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

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

+//ISBN 978-1-894049::CSL::Courses::PTUX//DOCUMENT Practical Transformation Using XSLT and XPath 2011-02-11 21:00UTC//EN

Fourteenth Edition - 2011-02-11
ISBN 978-1-894049-24-5

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

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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

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 46))
- 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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Practical Transformation Using XSLT and XPath

Practical Transformation Using XSLT and XPath

Series: Practical Transformation Using XSLT and XPath

Reference:

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 script and module writing for transformation
- an overview of every element in the recommendations
- an overview of every function in the recommendations
- introduction to XSL formatting semantics

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Information subject to restrictive legend on first page.

Transforming structured information

Introduction - Practical Transformation Using XSLT and XPath

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

This book covers every element, every attribute and every function of both XSLT and XPath, both versions 1.0 and 2.0:

- P1 content specific to XPath 1.0 is marked with a "P1" icon at the beginning of the line
- P2 content specific to XPath 2.0 is marked with a "P2" icon at the beginning of the line
- T1 content specific to XSLT 1.0 is marked with a "T1" icon at the beginning of the line
- T2 content specific to XSLT 2.0 is marked with a "T2" icon at the beginning of the line

First two chapters 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 chapters 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 chapters address XSLT vocabulary

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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Transforming structured information (cont.)

Introduction - Practical Transformation Using XSLT and XPath

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

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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Chapter 1 - The context of XSLT and XPath

- 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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Overview

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
  - information is identified with a vocabulary of labels (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 is using the expected set of labels

XML Path Language (XPath)

- the document model and addressing basis for XSLT and XQuery

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

- XSL Transformations (XSLT)
  - specifies the transformation of structured 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
- identifies the dictionary for the labels used to mark up information

Stylesheet Association

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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Extensible Markup Language (XML)

- `http://www.w3.org/TR/REC-xml`
- `http://www.w3.org/TR/xml11`

A Recommendation fulfilling two objectives for information representation:

- expressing information in a hierarchical arrangement using XML-defined markup
- restricting and/or validating the use of XML markup according to user-specified constraints

Document description and data description

- the roots of XML are from the ISO specification for Standard Generalized Markup Language (SGML) used for document description
- any hierarchical arrangement can be expressed using XML
- any non-hierarchical arrangement can be expressed hierarchically using XML
- XML now commonly used for the description of many kinds of data because of the platform independence of the use of markup and Unicode text

XML defines basic constraints on physical and logical hierarchies of syntax

- 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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Extensible Markup Language (XML) (cont.)

Physical hierarchy (the content organization):

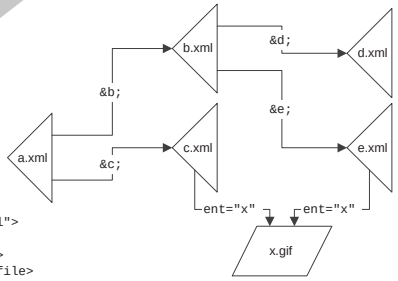
- single collection of information ("XML instance") from multiple physical resources ("XML entities")
  - an XML file is not required to be comprised of more than one physical entity
  - physical modularization typically used to manage a large information set in smaller fragments
  - inappropriately 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>
```



Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Extensible Markup Language (XML) (cont.)

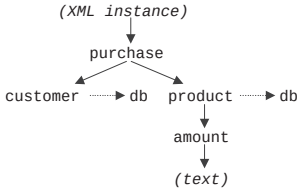
Logical hierarchy (the information):

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

```
<?xml version="1.0"?>
<purchase>
  <customer db="cust123"/>
  <product db="prod345">
    <amount>23.45</amount>
  </product>
</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.



A logical hierarchy need not come from XML syntax

- through "data projection" the logical tree of any information that can be organized as if it came from XML syntax is indistinguishable from that tree that actually does come from XML syntax
- the data model doesn't retain whatever syntax was used (XML or otherwise) to create the logical tree

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

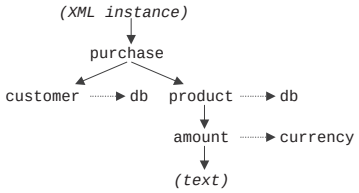
Extensible Markup Language (XML) (cont.)

XML allows user constraints on the logical hierarchy (the vocabulary)

- 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 defines the logical hierarchy of the information constructs *explicitly* according to user-specified constraints
- other structural and content schema languages exist for XML
  - validation constraints extend to values found within text and attribute content
  - different approaches to describing models provide different benefits
- constrains during generation and confirms during processing
- does not convey semantics of information being marked up

```
<?xml version="1.0"?>
<!DOCTYPE purchase [
<!ELEMENT purchase ( customer, product+ )>
<!ELEMENT customer EMPTY>
<!ATTLIST customer db CDATA #REQUIRED>
<!ELEMENT product  ( amount )>
<!ATTLIST product  db CDATA #REQUIRED>
<!ELEMENT amount   ( #PCDATA )>
<!ATTLIST amount   currency ( GBP | CAD | USD ) "USD"> ]>
<purchase>
  <customer db="cust123"/>
  <product db="prod345">
    <amount>23.45</amount>
  </product>
</purchase>
```

The DTD can supplement the data model with additional information:



- note how the shape of the tree is different in the presence of defaulted attribute declarations
  - the currency attribute is included in the tree when the DTD is present
  - without the DTD the logical tree for the sample instance does not include the currency attribute
  - the markup used is identical in both the example instances

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Extensible Markup Language (XML) (cont.)

The equivalent set of document constraints on the logical hierarchy expressed using W3C Schema could be in `purc.xsd`:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="purchase">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="customer">
        <xsd:complexType>
          <xsd:attribute name="db" use="required"/>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="product" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="amount">
              <xsd:complexType mixed="true">
                <xsd:attribute name="currency" default="USD">
                  <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                      <xsd:enumeration value="GBP"/>
                      <xsd:enumeration value="CAD"/>
                      <xsd:enumeration value="USD"/>
                    </xsd:restriction>
                  </xsd:simpleType>
                </xsd:attribute>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
          <xsd:attribute name="db" use="required"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Practical Transformation Using XSLT and XPath

Extensible Markup Language (XML) (cont.)

Chapter 1 - The context of XSLT and XPath
Section 1 - The XML family of Recommendations

The hint that a particular W3C Schema applies to a document is given via reserved attributes

- a processor is not obliged to use the hints

```
<?xml version="1.0"?>
<purchase xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="purc.xsd">
  <customer db="cust123"/>
  <product db="prod345">
    <amount>23.45</amount>
  </product>
</purchase>
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Information subject to restrictive legend on first page.

Extensible Markup Language (XML) (cont.)

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

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

P2 W3C Schema declarations inform the construction of the data model for the XML instance from the Post Schema Validation Infoset

- only when a schema-aware processor is being used *and* when validation is engaged for the source files
  - schema type assignment, default attribute and element value provision, white space normalization of element content
  - the user-supplied lexical form of elements and attributes with atomic schema types may be lost
- when not validated, input information items are treated per the XML information set
  - considered as having unknown data types
- DTD default attribute value declarations override W3C Schema defaults

P1 No respect of element content white space is 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 most XSLT 1 processors
- some recognition of white space can be influenced by the XSLT stylesheet

P2 White space text node disposition is at user request

- strip all, preserve all, strip ignorable

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Extensible Markup Language (XML) (cont.)

XML 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)

An XML document is only a labeled hierarchy of information

- 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 rendition or transformation concepts or constructs

- information representation only, not information presentation or processing
- no defined controls for implying rendering semantics
- the `xml:space` attribute signals whether white space in content is significant to the data definition

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XML information links

Links to useful information

- `http://www.xml.com/axml/axml.html` - annotated version of XML 1.0
- `http://xml.coverpages.org/xml.html` - Robin Cover's famous resource collection
- `http://xml.coverpages.org/xll.html` - Extensible Linking Language
- `http://xml.silmaril.ie/` - Peter Flynn FAQ
- `http://www.xmlbooks.com/` - a summary of available printed books
- `http://www.CraneSoftwrights.com/links/trn-20110211.htm` - training material
- `http://www.CraneSoftwrights.com/resources` - free resources
- `http://XMLGuild.info` - consulting and training expertise
- `http://wiki.eclipse.org/PsychoPathXPathProcessor` - standalone XPath 2.0 processor
- `http://xml.coverpages.org/elementsAndAttrs.html` - a summary of opinions
- `http://google-styleguide.googlecode.com/svn/trunk/xmlstyle.html` - a corporate perspective

Related initiatives and specifications

- `http://www.w3.org/TR/2004/REC-xml-infoset-20040204` - XML Information Set
- `http://www.w3.org/TR/xmlschema-0/` - W3C XML Schema
- `http://www.relax-ng.org` - ISO/IEC 19757-2 RELAX NG (based on RELAX and TREX)
- `http://www.schematron.com` - ISO/IEC 19757-3 Schematron
- `http://www.nvdl.org` - ISO/IEC 19757-4 Namespace-based Validation Dispatching Language (NVDL)
- `http://www.w3.org/TR/DOM-Level-2/` - Document Object Model Level 2
- `http://www.saxproject.org` - Simple API for XML

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XML Path Language (XPath)

Representing structured information

- 1 `http://www.w3.org/TR/xpath`
- 2 `http://www.w3.org/TR/xpath20`
- a data model for representing the information found in 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 XPointer, XSLT and XQuery is added
  - an expression of Boolean, numeric, string and node values as different data types
  - a set of functions working on the values
- 2 annotated with W3C Schema data type information when available
- 2 data model defined for use with XSLT and XQuery:
  - `http://www.w3.org/TR/xpath-datamodel/`

Addressing and finding structured information

- 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 languages needing to address information found in an XML document
  - `id('start')//question[@answer='y']`
    - address all question elements whose `answer` attribute is "`y`" that are descendants of the element in the current document whose unique identifier is "`start`"
    - the result is an address of element nodes
  - 2
```
for $each in id('start')//question[@answer='y']
    return if ($each/@weight) then $each/@weight * 100.
                              else 100.
```
    - for all question elements whose `answer` attribute is "`y`" that are descendants of the element in the current document whose unique identifier is "`start`", return a sequence of numbers where, if that element has a weight attribute return the weight multiplied by 100, otherwise just return 100
    - the result is a sequence of numbers suitable for processing, such as an argument to the `avg()` function

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XML Path Language (XPath) (cont.)

P1 XPath 1.0 is an addressing language and is *not* a query language

- only based on XML 1.0 and Namespaces in XML 1.0
  - expressed in terms of the XML Information Set
  - `http://www.w3.org/TR/xml-infoset`
- only addresses information that needs to be found in an XML document
- other aspects of querying involve working with the information that is addressed before returning a result to the requestor
  - instructions in XSLT perform query functionality
- 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)

P2 XPath 2.0 is very much a query language

- based on W3C Schema XSD 1.0 perspective of an XML document
- supports conditional expressions, actions on the result set, etc.
- very powerful and expressive language for manipulating all types of information before returning the result of manipulation for action

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

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 arrangements 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 arrangement 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) - versions 1.0 and 2.0
  - 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) - versions 1.0 and 1.1
  - 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 with XSL-FO

XSLT and XSL-FO are endorsed by members of WSSSL

- an association of researchers and developers passionate about markup technologies

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Extensible Stylesheet Language (XSL/XSL-FO)

- `http://www.w3.org/TR/2001/REC-xsl-20011015/`
- `http://www.w3.org/TR/xsl11` (`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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Extensible Stylesheet Language Transformations (XSLT)

Addressing, querying and publishing structured information

- 1.1 `http://www.w3.org/TR/xslt`
  - addressing structured information
- 2 `http://www.w3.org/TR/xslt20`
  - querying structured information
- a framework for complex and intelligent querying of structured content
  - with a powerful syntax for modular and extensible stylesheet writing
- works on XML documents
- works on any source of information projected as if it were an XML document
  - such projection is defined by the vendor, not by the specification
  - the specification sees all information as if it had been in an XML document
  - e.g. database tables, rows and columns
  - e.g. unstructured documents
  - e.g. proprietary binary formats
  - any information can be fit (or shoehorned) into an XML document by using data projection
- numerous features for publishing information for human consumption
  - e.g. formatting numbers, dates and times
  - e.g. polymorphism of stylesheet constructs for specialization of behaviors
  - e.g. elaborate grouping criteria
  - e.g. multiple result trees

Shares the same data model as XQuery

- built on XPath 2.0 with additional functions not available in XQuery expressions

Shares the same basic processing model as XQuery

- some XSLT and XQuery implementations share the same core engine
  - e.g. Saxon 9 `http://saxon.sf.net` treats XSLT and XQuery merely as different syntax skins over the same implementation engine

Shares the same serialization specification as XQuery

- used to frame query results as structured or non-structured output of transformation

Syntactically, XSLT is an XML vocabulary

- an XSLT stylesheet is a well-formed XML document
- all use of XPath 2.0 is in attributes of XSLT and other XML elements

Transformation specifications are termed "XSLT stylesheets"

- describing how new results are constructed from old inputs
- termed generically as "a transform" in this training material

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Extensible Stylesheet Language Transformations (XSLT) (cont.)

Transformation using construction 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 reposition over structures and information found in the source
- the information being transformed can be traversed in different ways any number of times required to construct 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 in different fragments
- T2 XSLT 2.0 has many more programming features and function calls than XSLT 1.0

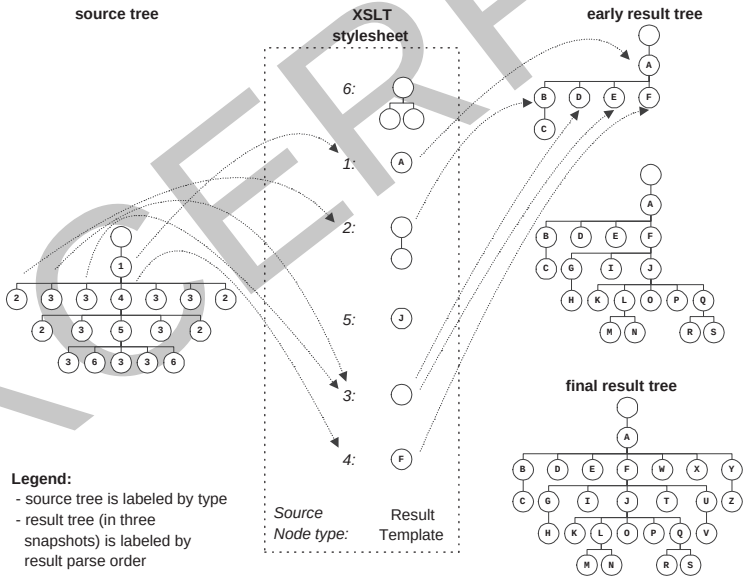
Many language features for modularization and leveraging stylesheets

- supports forms of polymorphism for stylesheet constructs
- supports extensive re-use of stylesheet fragments for generalized transformations or specific transformations
- overriding template rules
  - allows one to create "onion skins" of modifications to stylesheet libraries
- testing the presence of extensions before using them
  - allows one to run one stylesheet with multiple XSLT processors

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Extensible Stylesheet Language Transformations (XSLT) (cont.)

Illustration of templates triggered in source-tree order constructing a result:



Of note:

- the source tree contains nodes of six different types, labeled "1" through "6"
  - a number of nodes are found multiple times in the source tree
- the stylesheet contains fragmented examples of the result tree
  - each example template is associated with a node in the source tree
- the nodes in the source tree trigger the building of the result from the example templates
  - some examples are used multiple times in the result
- in this example, the source tree is visited strictly in parse order to generate the result tree
  - the stylesheet can visit the source tree in whatever order is required to trigger the assembly of the result tree in result parse order
  - result parse order is indicated by the letters "A" through "Z"

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLT properties

Expression syntax is iconic

- using XML markup allows one to *manifest* the output
  - XML is a first-class data type and output expression syntax
  - the syntax itself is abstracted into a tree of nodes
  - syntax not related to the information in the document is not preserved
- using other languages one must *describe the creation* of the output
  - XML is created using function calls, not built into the language syntax

Abstract structure result of nodes, not markup

- external result markup (if needed) is determined from the result node tree
- the result of transformation is a tree of nodes built from instantiated templates as an internal hierarchy that *may* be serialized externally as markup
- the processor may, but is not obliged to, externalize the result tree in XML or some other type of syntax if requested by the transform writer
  - the transform writer has little or no control over the syntactic constructs chosen by the processor for serialization
  - the transform writer can request certain behaviors that the processor can ignore
  - final result is guaranteed to comply with lexical requirements of the output method
    - when not coerced by certain transform controls
  - source tree markup syntax preservation cannot be implemented with a transform
    - because the source tree syntax is translated into source tree nodes and forgotten
- the processing model allows the processor to immediately serialize the result tree as markup while it is being built by the transform, and not maintain the complete result in memory
- the transform may request the processor emit the result tree using built-in available lexical conventions (XML, HTML or text-only conventions)
- 2 multiple result trees may be constructed and serialized

Not intended for syntactic general purpose XML transformations

- designed for downstream-processing and subsequent transformations or interpretation
  - does not include certain features appropriate for syntax-level general purpose transformations
    - unsuitable for original markup syntax preservation requirements
  - 2 XSLT 2.0 has more syntax serialization features than XSLT 1.0
  - includes facilities for working with the XSL vocabulary easily
- still powerful enough for *most* downstream-processing transformation needs
  - where the syntax choices when using XML are not important
  - absolutely general purpose when the output is going to be input to an XML processor

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLT properties (cont.)

Document model and vocabulary independent

- a transform is independent of any Document Type Definition (DTD) or schema that may have been used to constrain the instance being processed
- a 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 transform can process instances of different document models
- multiple instances of different models can be used in a single transformation
- different transforms can process a given single instance to produce different results

Source files and transforms

- one or more source files and one or more transform fragments
  - starting with a single source file and the top-most transform fragment
- 1 all stylesheets and source files must be well-formed XML
- 2 stylesheets must be XML, source files may be simple text or well-formed XML
  - zero or more source files and one or more stylesheet fragments
  - starting with the top-most stylesheet fragment and optionally a source file
- the processor is allowed to deliver well-formed XML from any data source
- Recommendation does not support SGML instances as input
  - see `http://www.w3.org/TR/NOTE-sgml-xml-971215` for a comparison of SGML and XML
  - 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 transform development
  - if the source document does not validate to the model expected by the transform writer, then a correctly functioning transform may exhibit incorrect behavior
  - time spent debugging the working transform is wasted if the source is incorrect
- can selectively validate input documents and result documents using W3C Schema

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLT properties (cont.)

Multiple source files possible

- 1 one mandatory primary source file
- one optional primary source file
  - in the absence of a source file a named template must be specified as where to start processing
- transform may access arbitrary other source files
  - including itself as a source file
  - names of resources hardwired within the transform
  - 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
- 2 simple text files can be input into the process

Extensible language design supplements processing

- a processor *may* support extensions specified in the transform but is not obliged to do so
  - extended functions
  - extended serialization conventions
  - extended sorting schemes
  - extended instructions
- access to non-standardized extensions is specified in standardized ways
- transform user-defined functions can be declared and used

Single-pass construction of the result node-tree

- unlike the Document Object Model (DOM)
  - reified node-tree manipulation (read/write) interface with syntax serialization
- unlike the Simple API for XML (SAX)
  - single-pass input event-handling interface with single-pass result markup syntax
- transform must construct the result tree in result-tree parse order *in one pass*
  - no revisiting of the result tree after construction
  - no revisiting an element's start tag after beginning that element's content
  - recall the result tree building shown on page 23
- the source trees can be traversed in any order (not necessarily in parse order)
  - information in the source trees can be ignored or selectively processed
- the result tree is emitted as if constructed chronologically 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 where partial trees may be constructed in parallel

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Historical development of the XSL and XQuery 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/`
- the XSL 1.1 Recommendation (XSL-FO) published December 5, 2006
  - `http://www.w3.org/TR/2006/REC-xsl11-20061205/`
- 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/2001/WD-xslt11-20010824`
  - 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 2.0/XPath 2.0/XQuery 1.0 originally published January 23, 2007, followed by editorial editions:
  - `http://www.w3.org/TR/2007/REC-xslt20-20070123/`
  - `http://www.w3.org/TR/2010/REC-xpath20-20101214/`
  - `http://www.w3.org/TR/2010/REC-xpath-datamodel-20101214/`
  - `http://www.w3.org/TR/2010/REC-xpath-functions-20101214/`
  - `http://www.w3.org/TR/2010/REC-xslt-xquery-serialization-20101214/`
  - `http://www.w3.org/TR/2010/REC-xquery-20101214/`
  - `http://www.w3.org/TR/2010/REC-xquery-semantics-20101214/`
  - `http://www.w3.org/TR/2010/REC-xqueryx-20101214/`

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

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.openmath.org/cocoon/openmath/` - OpenMath project work by David Carlisle
- `http://www.CraneSoftwrights.com/links/trn-20110211.htm` - comprehensive XSLT/XPath and XSL-FO training material
- `http://XMLGuild.info` - consulting and training expertise
- `http://www.CraneSoftwrights.com/resources`- free XSLT and XSL-FO resources
- `http://incrementaldevelopment.com/xsltrick/` - "Stupid XSLT Tricks"
- `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)

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Namespaces

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

An important role in information representation:

- vocabulary distinction in a single XML document
  - mixing information from different document models
  - labels in the hierarchy are globally unique and identifiable
  - a metaphor is that each namespace is a dictionary with words
    - each dictionary may have a different definition for the same word as found in other dictionaries
    - the namespace identifies which dictionary of words is in use
- possible use for resource discovery being considered
  - generalized associated information regarding information in an instance
  - possible access to document model, transforms, 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
    - following the metaphor, the one word has two different definitions and interpretations, one from each dictionary
    - 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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

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*
  - following the metaphor, the URI uniquely identifies the dictionary of the words
    - supplemental documentation defines the meaning of each of the words
  - 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:978-1-894049:example"`
  - `xmlns:ex2="urn:X-Crane:namespaces:documents:example2"`
  - `xmlns:ex3="ftp://ftp.CraneSoftwrights.com/ns/example3"`
  - `xmlns:ex4="mailto:gkholman@CraneSoftwrights.com"`
- explicitly does not require to de-reference any kind of information from the 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 prefix 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 a common convention but not standard
  - 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 XML tags
- the prefix is a syntactic shortcut preventing the need to specify long distinguishing strings

Different views of the name of `<svg:set>`:

- "`set`" is the local name
- "`svg:set`" is the qualified name
  - a name subject to namespace interpretation (prefixed or un-prefixed)
  - the lexical space for the W3C Schema QName data type
- "`{http://www.w3.org/2000/svg-20000629}set`" is the expanded name
  - combination of namespace URI (also called "namespace name") and the local part
  - the value space for the W3C Schema QName data type
  - the use of "`{`" and "`}`" is not standard, but is used by some tools such as Saxon
- "`http://www.w3.org/2000/svg-20000629#set`" is a URI value convention

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Namespaces (cont.)

An example of using namespaces in a Universal Business Language (UBL) invoice:

- for space reasons the lengthy namespace URI strings have been abbreviated
- note that namespaces are important because there are two elements with the same local name "`Location`", one in each of two different namespaces

