample files):

```
01 @echo off
02 REM xsljavaxt.bat
03 REM check arguments: %1=source XML, %2=script XSL, %3=result XML
04 REM environment: SET jclark=p:\jclark\
05 jre -cp "%jclark%xt.jar;%jclark%sax.jar;%jclark%xp.jar" ^
06     com.jclark.xsl.sax.Driver %1 %2 %3
07 REM post-process results
```

Note how on line 5 in the above example, the invocation file is expecting the environment variable named `jclark` to be set to the directory in which the required `.jar` files are found (commented out in line 4):

- `set jclark=`*`directory-with-.jar-files`*
- note the trailing directory separator character must be included in the value
- this is not obligatory as the invocation file can be changed to specify the locations of the `.jar` files explicitly

Note also that the invocation file is not complete with respect to passing top-level parameter binding values.

Invoking XT (cont.)

Annex D - Sample tool information
Section 4 - James Clark's XT

Using XT as a Windows executable

A Windows executable version of the tool named `xt.exe` and using the Microsoft Java Virtual Machine is available through links in the web-based documentation. A batch file can wrap the invocation of XT with preprocessing and post-processing commands, rather than invoking XT directly:

```
01 @echo off
02 REM xslxt.bat
03 REM check arguments: %1=source XML, %2=script XSL, %3=result XML
04 xt %1 %2 %3
05 REM post-process results
```

This indirect invocation appears to behave the same way as direct invocation:

```
01 C:\xslt\samp>..\prog\xslxt hello.xml hello.xsl hello.htm
02
03 C:\xslt\samp>type hello.htm
04 <b><i><u>Hello world.</u></i></b>
05 C:\xslt\samp>
```

Not shown above are two aspects of invocation not engaged by the example batch file:

- additional command line parameters following the source, stylesheet and output provide binding values for top-level parameters:
  - *parameter-name*=*parameter-value*
- the source and output arguments can specify directories instead of files to process all of the source files in the source directory with the specified stylesheet to produce correspondingly named output files in the output directory:
  - xt *input-directory* *stylesheet-file* *output-directory*

Invoking the Microsoft MSXML processor

Installing the latest web release of MSXML with IE5

The latest MSXML `msxml3.exe` is installed and engaged to override archaic implementations of MSXML installed by Internet Explorer 5. This is not required for Internet Explorer 6 as the product already includes an acceptable version of MSXML.

- installation package
  - `http://msdn.microsoft.com/downloads/webtechnology/ xml/msxml.asp`
- package registration utility `xmlinst.exe`
  - see link on above page for installation package
- latest Windows software installation utility (for `.msi` files)
  - `http://www.microsoft.com/msdownload/platformsdk/instmsi.htm`
  - separate versions for NT and other systems
- helpful documentation (unofficial FAQ)
  - `http://www.netcrucible.com/xslt/msxml-faq.htm`

To determine the current release of an MSXML DLL module, use the Windows Explorer and examine the properties. The DLL is found in the `system32` directory of the windows directory. The version tab of the properties dialogue includes an indication of the file version of the DLL.

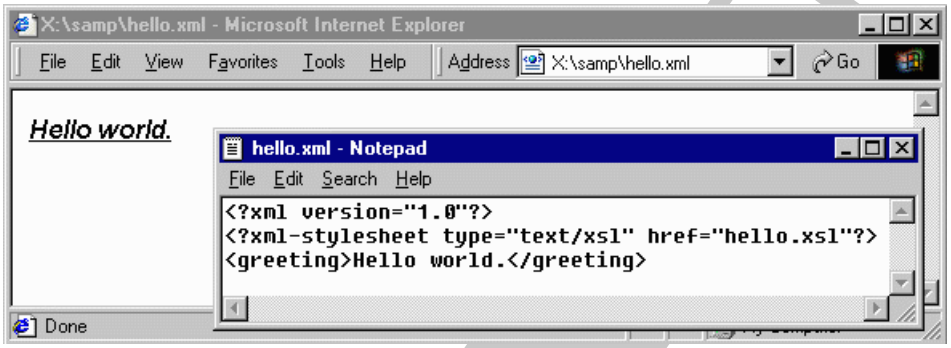
Invoking the Microsoft MSXML processor (cont.)

Annex D - Sample tool information
Section 5 - Microsoft Internet Explorer

Using the processor from within Internet Explorer

- recognizes the use of the stylesheet association processing instruction defined by W3C
  - any associated stylesheet declaration in the XML file is ignored
- can dynamically invoke the processor through scripting
- can utilize the processor on the server side for delivery of HTML

Note that when viewing an XML document, the IE standard menu function View/Source reveals the XML source and not the HTML intermediate form, as in the following example:



Using the processor outside of Internet Explorer:

- any associated stylesheet declaration in the XML file is ignored

Invoking the Microsoft MSXML processor (cont.)