```
<Invoice xmlns="urn:oasis:...:xsd:Invoice-2"
         xmlns:cbc="urn:oasis:...:xsd:CommonBasicComponents-2"
         xmlns:cac="urn:oasis:...:xsd:CommonAggregateComponents-2"
         xmlns:ext="urn:oasis:...:xsd:CommonExtensionComponents-2"
         xmlns:demo="urn:x-Demo:Demo">
  <ext:UBLExtensions>
    <ext:UBLExtension>
      <cbc:ID>Demo1</cbc:ID>
      <cbc:Name>Demonstration</cbc:Name>
      <ext:ExtensionAgencyID>CSL</ext:ExtensionAgencyID>
      <ext:ExtensionAgencyName>Crane Softwrights Ltd.
</ext:ExtensionAgencyName>
      <ext:ExtensionVersionID>0.1</ext:ExtensionVersionID>
      <ext:ExtensionAgencyURI>http://www.CraneSoftwrights.com/
links/res-dev.htm</ext:ExtensionAgencyURI>
      <ext:ExtensionURI>urn:x-Demo:Demo:0.1</ext:ExtensionURI>
      <ext:ExtensionReasonCode listURI="urn:x-Demo:Demo:ReasonCodes">1
</ext:ExtensionReasonCode>
      <ext:ExtensionReason>Illustration</ext:ExtensionReason>
      <ext:ExtensionContent>
        <demo:Demo>
          <demo:Thing>This is a test</demo:Thing>
          <cbc:ID>DemoTest</cbc:ID>
          <demo:Total currencyID="GBP">100.00</demo:Total>
        </demo:Demo>
      </ext:ExtensionContent>
    </ext:UBLExtension>
  </ext:UBLExtensions>

  <cbc:ID>A00095678</cbc:ID>
  <cbc:IssueDate>2005-06-21</cbc:IssueDate>
  <cbc:Note>sample</cbc:Note>
  <cac:AccountingSupplierParty>
    <cac:Party>
      <cac:PartyName>
...
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Namespaces (cont.)

Namespaces in XSLT and XSL-FO

- both files are in well-formed XML syntax
  - require all namespaces used to be declared; there are no defaults
- 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

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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Namespaces (cont.)

Naming of top-level constructs in XSLT

- libraries of transform fragments can isolate their constructs by using unique namespace URI strings
- building upon an existing library is done without risking the integrity of the existing stylesheets when one is disciplined about the naming of constructs
- in the following example, two different variables are declared because of the unique namespace URI strings (the prefixes are immaterial)
  - the first is in namespace "`urn:X-a`" and the second is in namespace "`urn:X-b`"
-
```
01 <xsl:variable name="a:thing" select="'abc'" xmlns:a="urn:X-a"/>
02 <xsl:variable name="a:thing" select="'def'" xmlns:a="urn:X-b"/>
```
- 1.0 stylesheet-defined function names must be namespace qualified
- the default namespace is never used for naming top-level constructs

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

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 type="text/xsl" href="../xs/xslstyle-docbook.xsl"?>
```

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

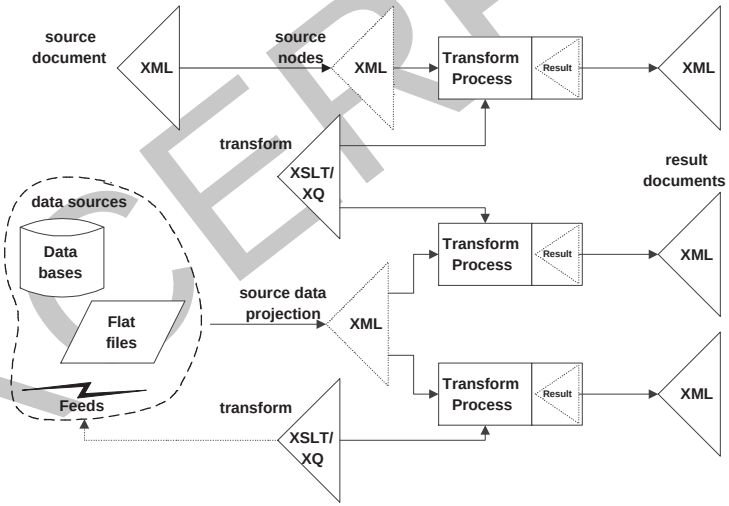
See XSLStyle™ (page 158) for an embedded XSLT documentation methodology

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Transformation from XML to XML

The basic behavior is to transform a hierarchical input into a hierarchical result tree:

- that result tree may be emitted as an XML instance



Of note:

- a given transform can be applied to more than one XML structure
- a given XML structure can have more than one transform applied
- a given XML structure can be derived from an XML file or projected from some other data source identified by the transform
- the result of construction is the abstract result tree within the transform process serialized to the emitted XML under the control of the process
- the dotted triangle in the process represents the abstract node tree of the result

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
  - XML files are drawn with a solid line, node structures are drawn with a dotted line
- unstructured files represented by parallelograms

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Transformation from XML to non-XML

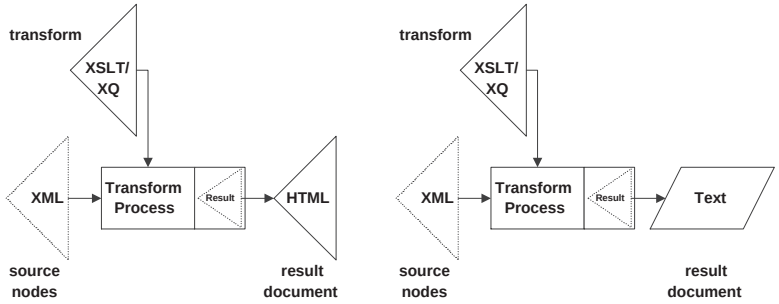
A processor may choose to recognize the transform's request to serialize a non-XML representation of the result tree:

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

Shared serialization specification between XSLT 2.0 and XQuery 1.0

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

At least two non-XML tree serialization methods common to all specifications:



- `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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

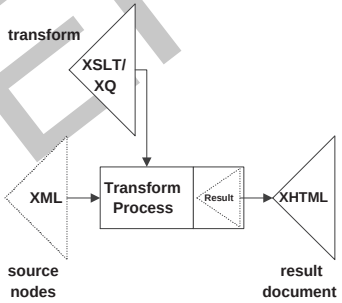
Transformation from XML to non-XML (cont.)

1.1 No standardized support for XHTML lexical conventions

- a processor could offer a custom extension, but many (possibly all?) do not

1.2 Standardized support for XHTML lexical conventions

- `xhtml`
  - browser compatibility guidelines for empty tags for elements defined to be empty
  - no markup minimization for empty elements for elements not defined to be empty

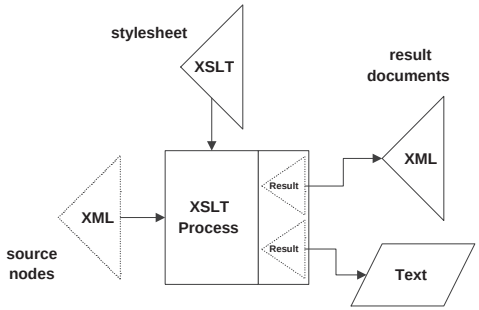


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Transformation from XML to non-XML (cont.)

1 Only standardized support for a single result tree
- most XSLT processors offer a custom extension, but there is no obligation to do so and it is not standardized

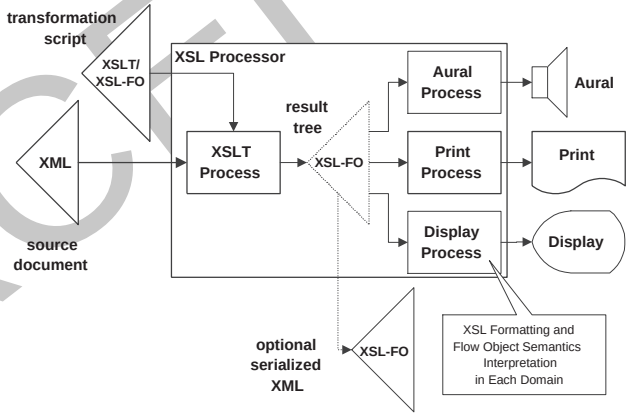
2 Standardized support for multiple result trees
- each result tree can have the same or different serialization
- multiple result trees are not accessible to a single XSL-FO process



Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Transforming and rendering XML information using XSLT and XSL-FO

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 (though 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 or XSLT vocabularies
- 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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

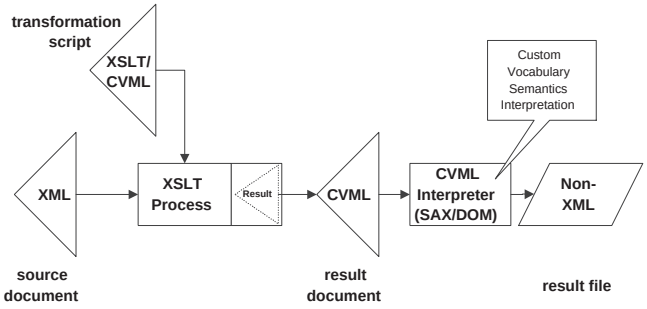
XML to binary or other formats

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

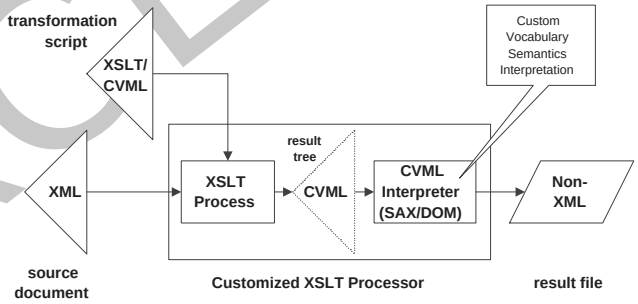
Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

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:

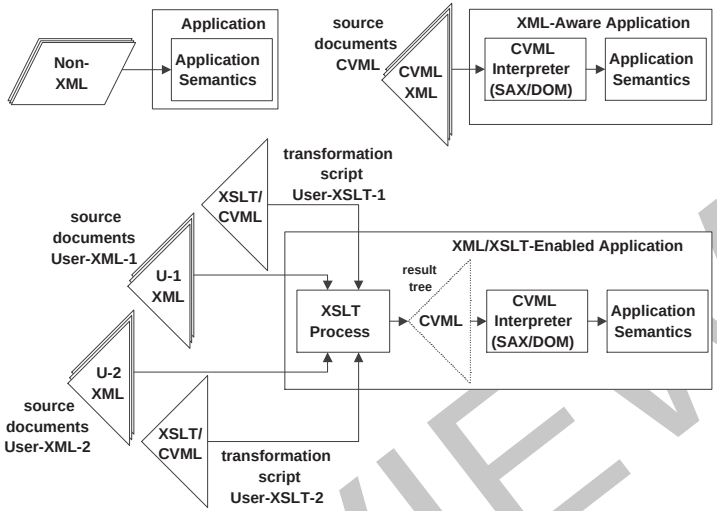


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLT as an application front-end

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

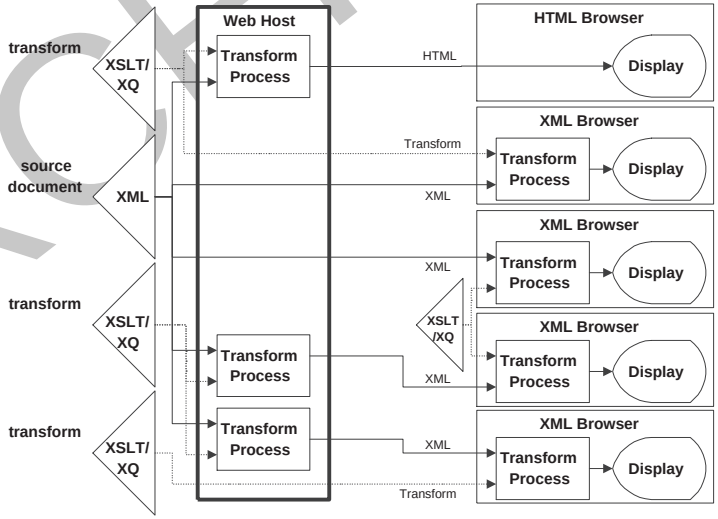


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

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 or XQuery are not supported in a user agent:
  - the host can take on the burden of transformation
- where XML and XSLT or XQuery 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



Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Three-tiered architectures (cont.)

Always performing server-side transformation:

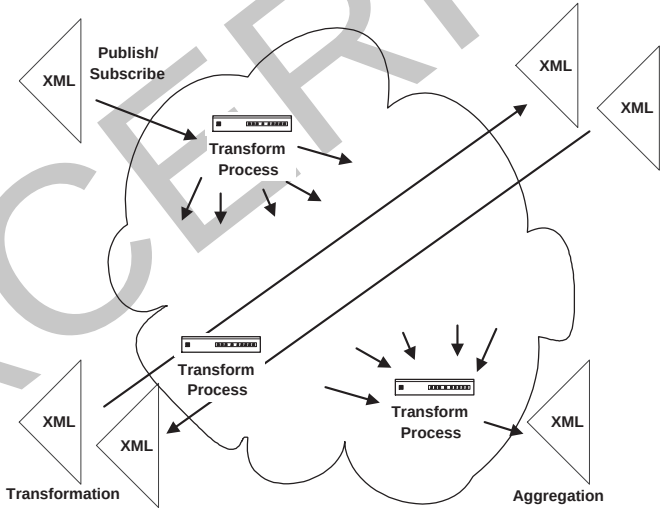
- good business sense in some cases
  - even if technically it is possible to send semantically-rich information
- never send unprocessed semantically-rich XML
  - or only send it to those who are entitled to it
    - for security reasons
    - for payment reasons
- translation into a presentation-orientation
  - using a markup language inherently supported by the user agent (e.g. HTML)
  - using a custom, semantic-less markup language with an associated transformation
- "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"

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLT and XQuery on the wire

XSLT and XQuery have a role in a large or small network cloud:

- simple transformation services can be made available to users on the network, unburdening the user's own infrastructure



Publish/Subscribe

- a network service can accept subscription requests from across the network
- the XML document from the publisher is routed to all subscription destinations
- a subscriber can request a transformation process so as to receive the published information in the desired structure

Aggregation

- a network service can accept XML documents from across the network
- a user of the service can receive the aggregate of all of the information
- the information can be transformed into a homogenous collection for ease of processing and analysis

Transformation

- a user of the network can utilize wire-speed transformation of outgoing and incoming documents to a peer

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Chapter 2 - Getting started with XSLT and XPath

- Introduction - Getting started
- Section 1 - Transform examples
- Section 2 - Syntax basics
- Section 3 - Approaches to transform design
- Section 4 - More transform examples

Outcomes:

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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Getting started

A few simple transformations:

- using Saxon
  - Saxon 6.5.5 (and later) support XSLT 1.0
  - Saxon 9 (and later) support XSLT 2.0 and XQuery 1.0
- 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 transforms

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

This material has a number of handy references harvested from the specification documents:

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

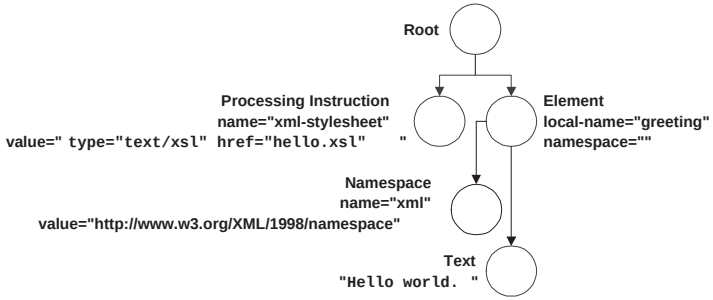
Some simple examples

Chapter 2 - Getting started with XSLT and XPath
Section 1 - Transform 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>
```

This is the complete logical tree of the entire instance:



Note that there is no node in the tree created by the XML Declaration

- the XML Declaration is a syntactic signal in an XML instance regarding the encoding and version of XML being used
- it is consumed by the XML processor as part of the parsing process and is not delivered to an application

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Some simple examples (cont.)

Chapter 2 - Getting started with XSLT and XPath
Section 1 - Transform examples

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"?>
02 <!--hellohtm.xsl-->
03 <!--XSLT 1.0 - http://www.CraneSoftwrights.com/training -->
04 <html xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
05       xsl:version="1.0">
06  <head><title>Greeting</title></head>
07  <body><p>Words of greeting:<br/>
08    <b><i><u><xsl:value-of select="greeting"/></u></i></b>
09    </p>
10  </body>
11 </html>
12
```

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

```
01 C:\ptux\samp>java -jar ../prog/saxon.jar hello.xml hellohtm.xsl
02 <html>
03  <head>
04   <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
05   <title>Greeting</title>
06  </head>
07  <body>
08   <p>Words of greeting:<br><b><i><u>Hello world.</u></i></b></p>
09  </body>
10 </html>
11 C:\ptux\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 4 added by Saxon is ensuring the character set of the web page is properly recognized by conforming user agents.

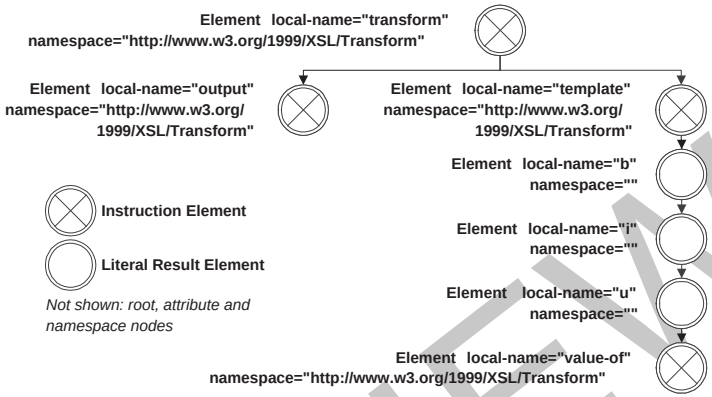
Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

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>
```

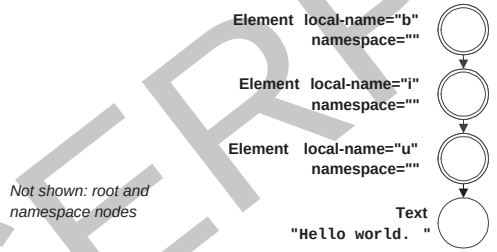
Remember that the syntax of the transform does not represent the syntax of the result, only the nodes of the result; the following is the node tree (not showing attribute and namespace nodes) of the stylesheet:



Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Some simple examples (cont.)

The node tree constructed using the stylesheet (page 50) on the source (page 48) is:

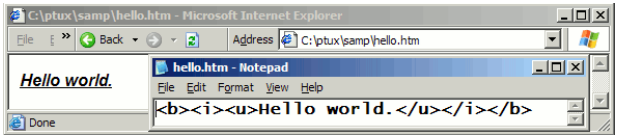


- note from the drawing conventions how the element nodes come from the operation node tree and the text node is calculated from the source node tree information

Using an MSDOS invocation to execute XSLT with the Saxon processor (with `-o` for the output file and `-a` to respect stylesheet association) we see the following tree serialization:

```
01 C:\ptux\samp>java -jar ../prog/saxon.jar -o hello.htm -a hello.xml
02
03 C:\ptux\samp>type hello.htm
04 <b><i><u>Hello world.</u></i></b>
05 C:\ptux\samp>
```

The result `hello.htm` file serialization of the constructed node tree can be viewed with a browser to see the results using the menu selection View/Source to examine the content:



Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Some simple examples (cont.)

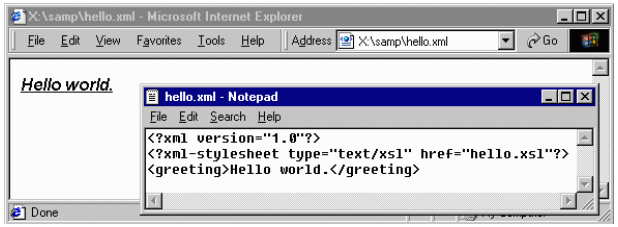
Two ways of working with the Microsoft XSLT processor:

Using the `msxml.bat` invocation batch file (documented in detail in free download preview of on-line tutorial material) at an MSDOS command line to execute the MSXML processor:

```
C:\ptux\samp>..\prog\msxml hello.xml hello.xsl hello-ms.htm
Invoking MSXML....

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



Other browsers support on-the-fly XSLT transformation

- not all browsers have XML processors that support all of the syntax features of XML
  - e.g. lack of support for XML entities

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLT 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
      - historically this is because XSLT was first published as one chapter of the XSL specification
      - all of the examples for XSL were written with "`xsl:`" and remained after XSLT was spun off as its own specification
    - 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
  - this dictates the data model rules for building of the source tree based on XPath 1 or XPath 2
  - also engages the incompatible version-specific behavior of the processor for certain instructions
  - using "`1.0`" for XSLT 2.0 either engages backwards-compatible behavior or signals an error and does not execute the transformation
  - 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`*`"`

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLT instructions and literal result elements

An XSLT instruction:

- is detected in the stylesheet tree only
  - not recognized if used in the source tree
  - instruction defined by the XSLT recommendation and specified using the prefix associated with the XSLT URI
- may be a control construct
  - the wrapper and top-level elements
  - procedural and process-control instructions
  - logical and physical stylesheet maintenance facilities
- may be a construction construct
  - synthesis of result tree nodes
  - copying of nodes from a source node tree
- may be a text value placeholder
  - any calculated value using `<xsl:value-of>` 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 a 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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLT templates and template rules

An XSLT template (a.k.a. "sequence constructor" in XSLT 2.0):

- specifies a fragment for constructing the result tree as a tree of nodes
  - see the nodes in Some simple examples (page 50)
- 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
  - see Extensible Stylesheet Language Transformations (XSLT) (page 23)

```
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 result tree 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 traditional 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 begins the construction of the result tree
- *all* subsequent construction is controlled by the stylesheet

T2 Can begin processing at a specified named template or mode at invocation

- source tree is optional when starting at an arbitrary rule

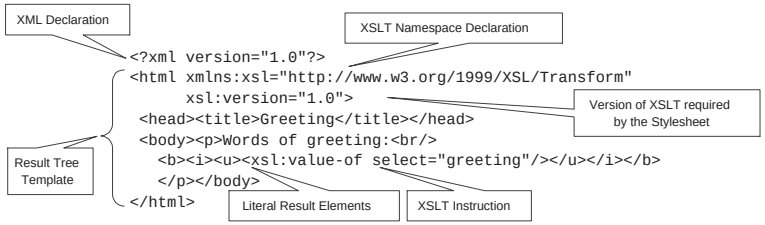
Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLT stylesheet components

Chapter 2 - Getting started with XSLT and XPath
Section 2 - Syntax basics

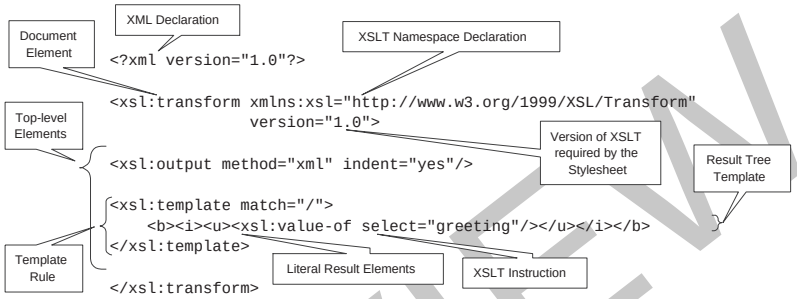
A "simplified" XSLT stylesheet:

- 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:



A more traditional 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
- traditional stylesheets can be utilized by other explicitly declared stylesheets
- identifiable components of this traditional XSLT script:



Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Pull and push constructs

Chapter 2 - Getting started with XSLT and XPath
Section 3 - Approaches to transform design

Consider for illustration an XML file containing sale and purchase information maintained chronologically, thus in an arbitrary order:

```
<chrono-info>
 <purchase>...</purchase>
 <sale>...</sale>
 <sale>...</sale>
 <purchase>...</purchase>
 <sale>...</sale>
 <purchase>...</purchase>
</chrono-info>
```

How one approaches accessing the information to create the result tree varies

- one can pull the information out of the source node tree
- one can push the information from the source node tree at the stylesheet

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Pull and push constructs (cont.)

Pulling the input data and repositioning in the tree

- the hierarchy of the source file is known by the transform writer
- at the point of building "the next" part of the result tree

The transform "pulls" information as and when needed:

- from known locations in the source node tree
- for extraction or calculation using the lexical value
  - `<xsl:value-of select="string(XPath-expression)"/>`
- for extraction or calculation using the schema-qualified value
  - `<xsl:value-of select="XPath-expression"/>`
- for wholesale copying of source tree nodes
  - `<xsl:copy-of select="XPath-expression"/>`
- for repositioning over a sequence of arbitrary locations or values of any data type
  - `<xsl:for-each select="XPath-node-set-expression">`
    `...template...`
    `</xsl:for-each>`
  - `<xsl:for-each select="XPath-sequence-expression">`
    `...template...`
    `</xsl:for-each>`

Transform-determined result order implements direct document construction

- the result tree is built by the transform obtaining each result component from the source, in result order, and framing each component as required with literal markup from the transform
- if the result can be described as a single template using only the "pull" instructions, the transform can be simply declared as a single result template

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Practical Transformation Using XSLT and XPath

Pull and push constructs (cont.)

Chapter 2 - Getting started with XSLT and XPath
Section 3 - Approaches to transform design

Pull example

- to process all of the sales records together, followed by all of the purchase records together, they are pulled from the source tree one set before the others:

```
- <xsl:template match="chrono-info">
    <sale-purchase-summary>
      <xsl:for-each select="sale">
        ...template for the sale...
      </xsl:for-each>
      <xsl:for-each select="purchase">
        ...template for the purchase...
      </xsl:for-each>
    </sale-purchase-summary>
  </xsl:template>
```

- each address "`sale`" and "`purchase`" will respectively find all `<sale>` and `<purchase>` child elements of `<chrono-info>`
- the order of the two instructions will construct the result tree by adding one template for each of the sale elements until all sale elements have been addressed, followed then by adding one template for each of the purchase elements until all purchase elements have been addressed:

```
- <sale-purchase-summary>
    ...result construction for sale...
    ...result construction for sale...
    ...result construction for sale...
    ...result construction for purchase...
    ...result construction for purchase...
    ...result construction for purchase...
  </sale-purchase-summary>
```

Recall the input elements shown on page 57

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Information subject to restrictive legend on first page.

Pull and push constructs (cont.)

Pushing the input data and repositioning in the tree

- 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 transformation
- to process all of the records in the order they appear in the document, they are pushed through the transform logic while specifying the union of all such nodes

XSLT stylesheets:

- the stylesheet "pushes" nodes of information at the template rules:
  - visits known (using names) or unknown (using a wild card) source tree nodes
  - `<xsl:apply-templates select="`*`XPath-node-expression`*`">`
- template rules respond to node visitations by constructing the result tree:
  - `<xsl:template match="`*`XPath-pattern`*`">`

Source-determined result order implements indirect tree construction

- the `<xsl:apply-templates>` instruction is the event generators
  - selects the source information the processor is to visit
- 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

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

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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Practical Transformation Using XSLT and XPath

Pull and push constructs (cont.)

Chapter 2 - Getting started with XSLT and XPath
Section 3 - Approaches to transform design

Push example

- using XSLT:

```
- <xsl:template match="chrono-info">
    <sale-purchase-summary>
      <xsl:apply-templates select="sale | purchase"/>
    </sale-purchase-summary>
  <xsl:template>
```

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

```
- <xsl:template match="sale">...template...</xsl:template>
  <xsl:template match="purchase">...template...</xsl:template>
```

- each address "`sale`" and "`purchase`" will respectively find all `<sale>` and all `<purchase>` elements in the source tree, but the union operator "`|`" will return the set of all those nodes in document order which may very well be interleaved
- the order of the two result expressions is irrelevant because each result expression will be triggered only by the kind of node being matched
- this will construct the result tree by adding one template for each of the sale elements and purchase elements in the document order encountered in the source tree:

```
- <sale-purchase-summary>
    ...result construction for purchase...
    ...result construction for sale...
    ...result construction for sale...
    ...result construction for purchase...
    ...result construction for sale...
    ...result construction for purchase...
  </sale-purchase-summary>
```

Recall the input elements shown on page 57

Contrast the result with that of the pull approach on page 59

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Information subject to restrictive legend on first page.

Processing XML with many transforms

Chapter 2 - Getting started with XSLT and XPath
Section 4 - More transform examples

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>
```

Of note:

- each product is identified with `id=` declared to be of type `ID`
  - permits the element to be addressed with a unique identifier
- there is no total information, only information about each product sale
- the product names are not duplicated
  - the product information is referenced using `idref=` from the product sale

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Processing XML with many transforms (cont.)

Chapter 2 - Getting started with XSLT and XPath
Section 4 - More transform examples

The equivalent set of document constraints on the logical hierarchy expressed using W3C Schema could be in `prod.xsd`:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="sales">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="products">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="product" maxOccurs="unbounded">
              <xsd:complexType mixed="true">
                <xsd:attribute name="id" type="xsd:ID"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="record">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="cust" maxOccurs="unbounded">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="prodsale" maxOccurs="unbounded">
                    <xsd:complexType>
                      <xsd:simpleContent>
                        <xsd:extension base="xsd:integer">
                         <xsd:attribute name="idref" type="xsd:IDREF"/>
                        </xsd:extension>
                      </xsd:simpleContent>
                    </xsd:complexType>
                  </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="num" type="xsd:string"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

- note the declaration of `prodsale` is an integer value
- note the ID/IDREF relationships expressed between `id=` and `idref=`

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Processing XML with many transforms (cont.)

The hint that a particular W3C Schema applies to a document is given via reserved attributes

- a processor is not obliged to use the hints

The following document has a self-referential consistency error that will not be detected unless schema validation is turned on:

- note how customer C1003 has a product sale pointing to a non-existent product

In addition, the ID/IDREF relationships are not recognized unless schema validation is turned on:

- any built-in facilities for supporting ID-typed attributes are not engaged

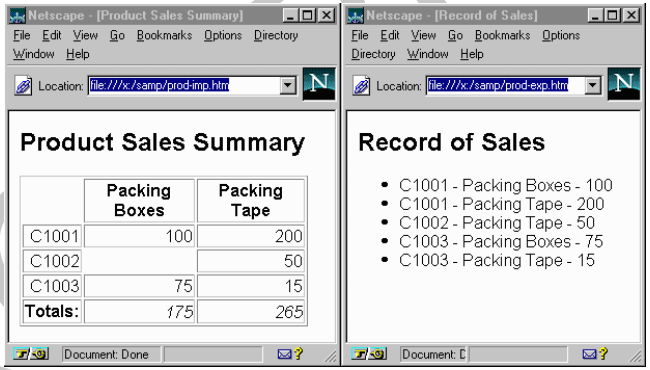
```
01 <?xml version="1.0"?><!--prod-bad.xml-->
02 <sales xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03        xsi:noNamespaceSchemaLocation="prod.xsd">
04   <products><product id="p1">Packing Boxes</product>
05             <product id="p2">Packing Tape</product></products>
06   <record><cust num="C1001">
07             <prodsale idref="p1">100</prodsale>
08             <prodsale idref="p2">200</prodsale></cust>
09           <cust num="C1002">
10             <prodsale idref="p2">50</prodsale></cust>
11           <cust num="C1003">
12             <prodsale idref="p1">75</prodsale>
13             <prodsale idref="p3">15</prodsale></cust></record>
14 </sales>
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Processing XML with many transforms (cont.)

Recall the sample data on page 62

- very dissimilar reports could be generated for the one data file by using different transforms:

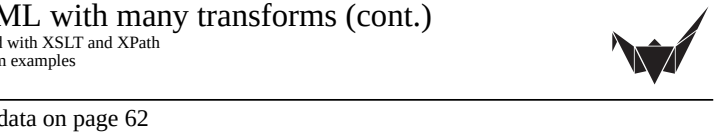


Of note:

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

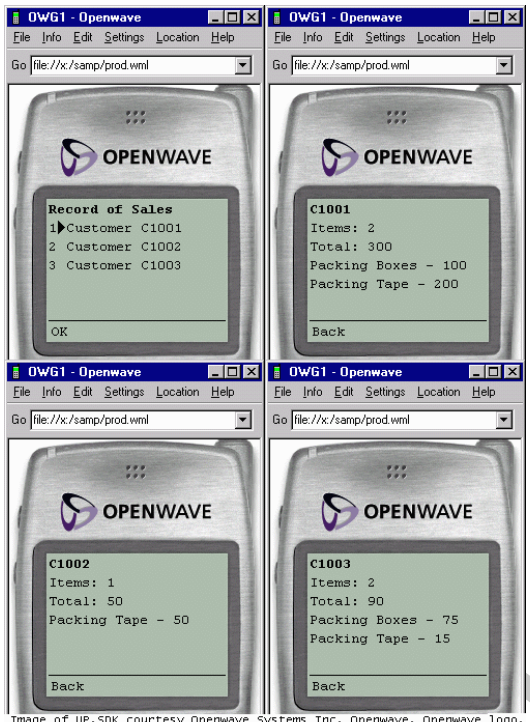
Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Processing XML with many transforms (cont.)



Recall the sample data on page 62

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



Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Processing XML with many transforms (cont.)

The simplified stylesheet `prod-pull.xsl` for the table (page 65) for the XML (page 62):

```
<?xml version="1.0"?><!--prod-pull.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 added 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 visited 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)

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Processing XML with many transforms (cont.)

The traditional stylesheet `prod-push.xsl` for the list (page 65) for the XML (page 62):

```
<?xml version="1.0"?><!--prod-push.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="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: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:stylesheet>
```

Source document is pushed through the stylesheet:

- the order of the template rules is irrelevant
  - only one node is being pushed at the stylesheet, so only one template responds
- 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

An importing stylesheet can exploit the template rule fragmentation

- another stylesheet can import this stylesheet and specialize the behavior of any top-level construct
- an overriding definition of any template rule will take precedence over that rule in the above transformation

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Processing XML with many transforms (cont.)

The traditional stylesheet `prod-wml.xsl` for the WML (page 66) and XML (page 62):

```
<?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 same source is visited twice using different template rules for processing to produce different results

An importing stylesheet can exploit the template rule fragmentation

- another stylesheet can import this stylesheet and specialize the behavior of any top-level construct
- an overriding definition of any template rule will take precedence over that rule in the above transformation

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Chapter 3 - XPath data model

- Introduction - The need for abstractions

Fourteenth Edition - 2011-02-11 - ISB N 978-1-894049-24-5

The need for abstractions

Dealing with information, not markup

- all input and output information manipulated in an abstract fashion
- separate node structures of information:
  - source trees (example on page 48)
    - the main source tree is optional in XSLT 2
    - 1.1 the main source tree is required in XSLT 1
    - multiple additional source documents may be read as separate node structures
  - operation tree (stylesheet example on page 50)
  - result tree (example on page 51)
    - 2.0 multiple result trees can be created using XSLT 2
- knowledge of input markup and control of output markup out of the hands of the transform writer
  - the writer deals with nodes of information, not characters of markup

Traversing a source document or transform document predictably

- XML syntax processed into an abstract data model tree of nodes
  - documents are described according to a data model for the XML markup
    - interpreted in terms of the XML Information Set
      - `http://www.w3.org/TR/xml-infoset/`
    - P2 maintained in terms of a formal XPath data model
      - `http://www.w3.org/TR/xpath-datamodel/`
  - 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
  - P2 some nodes have types based on W3C Schema post-schema validation infoset
    - `http://www.w3.org/TR/xmlschema-1/#d0e504`
  - the processor performs all operations using the node tree, not the document 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 transform 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 transform is responsible for specifying which source nodes get processed when and how

The XPath data model and the DOM data model are different

- the Document Object Model (DOM) is a data model of the information including the syntax in an XML document
- XPath is a data model of the information not including the syntax in an XML document

Fourteenth Edition - 2011-02-11 - ISB N 978-1-894049-24-5

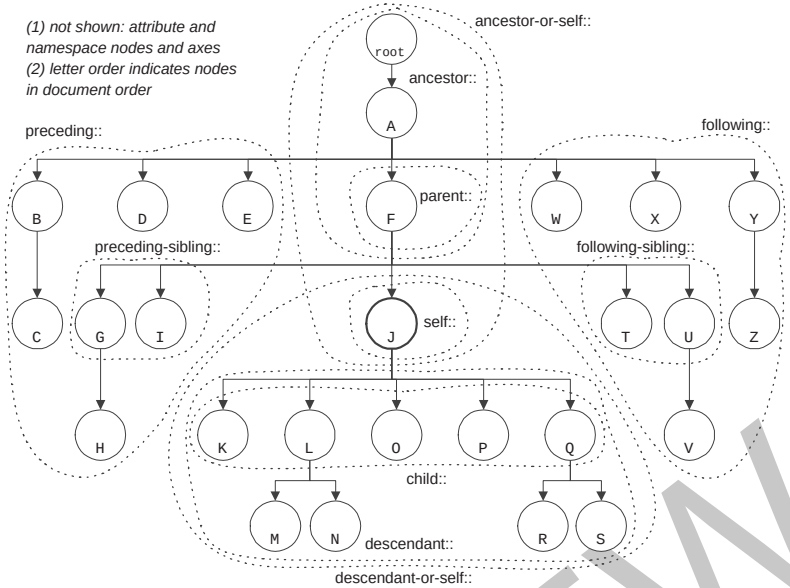
The need for abstractions (cont.)

Chapter 3 - XPath data model

Consider an XML document comprised of 26 empty and non-empty elements named "A" through "Z" (without any text or new-line characters of any kind):