The MSXML processor can be used from the MSDOS command line to emit transformations using the system character set:

- a command line scripting environment from Microsoft executes JavaScript applications
  - `http://msdn.microsoft.com/scripting/windowshost/` - Windows Scripting Host

The following is a simple script adapted from one posted to the XSL mail list. This invokes the MSXML processor after loading two Document Object Model (DOM) objects. In the following code, the processor uses the system character encoding when emitting the result tree (for example, the Latin1 character set on Western European Windows systems and the Shift-JIS character set on Japanese Windows systems):

```
01 //File:  msxml.js - 2000-04-20 16:30
02 //Info:  http://www.CraneSoftwrights.com/links/msxml.htm
03 //Args:  input-file style-file output-file
04 var xml  = WScript.CreateObject("Microsoft.XMLDOM");         //input
05 xml.validateOnParse=false;
06 xml.load(WScript.Arguments(0));
07 var xsl  = WScript.CreateObject("Microsoft.XMLDOM");         //style
08 xsl.validateOnParse=false;
09 xsl.load(WScript.Arguments(1));
10 var out = WScript.CreateObject("Scripting.FileSystemObject");//output
11 var replace = true; var unicode = false; //output file properties
12 var hdl = out.CreateTextFile( WScript.Arguments(2), replace, unicode)
13 hdl.write( xml.transformNode( xsl.documentElement ));
14 //eof
```


Invoking the Microsoft MSXML processor (cont.)

The following is a sample invocation batch file (the `cscript` program is from the Windows Scripting Host utility):

```
01 @echo off
02 REM msxml.bat
03 REM check arguments: %1=source XML, %2=script MSXML, %3=result HTML
04 cscript //nologo ..\prog\msxml.js %1 %2 %3
05 REM post-process results
```

The following is a sample invocation:

```
01 C:\xslt\samp>..\prog\msxml hello.xml hello.xsl hello.mshtm
02 C:\xslt\samp>type hello.mshtm
03 <b><i><u>Hello world.</u></i></b>
04 C:\xslt\samp>
```

Invoking the Microsoft MSXML processor (cont.)

In the following code, the processor uses the UTF-8 character encoding when emitting the result tree (adapted with kind permission from Makoto Murata)

```
01 //File:  msxmlu8.js - 2000-04-20 16:30
02 //Info:  http://www.CraneSoftwrights.com/links/msxml.htm
03 //Args:  input-file style-file output-file
04 var xml  = WScript.CreateObject("Microsoft.XMLDOM");          //input
05 xml.validateOnParse=false;
06 xml.load(WScript.Arguments(0));
07 var xsl  = WScript.CreateObject("Microsoft.XMLDOM");          //style
08 xsl.validateOnParse=false;
09 xsl.load(WScript.Arguments(1));
10 var out = WScript.CreateObject("Microsoft.XMLDOM");          //output
11 out.async = false;
12 out.validateOnParse=false;
13 xml.transformNodeToObject( xsl, out );
14 out.save(WScript.Arguments(2));
15 //eof
```

The two above processes are distinguished by the target of the transformation process:

- the system character set is used when writing the result of transforming the input XML document directly into a stream of characters
- the UTF-8 character set is used when writing out the instance serialization of the object created when transforming the input XML document into a new XML document object

Invoking the Microsoft MSXML processor (cont.)

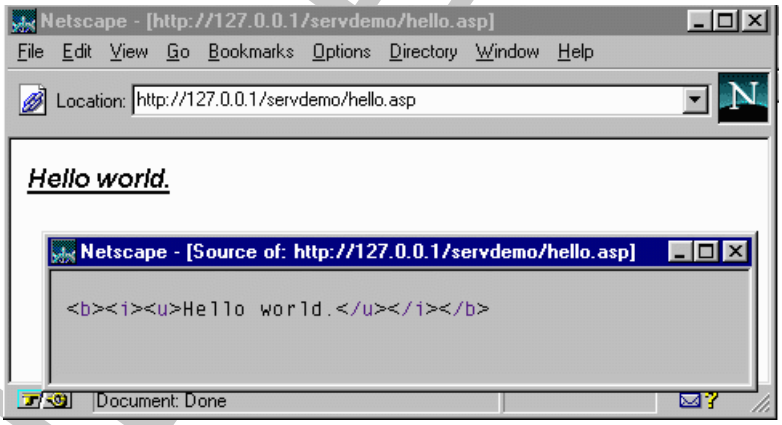
Annex D - Sample tool information
Section 5 - Microsoft Internet Explorer

Using the processor stand-alone on a server

A host process can translate the XML to HTML using the MSXML processor before transmitting the file to the browser:

- on Microsoft server platforms Active Server Pages (ASP) can be used
- the following JScript process uses the Microsoft implementation of the Document Object Model (DOM) and the Microsoft XSL processor to effect the transform

```
01 <%@ LANGUAGE = JScript %>
02 <%
03   // Load the XML source
04   var xml = Server.CreateObject("Microsoft.XMLDOM");
05   xml.async = false;
06   xml.validateOnParse = false;
07   xml.load(Server.MapPath("hello.xml"));
08   // Load the stylesheet file
09   var xsl = Server.CreateObject("Microsoft.XMLDOM");
10   xsl.async = false;
11   xsl.validateOnParse = false;
12   xsl.load(Server.MapPath("hello.xsl"));
13   // Process it
14   Response.Write(xml.transformNode(xsl));
15 %>
```



Where to go from here?

Conclusion - Practical Transformation Using XSLT and XPath

The work on XSL and XSLT continues:

- all XSLT, XPath and XSL-FO are now full W3C Recommendations
- long list of future feature considerations already being examined for new releases of the technology
- new products are continually being announced
- feedback is necessary from users like you!
    - use the XSL mail lists to contribute:
        - `http://www.mulberrytech.com/xsl/xsl-list/`
        - `http://groups.yahoo.com/group/XSL-FO`
        - `http://lists.w3.org/Archives/Public/www-xsl-fo/`
    - contact the XSL editors with comments about the specification:
        - `xsl-editors@w3.org`

Colophon

Conclusion - Practical Transformation Using XSLT and XPath

These materials were produced using structured information technologies as follows:

- authored source materials
  - content in numerous XML files maintained as external general entities for a complete prose book that can be made into a subset for training
    - specification of applicability of constructs for each configuration
      - 45- and 90-minute lecture, half-, full-, two- and three-day lecture and hands-on instruction, and book (prose) configurations
    - an XSLT transformation creates the subset of effective constructs from applying applicability to the complete file
    - content from other presentations/tutorials included semantically (not syntactically) during construct assembly
  - customized appearance engaged with marked sections and both parameter and general entities
    - different host company logos and venue and date marginalia
    - changing a single external parameter entity to a key file includes suite of files for given appearance
- accessible rendition in HTML
  - an XSLT stylesheet produces a collection of HTML files using Saxon for multiple file output
  - mono-spaced fonts and list-depth notation conventions assist the comprehension of the material when using screen-reader software
- printed handout deliverables
  - an XSLT stylesheet produces an instance of XSL formatting objects (XSL-FO) for rendering
  - XPDF `http://www.foolabs.com/xpdf` extracts raw text from PDF files for the back-of-the-book index methodology published as a free resource by Crane Softwrights Ltd.
  - XEP by RenderX `http://www.renderx.com` produces PostScript from XSL-FO
  - GhostScript `http://www.GhostScript.com` produces PDF from PostScript
  - the iText `http://itext.sf.net` PDF manipulation library for Java is used for page imposition by a custom Python `http://www.python.org` program running under the Jython `http://www.jython.org` environment

Obtaining a copy of the comprehensive tutorial

Conclusion - Practical Transformation Using XSLT and XPath

This comprehensive tutorial on XSLT and XPath is available for subscription purchase and free preview download:

- "Practical Transformation Using XSLT and XPath (XSL Transformations and the XML Path Language)" Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8
  - the free download preview excerpt of the publication indicates the number of pages for each topic
- the cost of purchase includes all future updates to the materials with email notification
  - the materials are updated after new releases of the W3C specifications
  - the materials are updated after incorporating comments gleaned during presentations and from feedback from customers
- available in PDF
  - formatted as 1-up or 2-up book pages per imaged page
  - dimensions in either US-letter or A4 page sizes
  - available as either single sided or double sided
- accessible rendition available for use with screen readers
- free preview download includes full text of first two modules and two useful annexes
- site-wide and world-wide staff licenses (one-time fee) are available

See `http://www.CraneSoftwrights.com/links/trn-20040203.htm` for more details.

Feedback

- the unorthodox style has been well-accepted by customers as an efficient learning presentation
- feedback from customers is important to improve or repair the content for future editions
- please send suggestions or comments (positive or negative) to `feedback@CraneSoftwrights.com`

US Government employee purchase

- US Government employees (not contractors) are entitled to obtain their personal prepaid copies at no charge from a government intranet location
- visit the Crane web site for details

Practical Transformation Using XSLT and XPath (XSL Transformations and the XML Path Language)

Crane Softwrights Ltd.
http://www.CraneSoftwrights.com

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

Practical Transformation Using XSLT and XPath

Table of contents
Indexed by slide number

Eleventh Edition - 2004-02-03 - ISBN 1-894049-12-8

Practical Transformation Using XSLT and XPath

Index