```
01 <A><B><C/></B><D/><E/><F><G><H/></G><I/><J><K/><L><M/><N/></L><O/>
02 <P/><Q><R/><S/></Q></J><T/><U><V/></U></F><W/><X/><Y><Z/></Y></A>
```

The following depicts this document's complete node tree (not showing namespaces):



- 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

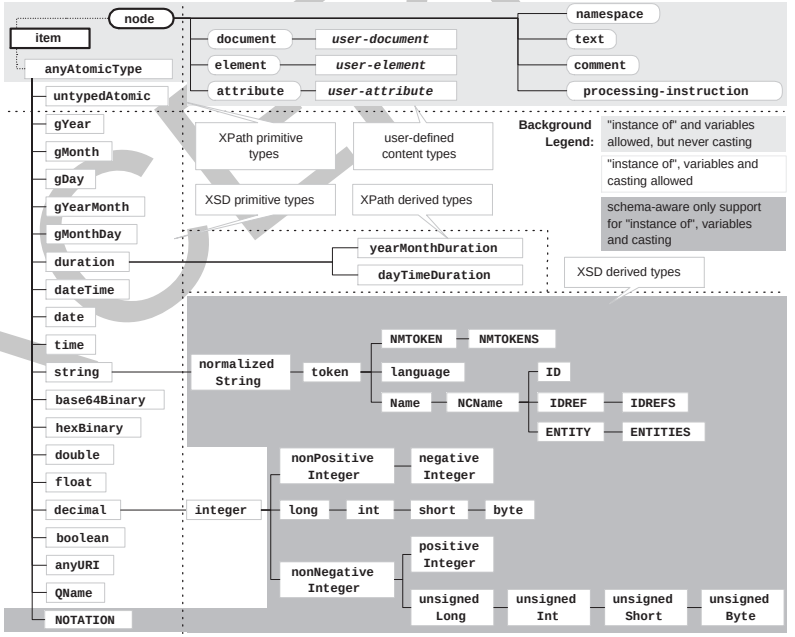
Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Data types

Chapter 3 - XPath data model

P1 XPath 1.0 treats node values as strings and has a limited number of data types
- boolean, number, string, node set and result tree fragment

P2 XPath 2.0 introduces data types based on the W3C Schema data type hierarchy:



When referenced in transforms, types must be namespace qualified:
- e.g. `xmlns:xs="http://www.w3.org/2001/XMLSchema"`
  - any prefix can be used

P2 Types allowed for casting are value constructors when used with function syntax:
- `<xsl:variable name="meetingStart" as="xs:dateTime" select="xs:dateTime('2005-12-04T11:00:00Z')"/>`

P2 Some types are only available in schema-aware versions of the processor
- e.g. `NOTATION` and most XSD derived types

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Sequence types

Chapter 3 - XPath data model

A sequence type is a declaration combination of data type and cardinality

- `item()` - union of any node type or atomic value
- `node()` - any node (tree construction - see page 70)
  - seven types of tree nodes described in this chapter
  - e.g. `element()`, `attribute()`, `text()`, etc.
  - three types of named tree nodes described in this chapter
  - e.g. `element(`*`name`*`)`, `attribute(`*`name`*`)`, `processing-instruction(`*`name`*`)`
  - two types of typed tree nodes described in this chapter
  - e.g. `element(`*`name`*`,`*`type`*`)`, `attribute(`*`name`*`,`*`type`*`)`
  - two types of declared tree nodes described in this chapter
  - e.g. `schema-element(`*`name`*`)`, `schema-attribute(`*`name`*`)`
  - qualified document nodes described in this chapter
  - e.g. `document-node(`*`element-name-type-declaration-test`*`)`
  - user-defined globally-declared types
  - e.g. `mySchema:zip-code`
- `xs:anyAtomicType` - any atomic value (lexical construction - see page 73)
  - W3C Schema data types
  - e.g. `xs:string`, `xs:integer`, `xs:duration`, `xs:gMonth`, etc.
  - XPath 2 data types
  - e.g. `xs:dayTimeDuration`, `xs:yearMonthDuration`
  - e.g. `xs:untypedAtomic` - an atomic value without a type

Non-zero cardinality specified using Kleene operators "`+`", "`?`" and "`*`", for example:

- `empty-sequence()` - zero to zero (i.e. no items of any kind)
- `xs:string` - exactly one string (i.e. mandatory)
- `xs:string?` - zero or one strings (i.e. optional)
- `xs:string+` - one or more strings (i.e. mandatory and repeatable)
- `xs:string*` - zero or more strings (i.e. optional and repeatable)
- `element( email )+` - one or more `<email>` elements
- `element( email )?` - zero or one `<email>` elements
- `xs:item()*` - zero or more items of any type

The following data types are not allowed as sequence types

- `xs:anyType` (un-validated element node)
- `xs:untyped` (un-validated attribute node content)
- `xs:anySimpleType` (lists and unions only)
  - includes `xs:IDREFS`, `xs:NMTOKENS` and `xs:ENTITIES`
  - includes user-defined list and union types

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Constructing result trees

Chapter 3 - XPath data model

Building a result predictably

- created as an abstract tree of nodes
  - the result node tree is constructed using nodes from the operation and source trees, and nodes synthesized by operation tree expressions
  - 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 is reliable and reproducible
    - the processor acts on instructions the same way every time
    - some aspects (e.g. order of attribute) is implementation-dependent
- one-pass construction of the result tree
  - no "going back and changing" anything in the result tree
  - created in a single pass in result parse-order
- serialization instantiates markup syntax
  - the emission of the result node tree (if so desired)
  - in XML markup according to the standard
    - the transform writer does *not* control which markup constructs are used for representing XML information
    - the 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 processor (e.g.: interpretation of a colloquial vocabulary into a binary format)

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XPath data model

Chapter 3 - XPath data model

The XPath productions covered in this chapter are:

- `(: :)`
  - commenting XPath expressions
- `if ( ) then else`
  - XPath choice statement
- `for ... return ...`
  - XPath tuple statement
- `empty-sequence()`
  - the empty sequence sequence type
- `/`
  - location path address steps
- `$`
  - variable references
- `( )`
  - parenthesized expressions

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 XPath functions covered in this chapter are:

- `last()`
  - the number of nodes in the context/current node list
- `position()`
  - the ordinal number of the current node in the context/current node list
- `.`
  - context item

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Chapter 4 - Processing model

- Introduction - A predictable behavior for processors

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

A predictable behavior for processors

The basic processing model for XSLT is designed to ensure predictability

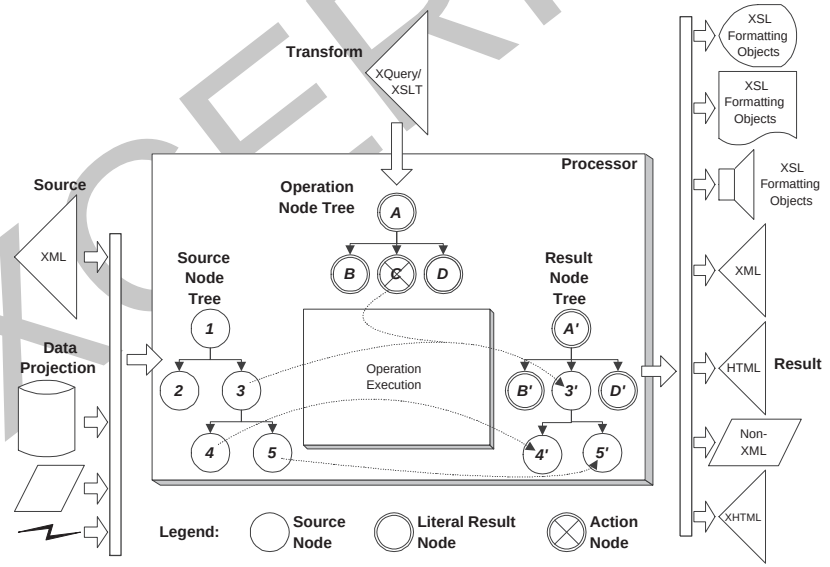
- predictable processing behavior every time
  - all aspects of the processing are well-defined
- processor builds the operation tree of nodes from the transformation expression
  - some nodes of which are evaluations and calculations
  - some nodes of which may be engaging extensions implemented by the processor
  - the remainder of which are literal result elements that are to be used in the construction of the result
- processor builds the source tree of nodes from the primary source resource
  - a primary source tree is not required
  - the markup of the XML source document is not material
  - the vocabulary used in the source is not material to the processor
    - with the exception of `xml:*=` attributes available for use with all XML vocabularies
- result tree construction starts with operation tree
  - start with the template of the template rule for the root node
  - alternatively start at a specified named template or mode
- the transform constructs the content of the result node tree in result parse order in one pass
  - the transform writer must plan the flow of the transform process according to the document order of the result
  - other source trees are created from other source files on request by the transform
  - components from the source trees are obtained where required when executing instructions found in the transform
  - 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 into markup
  - XML markup
  - HTML markup (using SGML lexical conventions)
  - XHTML markup (using XML lexical conventions)
  - simple text
  - syntax and lexical conventions recognized by the particular implementation of the processor (binary or text)
  - interpreted XSL formatting objects (e.g.: display, print, aural, etc.)
  - remember the processor is *not* required to support any particular serialization method and may choose to serialize the tree as XML only if at all

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

A predictable behavior for processors (cont.)

A simple illustration of the basic process

- the illustrated operation node tree has literal result nodes and a single operation node
- the operation node obtains information from a particular point in the source tree
- not shown in this illustration are built in behaviors copying nodes to the result tree



The diagram's action nodes are created from XSLT instructions.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

A predictable behavior for processors (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 to pull information from the source tree or to calculate values:

- `<xsl:copy-of>`
  - add to the result tree a copy of nodes from the source tree
- `<xsl:value-of>`
  - add to the result tree the evaluation of an expression or the value of a source tree node
- `<xsl:for-each>`
  - reposition to a selection of source tree nodes or values 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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Chapter 5 - Transformation environment

- Introduction - The transformation environment

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

The transformation environment

Chapter 5 - Transformation environment

Different ways are available to communicate to and from a processor

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

XPath 2 functions for diagnostics

- P2 `error()`
  - signaling a premature end of process
- P2 `trace()`
  - diagnostic reporting of function values

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

The transformation environment (cont.)

Chapter 5 - Transformation environment

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

Schemas and serialization:

- `<xsl:namespace-alias>`
  - specify a result tree namespace translation
- `<xsl:output>`
  - specify the desired serialization of the result tree
- T2 `<xsl:character-map>`
  - specify a translation of characters during serialization
- T2 `<xsl:output-character>`
  - specify a translation of single character during serialization
- T2 `<xsl:import-schema>`
  - gain the awareness of user-defined data types
- T2 `<xsl:result-document>`
  - create more than a single 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
- T2 `type-available()`
  - accessing system-defined property strings

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Chapter 6 - Transform and data management

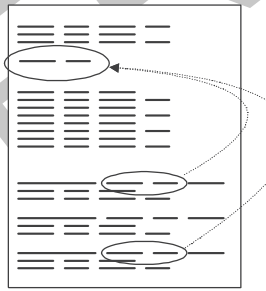
- Introduction - Why modularize logical and physical structures?

Fourteenth Edition - 2011-02-11 - ISB N 978-1-894049-24-5

Why modularize logical and physical structures?

Modularizing the logical structure supports development

- manipulating or reusing transform fragments within a given transform
- declaration and reuse of syntactic packages of transform logic and 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:

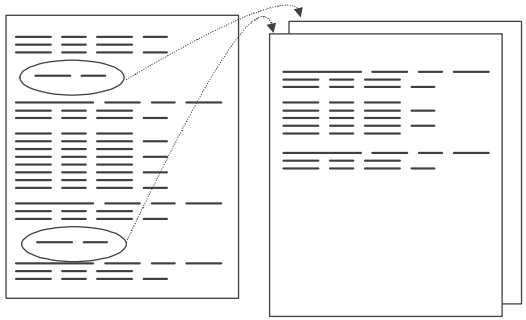
- XML internal general entities in XSLT stylesheets
- XML internal general entities in marked sections in external parameter entities
- variable bindings
- [1] user-defined functions
- XSLT named templates

Fourteenth Edition - 2011-02-11 - ISB N 978-1-894049-24-5

Why modularize logical and physical structures? (cont.)

Modularizing the physical structure of transforms supports reuse

- compartmentalization of code
- sharing and reuse of transform 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 processor implementation



This chapter overviews physical modularization using:

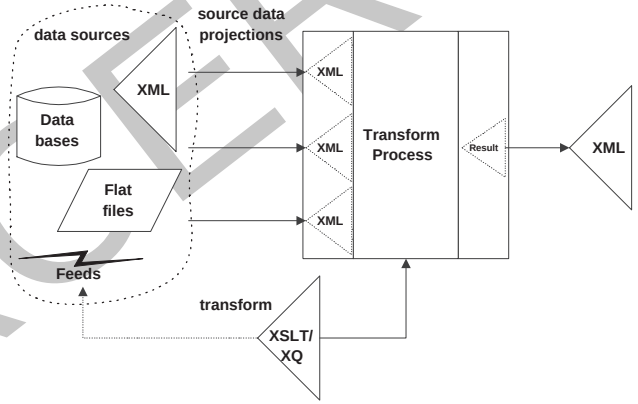
- XML external parsed general entities in XSLT stylesheets
- XSLT included and imported stylesheets
- extension functions
- XSLT extension elements (instructions)

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Why modularize logical and physical structures? (cont.)

Modularizing the data supports reuse

- compartmentalization of data
- focusing the responsibility of data to its most appropriate custodians
- sharing and reuse of content across an organization
- accessing content of different kinds through data projection



This chapter overviews physical modularization using:

- XML external unparsed entities
- T2 XPath functions for data access
- XSLT functions for data access

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Why modularize logical and physical structures? (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 as an instruction in XSLT
- T2 `<xsl:function>`
  - declare a function to be called by name as an subroutine in XPath
- T2 `<xsl:sequence>`
  - using XPath to express values returned by a function or a template
- `<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>`
  - override the importation of template rules
- T2 `<xsl:next-match>`
  - override the priority and importation of template rules
- `<xsl:fallback>`
  - accommodate the lack of implementation of an instruction element

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Why modularize logical and physical structures? (cont.)

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

Functions related to data modularization:

- P2 `collection()`
  - access to a collection of documents
- P2 `doc()`
  - access to multiple source documents
- P2 `doc-available()`
  - check for a document
- `document()`
  - access to multiple source documents
- T2 `unparsed-entity-public-id()`
  - finding the public identifier of an unparsed entity
- `unparsed-entity-uri()`
  - finding the URI of an unparsed entity
- T2 `unparsed-text()`
  - access to multiple documents
- T2 `unparsed-text-available()`
  - check for a document

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Chapter 7 - Data type expressions and functions

- Introduction - Data type expressions and functions

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Data type expressions and functions

Powerful functions and expression support

- this chapter describes functions and expressions to manipulate variables and values of data types
  - the XSLT specification includes facilities implementing 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 operators
  - P2 sequence functions and operator
  - P2 date and time functions and operators
  - P2 item functions and operators
- P2 regular expressions
- T2 string analysis
- access to the source node tree
  - de-referencing pointers between information items
  - setting up lookup tables to tree nodes

This training material assumes `xmlns:xs="http://www.w3.org/2001/XMLSchema"` when referencing W3C Schema data types

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.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Data type expressions and functions (cont.)

The XPath keywords covered in this chapter are as follows.

- `|`
  - union operator
- `union`
  - union operator
- `intersect`
  - intersection operator
- `except`
  - exception operator
- `to`
  - create a sequence of integers
- `instance of`
  - testing the type of an item
- `castable as`
  - testing the conversion of an item
- `cast as`
  - converting an item to the given type
- `treat as`
  - validating an item as a given type
- `or`
  - boolean operator
- `and`
  - boolean operator
- `is << >>`
  - document order comparison operators
- `eq ne lt le gt ge`
  - singleton value comparison operators
- `= != < <= > >=`
  - value comparison operators
- `some every`
  - quantified expressions

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Data type expressions and functions (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 string analysis and regular expressions:

- `<xsl:analyze-string>`
  - determine the matching components in an analysis of a string
- `<xsl:matching-substring>`
  - act on matching components from the analysis of a string
- `<xsl:non-matching-substring>`
  - act on non-matching components from the analysis of a string

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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Data type expressions and functions (cont.)

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

Functions related to number data types:

- 2 `abs()`
  - returning the absolute value
- `ceiling()`
  - rounding a number up
- `floor()`
  - rounding a number down
- `number()`
  - casting an argument to a number
- `round()`
  - rounding a number
- 2 `round-half-to-even()`
  - rounding a number

Functions related to string data types:

- 2 `codepoint-equal()`
  - Unicode string comparison
- 2 `codepoints-to-string()`
  - Unicode string conversion
- 2 `compare()`
  - string comparison
- `concat()`
  - string concatenation
- `contains()`
  - string detection

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Practical Transformation Using XSLT and XPath

Data type expressions and functions (cont.)

Chapter 7 - Data type expressions and functions

Functions related to string data types (cont.):

- 2 `default-collation()`
  - obtaining the default collation
- 2 `ends-with()`
  - establish the presence of a string
- `format-number()`
  - adding punctuation and controlling number display
- 2 `lower-case()`
  - string case folding
- 2 `matches()`
  - regular expression matching
- `normalize-space()`
  - normalizing extraneous spaces in a string
- 2 `normalize-unicode()`
  - normalizing Unicode characters in a string
- 2 `replace()`
  - regular expression replacement
- `starts-with()`
  - establishing the presence of a string
- `string()`
  - casting an argument to a string
- 2 `string-join()`
  - join a sequence of strings into a single string
- `string-length()`
  - finding the length of a string
- 2 `string-to-codepoints()`
  - Unicode string conversion
- `substring()`
  - returning a portion of a string
- `substring-after()`
  - returning a portion of a string
- `substring-before()`
  - returning a portion of a string
- 2 `tokenize()`
  - regular expression tokenizing
- `translate()`
  - translating characters found in a string
- 2 `upper-case()`
  - string case folding

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Information subject to restrictive legend on first page.

Data type expressions and functions (cont.)

Functions related to sequences:

- P2 `avg()`
  - return the average of members of a numerical sequence
- `count()`
  - return a count of members of the sequence
- P2 `deep-equal()`
  - return an indication of two sequences being identical
- P2 `distinct-values()`
  - return a sequence with duplicate members removed
- P2 `empty()`
  - return an indication of the sequence being empty
- P2 `exactly-one()`
  - return an indication of the cardinality of a sequence
- P2 `exists()`
  - return an indication of the sequence not being empty
- P2 `index-of()`
  - return index pointers into a sequence
- P2 `insert-before()`
  - return a sequence with members inserted
- P2 `max()`
  - return the maximum value of the members of the numeric sequence
- P2 `min()`
  - return the minimum value of the members of the numeric sequence
- P2 `one-or-more()`
  - return an indication of the cardinality of a sequence
- P2 `remove()`
  - return a sequence with a member removed
- P2 `reverse()`
  - return the reverse of a sequence
- P2 `subsequence()`
  - return a portion of a sequence
- `sum()`
  - return a sum of sequence members
- P2 `unordered()`
  - return an unordered sequence
- P2 `zero-or-one()`
  - return an indication of the cardinality of a sequence

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Data type expressions and functions (cont.)

Functions related to date and time:

- P2 `adjust-date-to-timezone()`
  - return adjusted date
- P2 `adjust-dateTime-to-timezone()`
  - return adjusted date and time
- P2 `adjust-time-to-timezone()`
  - return adjusted time
- P2 `current-date()`
  - return date/time component
- P2 `current-dateTime()`
  - return date/time component
- P2 `current-time()`
  - return date/time component
- P2 `dateTime()`
  - return date/time component
- P2 `day-from-date()`
  - return date/time component
- P2 `day-from-dateTime()`
  - return date/time component
- P2 `days-from-duration()`
  - return date/time component
- T2 `format-date()`
  - format date string
- T2 `format-dateTime()`
  - format date and time string
- T2 `format-time()`
  - format time string
- P2 `hours-from-dateTime()`
  - return date/time component
- P2 `hours-from-time()`
  - return time component
- P2 `hours-from-duration()`
  - return date/time component
- P2 `implicit-timezone()`
  - return date/time component

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Data type expressions and functions (cont.)

Functions related to date and time (cont.):

- `minutes-from-dateTime()`
  - return date/time component
- `minutes-from-duration()`
  - return date/time component
- `minutes-from-time()`
  - return date/time component
- `month-from-date()`
  - return date/time component
- `month-from-dateTime()`
  - return date/time component
- `months-from-duration()`
  - return date/time component
- `seconds-from-dateTime()`
  - return date/time component
- `seconds-from-duration()`
  - return date/time component
- `seconds-from-time()`
  - return date/time component
- `timezone-from-date()`
  - return date/time component
- `timezone-from-time()`
  - return date/time component
- `timezone-from-dateTime()`
  - return date/time component
- `year-from-date()`
  - return date/time component
- `year-from-dateTime()`
  - return date/time component
- `years-from-duration()`
  - return date/time component

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Data type expressions and functions (cont.)

Functions related to node data types:

- `base-uri()`
  - obtaining a node's base URI
- `data()`
  - obtaining a node's data
- `document-uri()`
  - obtaining a node's document URI
- `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
- `nilled()`
  - obtaining a node's nilled status
- `node-name()`
  - obtaining a node name
- `root()`
  - obtaining the root node of a tree
- `static-base-uri()`
  - obtaining the static base URI

Qualified name functions:

- `in-scope-prefixes()`
  - return a set of prefixes
- `local-name-from-QName()`
  - return a local name
- `namespace-uri-for-prefix()`
  - return a namespace URI
- `namespace-uri-from-QName()`
  - return a namespace URI
- `prefix-from-QName()`
  - return a namespace prefix
- `QName()`
  - return a qualified name
- `resolve-QName()`
  - resolve a qualified name

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Data type expressions and functions (cont.)

Other functions:

- `current()`
  - current node access
- P2 `encode-for-uri()`
  - return an encoded string
- P2 `escape-html-uri()`
  - return an encoded string
- `id()`
  - accessing ID values in source node trees
- P2 `idref()`
  - accessing references to ID values in source node trees
- P2 `iri-to-uri()`
  - return an encoded string
- `key()`
  - accessing key nodes from key tables
- T2 `regex-group()`
  - regular expression group retrieval
- P2 `resolve-uri()`
  - return an absolute URI

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Chapter 8 - Constructing the result tree

- Introduction - Constructing result-tree nodes

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Constructing result-tree nodes

Result-tree nodes are used both in the result tree and in the transformation

- the creation of the result tree
- T1 creating a result-tree fragment
- T2 creating a temporary tree

Recall the earlier processing model diagram (page 79)

- the diagram depicts the copying of nodes from the operation tree and the source tree to the result tree
  - the operation tree nodes that are copied to the result tree are the literal result elements
- also possible to explicitly add nodes of different types to the result tree

XSLT supports:

- direct construction of result tree nodes
- different ways to copy nodes from the source tree to the result tree
- constructing text nodes in the result tree reflecting numbering information found in the source tree

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Constructing result-tree nodes (cont.)

The XSLT instructions covered in this chapter are as follows.

Instructions related to building the result tree:

- `<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
- T2 `<xsl:document>`
  - instantiate a document node in the result tree
- `<xsl:element>`
  - instantiate an element node in the result tree
- T2 `<xsl:namespace>`
  - instantiate a namespace 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
- `<xsl:number>`
  - add a string to the result tree representing the position of the current node

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Chapter 9 - Sorting and grouping

- Introduction - Sorting and grouping

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Sorting and grouping

This chapter covers how to arrange the construction of results in ordered fashion.

Sorting

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

- designed for sorting the context list using multiple criteria
  - each criterion is a single value calculated for each node
  - value may be simple node value or may be any XPath evaluation relative to node
- the source tree is untouched during sorting
  - items being sorted are selected from the tree and the selection itself is sorted, not the tree
- sorting can be language based, numeric based, type-based or based on custom semantics
- the context list can be any arbitrary sequence
- multiple keys are used to sort clumps of equal values by other values
  - e.g. a secondary key is used when the primary key finds clumps of equal values
    - the secondary key is only applied to the clumped values, maintaining the set of clumped values in the same position of the primary sort
  - e.g. a tertiary key is used when the secondary key finds clumps of equal values
    - the tertiary key is only applied to the clumped values after the secondary sort, maintaining the set of clumped values in the same position of the secondary sort, in the same position of the primary sort
  - leftover clumped values after all keys are accommodated are left in context list sequence order

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Sorting and grouping (cont.)

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":

- collecting information while separating and grouping it by common values
  - i.e. grouping the clumps under the value that created the clump
  - selecting a single piece of composite information obtains all components
  - a simple sort doesn't partition the composite information into constituent pieces
- 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
- 1 no explicit support for grouping under duplicate source tree node values
- 2 explicit support for grouping under duplicate source tree node values

1 This chapter covers three techniques of using XSLT 1.0 to group constructs when processing:

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

2 This chapter also covers the built-in XSLT 2.0 facility for grouping

- `group-adjacent`, `group-by`, `group-starting-with` and `group-ending-with`

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Sorting and grouping (cont.)

The XSLT instructions covered in this chapter are:

- `<xsl:sort>`
  - specify a criterion with which to sort a set of nodes
- 2 `<xsl:perform-sort>`
  - specify the criteria with which to sort a sequence of items
- 2 `<xsl:for-each-group>`
  - act on a set of items according to grouping criteria

The XSLT functions covered in this chapter are:

- 2 `current-grouping-key()`
  - returns the value by which members of the current set of grouped items are grouped
- 2 `current-group()`
  - returns the members of the current set of grouped items

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Annex A - XML to HTML transformation

- Introduction - Historical web standards for presentation

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Historical web standards for presentation

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

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

Extensible Hypertext Markup Language (XHTML)

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

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`.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Annex B - XSL formatting semantics introduction

- Introduction - Formatting objectives

Outcomes:

- awareness of the formatting objectives of the XSL development committee

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Formatting objectives

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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

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
  - same architecture as when producing HTML from XML
  - an XSL-FO engine interprets an XSL-FO instance just as a browser interprets HTML
- 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
- recall the XSL-FO engine incorporating XSLT (page 39)

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.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Annex C - Instruction, function and grammar summaries

- Introduction - Quick summaries
- Section 1 - Vocabulary, functions and grammars XSLT 1.0 and XPath 1.0
- Section 2 - Vocabulary, functions and grammars XSLT 2.0 and XPath 2.0

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Quick summaries

Annex C - Instruction, function and grammar summaries

This annex lists alphabetized references to the components of the specifications. Each entry notes the chapter in this book where the construct is primarily described.

The specifications are rigorous references to all of the facilities and functions:

XSLT 1.0/XPath 1.0:

- `http://www.w3.org/TR/1999/REC-xslt-19991116`
- `http://www.w3.org/TR/1999/REC-xpath-19991116`

XSLT 2.0/XPath 2.0/XQuery 1.0

- `http://www.w3.org/TR/2007/REC-xslt20-20070123/`
- `http://www.w3.org/TR/2007/REC-xpath20-20070123/`
- `http://www.w3.org/TR/2007/REC-xpath-datamodel-20070123/`
- `http://www.w3.org/TR/2007/REC-xpath-functions-20070123/`
- `http://www.w3.org/TR/2007/REC-xslt-xquery-serialization-20070123/`
- `http://www.w3.org/TR/2007/REC-xquery-20070123/`
- `http://www.w3.org/TR/2007/REC-xquery-semantics-20070123/`
- `http://www.w3.org/TR/2007/REC-xqueryx-20070123`

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLT 1.0 element summary

Annex C - Instruction, function and grammar summaries
Section 1 - Vocabulary, functions and grammars XSLT 1.0 and XPath 1.0

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 structures? (page 88)

- XSLT 1.0 5.6 Overriding Template Rules
-
```
<xsl:apply-imports/>
```

`apply-templates` (instruction) - A predictable behavior for processors (page 80)

- XSLT 1.0 5.4 Applying Template Rules
-
```
<xsl:apply-templates mode="qname"?
                     select="node-set-expression"?>
  (<xsl:sort>|<xsl:with-param>)*
</xsl:apply-templates>
```

`attribute` (instruction) - Constructing result-tree nodes (page 103)

- XSLT 1.0 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) - Constructing result-tree nodes (page 103)

- XSLT 1.0 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 structures? (page 88)

- XSLT 1.0 6 Named Templates
-
```
<xsl:call-template name="qname">
  <xsl:with-param>*
</xsl:call-template>
```

`choose` (instruction) - A predictable behavior for processors (page 80)

- XSLT 1.0 9.2 Conditional Processing with `xsl:choose`
-
```
<xsl:choose>
  (<xsl:when>+,<xsl:otherwise>?)
</xsl:choose>
```

`comment` (instruction) - Constructing result-tree nodes (page 103)

- XSLT 1.0 7.4 Creating Comments
-
```
<xsl:comment>
  template
</xsl:comment>
```

`copy` (instruction) - Constructing result-tree nodes (page 103)

- XSLT 1.0 7.5 Copying
-
```
<xsl:copy use-attribute-sets="qnames"?>
  template
</xsl:copy>
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

`copy-of` (instruction) - Constructing result-tree nodes (page 103)

- XSLT 1.0 11.3 Using Values of Variables and Parameters with `xsl:copy-of`
-
```
<xsl:copy-of select="expression"/>
```

`decimal-format` (top level element) - Data type expressions and functions (page 93)

- XSLT 1.0 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) - Constructing result-tree nodes (page 103)

- XSLT 1.0 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 structures? (page 88)

- XSLT 1.0 15 Fallback
-
```
<xsl:fallback>
  template
</xsl:fallback>
```

`for-each` (instruction) - A predictable behavior for processors (page 80)

- XSLT 1.0 8 Repetition
-
```
<xsl:for-each select="node-set-expression">
  (<xsl:sort>*,template)
</xsl:for-each>
```

`if` (instruction) - A predictable behavior for processors (page 80)

- XSLT 1.0 9.1 Conditional Processing with `xsl:if`
-
```
<xsl:if test="boolean-expression">
  template
</xsl:if>
```

`import` - Why modularize logical and physical structures? (page 88)

- XSLT 1.0 2.6 Combining Stylesheets
-
```
<xsl:import href="uri-reference"/>
```

`include` (top level element) - Why modularize logical and physical structures? (page 88)

- XSLT 1.0 2.6 Combining Stylesheets
-
```
<xsl:include href="uri-reference"/>
```

`key` (top level element) - Data type expressions and functions (page 93)

- XSLT 1.0 12.2 Keys
-
```
<xsl:key match="pattern"
         name="qname"
         use="expression"/>
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

`message` (instruction) - The transformation environment (page 83)

- XSLT 1.0 13 Messages
-
```
<xsl:message terminate="yes|no"?>
  template
</xsl:message>
```

`namespace-alias` (top level element) - The transformation environment (page 83)

- XSLT 1.0 7.1 Creating Elements and Attributes
-
```
<xsl:namespace-alias result-prefix="prefix|#default"
                     stylesheet-prefix="prefix|#default"/>
```

`number` (instruction) - Constructing result-tree nodes (page 103)

- XSLT 1.0 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` - A predictable behavior for processors (page 80)

- XSLT 1.0 9.2 Conditional Processing with `xsl:choose`
-
```
<xsl:otherwise>
  template
</xsl:otherwise>
```

`output` (top level element) - The transformation environment (page 83)

- XSLT 1.0 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 structures? (page 88)

- XSLT 1.0 11 Variables and Parameters
-
```
<xsl:param name="qname"
           select="expression"?>
  template
</xsl:param>
```

`preserve-space` (top level element) - XPath data model (page 76)

- XSLT 1.0 3.4 Whitespace Stripping
-
```
<xsl:preserve-space elements="tokens"/>
```

`processing-instruction` (instruction) - Constructing result-tree nodes (page 103)

- XSLT 1.0 7.3 Creating Processing Instructions
-
```
<xsl:processing-instruction name="ncname|{string-expression}">
  template
</xsl:processing-instruction>
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

`sort` - Sorting and grouping (page 107)

- XSLT 1.0 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) - XPath data model (page 76)

- XSLT 1.0 3.4 Whitespace Stripping
- ```
  <xsl:strip-space elements="tokens"/>
  ```

`stylesheet` - The transformation environment (page 83)

- XSLT 1.0 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 processors (page 80)

- XSLT 1.0 5.3 Defining Template Rules
- ```
  <xsl:template match="pattern"?
                mode="qname"?
                name="qname"?
                priority="number"?>
    (<xsl:param>*,template)
  </xsl:template>
  ```

`text` (instruction) - Constructing result-tree nodes (page 103)

- XSLT 1.0 7.2 Creating Text
- ```
  <xsl:text disable-output-escaping="yes|no"?>
    #PCDATA
  </xsl:text>
  ```

`transform` - The transformation environment (page 83)

- XSLT 1.0 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 processors (page 80)

- XSLT 1.0 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 structures? (page 88)

- XSLT 1.0 11 Variables and Parameters
- ```
  <xsl:variable name="qname"
                select="expression"?>
    template
  </xsl:variable>
  ```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Practical Transformation Using XSLT and XPath

`when` - A predictable behavior for processors (page 80)

- XSLT 1.0 9.2 Conditional Processing with `xsl:choose`
- ```
  <xsl:when test="boolean-expression">
    template
  </xsl:when>
  ```

`with-param` - Why modularize logical and physical structures? (page 88)

- XSLT 1.0 11.6 Passing Parameters to Templates
- ```
  <xsl:with-param name="qname"
                  select="expression"?>
    template
  </xsl:with-param>
  ```


Copyright © Crane Softwrights Ltd.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Information subject to restrictive legend on first page.

XPath 1.0 and XSLT 1.0 function summary

Annex C - Instruction, function and grammar summaries
Section 1 - Vocabulary, functions and grammars XSLT 1.0 and XPath 1.0

All functions of both XPath 1.0 and XSLT 2.0 in alphabetical order follow. This information is mechanically derived from the XPath 1.0 and XSLT 1.0 Recommendations.

`boolean` - Data type expressions and functions (page 94)
- XPath 1.0 4.3 Boolean Functions
- *`boolean`* `boolean(` *`object`* `)`

`ceiling` - Data type expressions and functions (page 94)
- XPath 1.0 4.4 Number Functions
- *`number`* `ceiling(` *`number`* `)`

`concat` - Data type expressions and functions (page 94)
- XPath 1.0 4.2 String Functions
- *`string`* `concat(` *`string`*`,` *`string`*`,` *`string*`* `)`

`contains` - Data type expressions and functions (page 94)
- XPath 1.0 4.2 String Functions
- *`boolean`* `contains(` *`string`*`,` *`string`* `)`

`count` - Data type expressions and functions (page 96)
- XPath 1.0 4.1 Node Set Functions
- *`number`* `count(` *`node-set`* `)`

`current` - Data type expressions and functions (page 100)
- XSLT 1.0 12.4 Miscellaneous Additional Functions
- *`node-set`* `current( )`

`document` - Why modularize logical and physical structures? (page 89)
- XSLT 1.0 12.1 Multiple Source Documents
- *`node-set`* `document(` *`object`*`,` *`node-set?`* `)`

`element-available` - Why modularize logical and physical structures? (page 89)
- XSLT 1.0 15 Fallback
- *`boolean`* `element-available(` *`string`* `)`

`false` - Data type expressions and functions (page 94)
- XPath 1.0 4.3 Boolean Functions
- *`boolean`* `false( )`

`floor` - Data type expressions and functions (page 94)
- XPath 1.0 4.4 Number Functions
- *`number`* `floor(` *`number`* `)`

`format-number` - Data type expressions and functions (page 95)
- XSLT 1.0 12.3 Number Formatting
- *`string`* `format-number(` *`number`*`,` *`string`*`,` *`string?`* `)`

`function-available` - Why modularize logical and physical structures? (page 89)
- XSLT 1.0 15 Fallback
- *`boolean`* `function-available(` *`string`* `)`

`generate-id` - Data type expressions and functions (page 99)
- XSLT 1.0 12.4 Miscellaneous Additional Functions
- *`string`* `generate-id(` *`node-set?`* `)`

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

`id` - Data type expressions and functions (page 100)
- XPath 1.0 4.1 Node Set Functions
- *`node-set`* `id(` *`object`* `)`

`key` - Data type expressions and functions (page 100)
- XSLT 1.0 12.2 Keys
- *`node-set`* `key(` *`string`*`,` *`object`* `)`

`lang` - Data type expressions and functions (page 94)
- XPath 1.0 4.3 Boolean Functions
- *`boolean`* `lang(` *`string`* `)`

`last` - XPath data model (page 76)
- XPath 1.0 4.1 Node Set Functions
- *`number`* `last( )`

`local-name` - Data type expressions and functions (page 99)
- XPath 1.0 4.1 Node Set Functions
- *`string`* `local-name(` *`node-set?`* `)`

`name` - Data type expressions and functions (page 99)
- XPath 1.0 4.1 Node Set Functions
- *`string`* `name(` *`node-set?`* `)`

`namespace-uri` - Data type expressions and functions (page 99)
- XPath 1.0 4.1 Node Set Functions
- *`string`* `namespace-uri(` *`node-set?`* `)`

`normalize-space` - Data type expressions and functions (page 95)
- XPath 1.0 4.2 String Functions
- *`string`* `normalize-space(` *`string?`* `)`

`not` - Data type expressions and functions (page 94)
- XPath 1.0 4.3 Boolean Functions
- *`boolean`* `not(` *`boolean`* `)`

`number` - Data type expressions and functions (page 94)
- XPath 1.0 4.4 Number Functions
- *`number`* `number(` *`object?`* `)`

`position` - XPath data model (page 76)
- XPath 1.0 4.1 Node Set Functions
- *`number`* `position( )`

`round` - Data type expressions and functions (page 94)
- XPath 1.0 4.4 Number Functions
- *`number`* `round(` *`number`* `)`

`starts-with` - Data type expressions and functions (page 95)
- XPath 1.0 4.2 String Functions
- *`boolean`* `starts-with(` *`string`*`,` *`string`* `)`

`string` - Data type expressions and functions (page 95)
- XPath 1.0 4.2 String Functions
- *`string`* `string(` *`object?`* `)`

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

`string-length` - Data type expressions and functions (page 95)
- XPath 1.0 4.2 String Functions
- *`number`* `string-length(` *`string?`* `)`

`substring` - Data type expressions and functions (page 95)
- XPath 1.0 4.2 String Functions
- *`string`* `substring(` *`string`*`,` *`number`*`,` *`number?`* `)`

`substring-after` - Data type expressions and functions (page 95)
- XPath 1.0 4.2 String Functions
- *`string`* `substring-after(` *`string`*`,` *`string`* `)`

`substring-before` - Data type expressions and functions (page 95)
- XPath 1.0 4.2 String Functions
- *`string`* `substring-before(` *`string`*`,` *`string`* `)`

`sum` - Data type expressions and functions (page 96)
- XPath 1.0 4.4 Number Functions
- *`number`* `sum(` *`node-set`* `)`

`system-property` - The transformation environment (page 83)
- XSLT 1.0 12.4 Miscellaneous Additional Functions
- *`object`* `system-property(` *`string`* `)`

`translate` - Data type expressions and functions (page 95)
- XPath 1.0 4.2 String Functions
- *`string`* `translate(` *`string`*`,` *`string`*`,` *`string`* `)`

`true` - Data type expressions and functions (page 94)
- XPath 1.0 4.3 Boolean Functions
- *`boolean`* `true( )`

`unparsed-entity-uri` - Why modularize logical and physical structures? (page 89)
- XSLT 1.0 12.4 Miscellaneous Additional Functions
- *`string`* `unparsed-entity-uri(` *`string`* `)`

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XPath 1.0 grammar productions

Annex C - Instruction, function and grammar summaries
Section 1 - Vocabulary, functions and grammars XSLT 1.0 and XPath 1.0

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)

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

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]
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

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]
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

XSLT 1.0 grammar productions

Annex C - Instruction, function and grammar summaries
Section 1 - Vocabulary, functions and grammars XSLT 1.0 and XPath 1.0

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-1.0-29] ')'
                   | 'key' '(' Literal[XPath-1.0-29] ',' Literal[XPath-1.0-29]
                   ')'
[4] RelativePathPattern ::= StepPattern[5]
                          | RelativePathPattern[4] '/' StepPattern[5]
                          | RelativePathPattern[4] '//' StepPattern[5]
[5] StepPattern ::= ChildOrAttributeAxisSpecifier[6] NodeTest[XPath-1.0-7]
                  Predicate[XPath-1.0-8]*
[6] ChildOrAttributeAxisSpecifier ::= AbbreviatedAxisSpecifier[XPath-1.0-13]
                                    | ('child' | 'attribute') '::'
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLT 2.0 element summary

Annex C - Instruction, function and grammar summaries
Section 2 - Vocabulary, functions and grammars XSLT 2.0 and XPath 2.0

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.

`analyze-string` - Data type expressions and functions (page 93)

- XSLT 2.0 - 15.1 The xsl:analyze-string instruction

```
- <!-- Category: instruction -->
  <xsl:analyze-string
    select = expression
    regex = { string }
    flags? = { string }>
   <!-- Content: (xsl:matching-substring?, xsl:non-matching-substring?,
   xsl:fallback*) -->
  </xsl:analyze-string>
```

`apply-imports` - Why modularize logical and physical structures? (page 88)

- XSLT 2.0 - 6.7 Overriding Template Rules

```
- <!-- Category: instruction -->
  <xsl:apply-imports>
    <!-- Content: xsl:with-param* -->
  </xsl:apply-imports>
```

`apply-templates` - A predictable behavior for processors (page 80)

- XSLT 2.0 - 6.3 Applying Template Rules

```
- <!-- Category: instruction -->
  <xsl:apply-templates
    select? = expression
    mode? = token>
    <!-- Content: (xsl:sort | xsl:with-param)* -->
  </xsl:apply-templates>
```

`attribute` - Constructing result-tree nodes (page 103)

- XSLT 2.0 - 11.3 Creating Attribute Nodes Using xsl:attribute

```
- <!-- Category: instruction -->
  <xsl:attribute
    name = { qname }
    namespace? = { uri-reference }
    select? = expression
    separator? = { string }
    type? = qname
    validation? = "strict" | "lax" | "preserve" | "strip">
    <!-- Content: sequence-constructor -->
  </xsl:attribute>
```

`attribute-set` - Constructing result-tree nodes (page 103)

- XSLT 2.0 - 10.2 Named Attribute Sets

```
- <!-- Category: declaration -->
  <xsl:attribute-set
    name = qname
    use-attribute-sets? = qnames>
    <!-- Content: xsl:attribute* -->
  </xsl:attribute-set>
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

`call-template` - Why modularize logical and physical structures? (page 88)

- XSLT 2.0 - 10.1 Named Templates
- ```
  <!-- Category: instruction -->
  <xsl:call-template
    name = qname>
    <!-- Content: xsl:with-param* -->
  </xsl:call-template>
  ```

`character-map` - The transformation environment (page 83)

- XSLT 2.0 - 20.1 Character Maps
- ```
  <!-- Category: declaration -->
  <xsl:character-map
    name = qname
    use-character-maps? = qnames>
    <!-- Content: (xsl:output-character*) -->
  </xsl:character-map>
  ```

`choose` - A predictable behavior for processors (page 80)

- XSLT 2.0 - 8.2 Conditional Processing with xsl:choose
- ```
  <!-- Category: instruction -->
  <xsl:choose>
    <!-- Content: (xsl:when+, xsl:otherwise?) -->
  </xsl:choose>
  ```

`comment` - Constructing result-tree nodes (page 103)

- XSLT 2.0 - 11.8 Creating Comments
- ```
  <!-- Category: instruction -->
  <xsl:comment
    select? = expression>
    <!-- Content: sequence-constructor -->
  </xsl:comment>
  ```

`copy` - Constructing result-tree nodes (page 103)

- XSLT 2.0 - 11.9.1 Shallow Copy
- ```
  <!-- Category: instruction -->
  <xsl:copy
    copy-namespaces? = "yes" | "no"
    inherit-namespaces? = "yes" | "no"
    use-attribute-sets? = qnames
    type? = qname
    validation? = "strict" | "lax" | "preserve" | "strip">
    <!-- Content: sequence-constructor -->
  </xsl:copy>
  ```

`copy-of` - Constructing result-tree nodes (page 103)

- XSLT 2.0 - 11.9.2 Deep Copy
- ```
  <!-- Category: instruction -->
  <xsl:copy-of
    select = expression
    copy-namespaces? = "yes" | "no"
    type? = qname
    validation? = "strict" | "lax" | "preserve" | "strip" />
  ```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

`decimal-format` - Data type expressions and functions (page 93)

- XSLT 2.0 - 16.4.1 Defining a Decimal Format
- ```
  <!-- Category: declaration -->
  <xsl:decimal-format
    name? = qname
    decimal-separator? = char
    grouping-separator? = char
    infinity? = string
    minus-sign? = char
    NaN? = string
    percent? = char
    per-mille? = char
    zero-digit? = char
    digit? = char
    pattern-separator? = char />
  ```

`document` - Constructing result-tree nodes (page 103)

- XSLT 2.0 - 11.5 Creating Document Nodes
- ```
  <!-- Category: instruction -->
  <xsl:document
    validation? = "strict" | "lax" | "preserve" | "strip"
    type? = qname>
    <!-- Content: sequence-constructor -->
  </xsl:document>
  ```

`element` - Constructing result-tree nodes (page 103)

- XSLT 2.0 - 11.2 Creating Element Nodes Using xsl:element
- ```
  <!-- Category: instruction -->
  <xsl:element
    name = { qname }
    namespace? = { uri-reference }
    inherit-namespaces? = "yes" | "no"
    use-attribute-sets? = qnames
    type? = qname
    validation? = "strict" | "lax" | "preserve" | "strip">
    <!-- Content: sequence-constructor -->
  </xsl:element>
  ```

`fallback` - Why modularize logical and physical structures? (page 88)

- XSLT 2.0 - 18.2.3 Fallback
- ```
  <!-- Category: instruction -->
  <xsl:fallback>
    <!-- Content: sequence-constructor -->
  </xsl:fallback>
  ```

`for-each` - A predictable behavior for processors (page 80)

- XSLT 2.0 - 7 Repetition
- ```
  <!-- Category: instruction -->
  <xsl:for-each
    select = expression>
    <!-- Content: (xsl:sort*, sequence-constructor) -->
  </xsl:for-each>
  ```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
  ```

`for-each-group` - Sorting and grouping (page 107)

- XSLT 2.0 - 14.3 The xsl:for-each-group Element

```
- <!-- Category: instruction -->
  <xsl:for-each-group
    select = expression
    group-by? = expression
    group-adjacent? = expression
    group-starting-with? = pattern
    group-ending-with? = pattern
    collation? = { uri }>
    <!-- Content: (xsl:sort*, sequence-constructor) -->
  </xsl:for-each-group>
```

`function` - Why modularize logical and physical structures? (page 88)

- XSLT 2.0 - 10.3 Stylesheet Functions

```
- <!-- Category: declaration -->
  <xsl:function
    name = qname
    as? = sequence-type
    override? = "yes" | "no">
    <!-- Content: (xsl:param*, sequence-constructor) -->
  </xsl:function>
```

`if` - A predictable behavior for processors (page 80)

- XSLT 2.0 - 8.1 Conditional Processing with xsl:if

```
- <!-- Category: instruction -->
  <xsl:if
    test = expression>
    <!-- Content: sequence-constructor -->
  </xsl:if>
```

`import` - Why modularize logical and physical structures? (page 88)

- XSLT 2.0 - 3.10.3 Stylesheet Import

```
- <!-- Category: declaration -->
  <xsl:import
    href = uri-reference />
```

`import-schema` - The transformation environment (page 83)

- XSLT 2.0 - 3.14 Importing Schema Components

```
- <!-- Category: declaration -->
  <xsl:import-schema
    namespace? = uri-reference
    schema-location? = uri-reference>
    <!-- Content: xs:schema? -->
  </xsl:import-schema>
```

`include` - Why modularize logical and physical structures? (page 88)

- XSLT 2.0 - 3.10.2 Stylesheet Inclusion

```
- <!-- Category: declaration -->
  <xsl:include
    href = uri-reference />
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Practical Transformation Using XSLT and XPath

`key` - Data type expressions and functions (page 93)

- XSLT 2.0 - 16.3.1 The xsl:key Declaration

```
- <!-- Category: declaration -->
  <xsl:key
    name = qname
    match = pattern
    use? = expression
    collation? = uri>
    <!-- Content: sequence-constructor -->
  </xsl:key>
```

`matching-substring` - Data type expressions and functions (page 93)

- XSLT 2.0 - 15.1 The xsl:analyze-string instruction

```
- <xsl:matching-substring>
    <!-- Content: sequence-constructor -->
  </xsl:matching-substring>
```

`message` - The transformation environment (page 83)

- XSLT 2.0 - 17 Messages

```
- <!-- Category: instruction -->
  <xsl:message
    select? = expression
    terminate? = { "yes" | "no" }>
    <!-- Content: sequence-constructor -->
  </xsl:message>
```

`namespace` - Constructing result-tree nodes (page 103)

- XSLT 2.0 - 11.7 Creating Namespace Nodes

```
- <!-- Category: instruction -->
  <xsl:namespace
    name = { ncname }
    select? = expression>
    <!-- Content: sequence-constructor -->
  </xsl:namespace>
```

`namespace-alias` - The transformation environment (page 83)

- XSLT 2.0 - 11.1.4 Namespace Aliasing

```
- <!-- Category: declaration -->
  <xsl:namespace-alias
    stylesheet-prefix = prefix | "#default"
    result-prefix = prefix | "#default" />
```

`next-match` - Why modularize logical and physical structures? (page 88)

- XSLT 2.0 - 6.7 Overriding Template Rules

```
- <!-- Category: instruction -->
  <xsl:next-match>
    <!-- Content: (xsl:with-param | xsl:fallback)* -->
  </xsl:next-match>
```

`non-matching-substring` - Data type expressions and functions (page 93)

- XSLT 2.0 - 15.1 The xsl:analyze-string instruction

```
- <xsl:non-matching-substring>
    <!-- Content: sequence-constructor -->
  </xsl:non-matching-substring>
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Information subject to restrictive legend on first page.

`number` - Constructing result-tree nodes (page 103)

- XSLT 2.0 - 12 Numbering
- ```
  <!-- Category: instruction -->
  <xsl:number
    value? = expression
    select? = expression
    level? = "single" | "multiple" | "any"
    count? = pattern
    from? = pattern
    format? = { string }
    lang? = { nmtoken }
    letter-value? = { "alphabetic" | "traditional" }
    ordinal? = { string }
    grouping-separator? = { char }
    grouping-size? = { number
  } />
  ```

`otherwise` - A predictable behavior for processors (page 80)

- XSLT 2.0 - 8.2 Conditional Processing with xsl:choose
- ```
  <xsl:otherwise>
    <!-- Content: sequence-constructor -->
  </xsl:otherwise>
  ```

`output` - The transformation environment (page 83)

- XSLT 2.0 - 20 Serialization
- ```
  <!-- Category: declaration -->
  <xsl:output
    name? = qname
    method? = "xml" | "html" | "xhtml" | "text" | qname-but-not-ncname
    byte-order-mark? = "yes" | "no"
    cdata-section-elements? = qnames
    doctype-public? = string
    doctype-system? = string
    encoding? = string
    escape-uri-attributes? = "yes" | "no"
    include-content-type? = "yes" | "no"
    indent? = "yes" | "no"
    media-type? = string
    normalization-form? = "NFC" | "NFD" | "NFKC" | "NFKD" |
  "fully-normalized" | "none" | nmtoken
    omit-xml-declaration? = "yes" | "no"
    standalone? = "yes" | "no" | "omit"
    undeclare-prefixes? = "yes" | "no"
    use-character-maps? = qnames
    version? = nmtoken />
  ```

`output-character` - The transformation environment (page 83)

- XSLT 2.0 - 20.1 Character Maps
- ```
  <xsl:output-character
    character = char
    string = string />
  ```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

`param` - Why modularize logical and physical structures? (page 88)

- XSLT 2.0 - 9.2 Parameters
- ```
  <!-- Category: declaration -->
  <xsl:param
    name = qname
    select? = expression
    as? = sequence-type
    required? = "yes" | "no"
    tunnel? = "yes" | "no">
    <!-- Content: sequence-constructor -->
  </xsl:param>
  ```

`perform-sort` - Sorting and grouping (page 107)

- XSLT 2.0 - 13.2 Creating a Sorted Sequence
- ```
  <!-- Category: instruction -->
  <xsl:perform-sort
    select? = expression>
    <!-- Content: (xsl:sort+, sequence-constructor) -->
  </xsl:perform-sort>
  ```

`preserve-space` - XPath data model (page 76)

- XSLT 2.0 - 4.4 Stripping Whitespace from a Source Tree
- ```
  <!-- Category: declaration -->
  <xsl:preserve-space
    elements = tokens />
  ```

`processing-instruction` - Constructing result-tree nodes (page 103)

- XSLT 2.0 - 11.6 Creating Processing Instructions
- ```
  <!-- Category: instruction -->
  <xsl:processing-instruction
    name = { ncname }
    select? = expression>
    <!-- Content: sequence-constructor -->
  </xsl:processing-instruction>
  ```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

`result-document` - The transformation environment (page 83)

- XSLT 2.0 - 19.1 Creating Final Result Trees
- ```
  <!-- Category: instruction -->
  <xsl:result-document
    format? = { qname }
    href? = { uri-reference }
    validation? = "strict" | "lax" | "preserve" | "strip"
    type? = qname
    method? = { "xml" | "html" | "xhtml" | "text" | qname-but-not-ncname
   }
    byte-order-mark? = { "yes" | "no" }
    cdata-section-elements? = { qnames }
    doctype-public? = { string }
    doctype-system? = { string }
    encoding? = { string }
    escape-uri-attributes? = { "yes" | "no" }
    include-content-type? = { "yes" | "no" }
    indent? = { "yes" | "no" }
    media-type? = { string }
    normalization-form? = { "NFC" | "NFD" | "NFKC" | "NFKD"
  | "fully-normalized" | "none" | nmtoken }
    omit-xml-declaration? = { "yes" | "no" }
    standalone? = { "yes" | "no" | "omit" }
    undeclare-prefixes? = { "yes" | "no" }
    use-character-maps? = qnames
    output-version? = { nmtoken }>
    <!-- Content: sequence-constructor -->
  </xsl:result-document>
  ```

`sequence` - Why modularize logical and physical structures? (page 88)

- XSLT 2.0 - 11.10 Constructing Sequences
- ```
  <!-- Category: instruction -->
  <xsl:sequence
    select = expression>
    <!-- Content: xsl:fallback* -->
  </xsl:sequence>
  ```

`sort` - Sorting and grouping (page 107)

- XSLT 2.0 - 13.1 The xsl:sort Element
- ```
  <xsl:sort
    select? = expression
    lang? = { nmtoken }
    order? = { "ascending" | "descending" }
    collation? = { uri }
    stable? = { "yes" | "no" }
    case-order? = { "upper-first" | "lower-first" }
    data-type? = { "text" | "number" | qname-but-not-ncname }>
    <!-- Content: sequence-constructor -->
  </xsl:sort>
  ```

`strip-space` - XPath data model (page 76)

- XSLT 2.0 - 4.4 Stripping Whitespace from a Source Tree
- ```
  <!-- Category: declaration -->
  <xsl:strip-space
    elements = tokens />
  ```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

`stylesheet` - The transformation environment (page 83)

- XSLT 2.0 - 3.6 Stylesheet Element
- ```
  <xsl:stylesheet
    id? = id
    extension-element-prefixes? = tokens
    exclude-result-prefixes? = tokens
    version = number
    xpath-default-namespace? = uri
    default-validation? = "preserve" | "strip"
    default-collation? = uri-list
    input-type-annotations? = "preserve" | "strip" | "unspecified">
    <!-- Content: (xsl:import*, other-declarations) -->
  </xsl:stylesheet>
  ```

`template` - A predictable behavior for processors (page 80)

- XSLT 2.0 - 6.1 Defining Templates
- ```
  <!-- Category: declaration -->
  <xsl:template
    match? = pattern
    name? = qname
    priority? = number
    mode? = tokens
    as? = sequence-type>
    <!-- Content: (xsl:param*, sequence-constructor) -->
  </xsl:template>
  ```

`text` - Constructing result-tree nodes (page 103)

- XSLT 2.0 - 11.4.2 Creating Text Nodes Using xsl:text
- ```
  <!-- Category: instruction -->
  <xsl:text
    [disable-output-escaping]?
  = "yes" | "no">
    <!-- Content: #PCDATA -->
  </xsl:text>
  ```

`transform` - The transformation environment (page 83)

- XSLT 2.0 - 3.6 Stylesheet Element
- ```
  <xsl:transform
    id? = id
    extension-element-prefixes? = tokens
    exclude-result-prefixes? = tokens
    version = number
    xpath-default-namespace? = uri
    default-validation? = "preserve" | "strip"
    default-collation? = uri-list
    input-type-annotations? = "preserve" | "strip" | "unspecified">
    <!-- Content: (xsl:import*, other-declarations) -->
  </xsl:transform>
  ```

`value-of` - A predictable behavior for processors (page 80)

- XSLT 2.0 - 11.4.3 Generating Text with xsl:value-of
- ```
  <!-- Category: instruction -->
  <xsl:value-of
    select? = expression
    separator? = { string }
    [disable-output-escaping]?
  = "yes" | "no">
    <!-- Content: sequence-constructor -->
  </xsl:value-of>
  ```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.
  ```

`variable` - Why modularize logical and physical structures? (page 88)

- XSLT 2.0 - 9.1 Variables
- ```
  <!-- Category: declaration -->
  <!-- Category: instruction -->
  <xsl:variable
    name = qname
    select? = expression
    as? = sequence-type>
    <!-- Content: sequence-constructor -->
  </xsl:variable>
  ```

`when` - A predictable behavior for processors (page 80)

- XSLT 2.0 - 8.2 Conditional Processing with xsl:choose
- ```
  <xsl:when
    test = expression>
    <!-- Content: sequence-constructor -->
  </xsl:when>
  ```

`with-param` - Why modularize logical and physical structures? (page 88)

- XSLT 2.0 - 10.1.1 Passing Parameters to Templates
- ```
  <xsl:with-param
    name = qname
    select? = expression
    as? = sequence-type
    tunnel? = "yes" | "no">
    <!-- Content: sequence-constructor -->
  </xsl:with-param>
  ```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XPath 2.0 and XSLT 2.0 function summary

Annex C - Instruction, function and grammar summaries
Section 2 - Vocabulary, functions and grammars XSLT 2.0 and XPath 2.0

All functions of both XPath 2.0 and XSLT 2.0 in alphabetical order follow. This information is mechanically derived from the XPath 2.0 Functions and XSLT 2.0 Recommendations.

`abs` - Data type expressions and functions (page 94)

- XPath 2.0 - 6.4 Functions on Numeric Values
- *`numeric`* `abs(` *`numeric`* `)`

`adjust-date-to-timezone` - Data type expressions and functions (page 97)

- XPath 2.0 - 10.7 Timezone Adjustment Functions on Dates and Time Values
- *`xs:date`* `adjust-date-to-timezone(` *`xs:date`* `)`

  `xs:date` `adjust-date-to-timezone(` *`xs:date`*`,` *`xs:dayTimeDuration`* `)`

`adjust-dateTime-to-timezone` - Data type expressions and functions (page 97)

- XPath 2.0 - 10.7 Timezone Adjustment Functions on Dates and Time Values
- *`xs:dateTime`* `adjust-dateTime-to-timezone(` *`xs:dateTime`* `)`

  `xs:dateTime` `adjust-dateTime-to-timezone(` *`xs:dateTime`*`,` *`xs:dayTimeDuration`* `)`

`adjust-time-to-timezone` - Data type expressions and functions (page 97)

- XPath 2.0 - 10.7 Timezone Adjustment Functions on Dates and Time Values
- *`xs:time`* `adjust-time-to-timezone(` *`xs:time`* `)`

  `xs:time` `adjust-time-to-timezone(` *`xs:time`*`,` *`xs:dayTimeDuration`* `)`

`avg` - Data type expressions and functions (page 96)

- XPath 2.0 - 15.4 Aggregate Functions
- *`xs:anyAtomicType`* `avg(` *`xs:anyAtomicType*`* `)`

`base-uri` - Data type expressions and functions (page 99)

- XPath 2.0 - 2 Accessors
- *`xs:anyURI`* `base-uri( )`

  `xs:anyURI` `base-uri(` *`node()`* `)`

`boolean` - Data type expressions and functions (page 94)

- XPath 2.0 - 15.1 General Functions and Operators on Sequences
- *`xs:boolean`* `boolean(` *`item()*`* `)`

`ceiling` - Data type expressions and functions (page 94)

- XPath 2.0 - 6.4 Functions on Numeric Values
- *`numeric`* `ceiling(` *`numeric`* `)`

`codepoint-equal` - Data type expressions and functions (page 94)

- XPath 2.0 - 7.3 Equality and Comparison of Strings
- *`xs:boolean`* `codepoint-equal(` *`xs:string`*`,` *`xs:string`* `)`

`codepoints-to-string` - Data type expressions and functions (page 94)

- XPath 2.0 - 7.2 Functions to Assemble and Disassemble Strings
- *`xs:string`* `codepoints-to-string(` *`xs:integer*`* `)`

`collection` - Why modularize logical and physical structures? (page 89)

- XPath 2.0 - 15.5 Functions and Operators that Generate Sequences
- *`node()*`* `collection( )`

  `node()*` `collection(` *`xs:string`* `)`

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
  ```

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

`system-property` - The transformation environment (page 83)
- XSLT 2.0 - 16.6.5 system-property
- *`xs:string`* `system-property(` *`property-name`* `)`

`timezone-from-date` - Data type expressions and functions (page 98)
- XPath 2.0 - 10.5 Component Extraction Functions on Durations, Dates and Times
- *`xs:dayTimeDuration`* `timezone-from-date(` *`xs:date`* `)`

`timezone-from-dateTime` - Data type expressions and functions (page 98)
- XPath 2.0 - 10.5 Component Extraction Functions on Durations, Dates and Times
- *`xs:dayTimeDuration`* `timezone-from-dateTime(` *`xs:dateTime`* `)`

`timezone-from-time` - Data type expressions and functions (page 98)
- XPath 2.0 - 10.5 Component Extraction Functions on Durations, Dates and Times
- *`xs:dayTimeDuration`* `timezone-from-time(` *`xs:time`* `)`

`tokenize` - Data type expressions and functions (page 95)
- XPath 2.0 - 7.6 String Functions that Use Pattern Matching
- *`xs:string*`* `tokenize(` *`xs:string`*`, `*`xs:string`* `)`

  `xs:string*` `tokenize(` *`xs:string`*`, `*`xs:string`*`, `*`xs:string`* `)`

`trace` - The transformation environment (page 82)
- XPath 2.0 - 4 The Trace Function
- *`item()*`* `trace(` *`item()*`*`, `*`xs:string`* `)`

`translate` - Data type expressions and functions (page 95)
- XPath 2.0 - 7.4 Functions on String Values
- *`xs:string`* `translate(` *`xs:string`*`, `*`xs:string`*`, `*`xs:string`* `)`

`true` - Data type expressions and functions (page 94)
- XPath 2.0 - 9.1 Additional Boolean Constructor Functions
- *`xs:boolean`* `true(  )`

`type-available` - The transformation environment (page 83)
- XSLT 2.0 - 18.1.4 Testing Availability of Types
- *`xs:boolean`* `type-available(` *`type-name`* `)`

`unordered` - Data type expressions and functions (page 96)
- XPath 2.0 - 15.1 General Functions and Operators on Sequences
- *`item()*`* `unordered(` *`item()*`* `)`

`unparsed-entity-public-id` - Why modularize logical and physical structures? (page 89)
- XSLT 2.0 - 16.6.3 unparsed-entity-public-id
- *`xs:string`* `unparsed-entity-public-id(` *`entity-name`* `)`

`unparsed-entity-uri` - Why modularize logical and physical structures? (page 89)
- XSLT 2.0 - 16.6.2 unparsed-entity-uri
- *`xs:anyURI`* `unparsed-entity-uri(` *`entity-name`* `)`

`unparsed-text` - Why modularize logical and physical structures? (page 89)
- XSLT 2.0 - 16.2 Reading Text Files
- *`xs:string?`* `unparsed-text(` *`href`* `)`

  `xs:string?` `unparsed-text(` *`href`*`, `*`encoding`* `)`

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

`unparsed-text-available` - Why modularize logical and physical structures? (page 89)
- XSLT 2.0 - 16.2 Reading Text Files
- *`xs:boolean`* `unparsed-text-available(` *`href`* `)`

  `xs:boolean` `unparsed-text-available(` *`href`*`, `*`encoding`* `)`

`upper-case` - Data type expressions and functions (page 95)
- XPath 2.0 - 7.4 Functions on String Values
- *`xs:string`* `upper-case(` *`xs:string`* `)`

`year-from-date` - Data type expressions and functions (page 98)
- XPath 2.0 - 10.5 Component Extraction Functions on Durations, Dates and Times
- *`xs:integer`* `year-from-date(` *`xs:date`* `)`

`year-from-dateTime` - Data type expressions and functions (page 98)
- XPath 2.0 - 10.5 Component Extraction Functions on Durations, Dates and Times
- *`xs:integer`* `year-from-dateTime(` *`xs:dateTime`* `)`

`years-from-duration` - Data type expressions and functions (page 98)
- XPath 2.0 - 10.5 Component Extraction Functions on Durations, Dates and Times
- *`xs:integer`* `years-from-duration(` *`xs:duration`* `)`

`zero-or-one` - Data type expressions and functions (page 96)
- XPath 2.0 - 15.2 Functions That Test the Cardinality of Sequences
- *`item()`* `zero-or-one(` *`item()*`* `)`

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

XPath 2.0 grammar productions

Annex C - Instruction, function and grammar summaries
Section 2 - Vocabulary, functions and grammars XSLT 2.0 and XPath 2.0

Expressions (3)
[1] XPath ::= Expr[2]
[2] Expr ::= ExprSingle[3] ("," ExprSingle[3])*
[3] ExprSingle ::= ForExpr[4]| QuantifiedExpr[6]| IfExpr[7]| OrExpr[8]

For Expressions (3.7)
[4] ForExpr ::= SimpleForClause[5] "return" ExprSingle[3]
[5] SimpleForClause ::= "for" "$" VarName[45] "in" ExprSingle[3] ("," "$" VarName[45] "in" ExprSingle[3])*

Quantified Expressions (3.9)
[6] QuantifiedExpr ::= ("some" | "every") "$" VarName[45] "in" ExprSingle[3] ("," "$" VarName[45] "in" ExprSingle[3])* "satisfies" ExprSingle[3]

Conditional Expressions (3.8)
[7] IfExpr ::= "if" "(" Expr[2] ")" "then" ExprSingle[3] "else" ExprSingle[3]

Logical Expressions (3.6)
[8] OrExpr ::= AndExpr[9] ("or" AndExpr[9])*
[9] AndExpr ::= ComparisonExpr[10] ("and" ComparisonExpr[10])*

Comparison Expressions (3.5)
[10] ComparisonExpr ::= RangeExpr[11] ((ValueComp[23]| GeneralComp[22]| NodeComp[24]) RangeExpr[11])?

Constructing Sequences (3.3.1)
[11] RangeExpr ::= AdditiveExpr[12] ("to" AdditiveExpr[12])?

Arithmetic Expressions (3.4)
[12] AdditiveExpr ::= MultiplicativeExpr[13] (("+" | "-") MultiplicativeExpr[13])*
[13] MultiplicativeExpr ::= UnionExpr[14] (("*" | "div" | "idiv" | "mod") UnionExpr[14])*

Combining Node Sequences (3.3.3)
[14] UnionExpr ::= IntersectExceptExpr[15] (("union" | "|") IntersectExceptExpr[15])*
[15] IntersectExceptExpr ::= InstanceofExpr[16] (("intersect" | "except") InstanceofExpr[16])*

Instance Of (3.10.1)
[16] InstanceofExpr ::= TreatExpr[17] ("instance" "of" SequenceType[50])?

Treat (3.10.5)
[17] TreatExpr ::= CastableExpr[18] ("treat" "as" SequenceType[50])?

Castable (3.10.3)
[18] CastableExpr ::= CastExpr[19] ("castable" "as" SingleType[49])?

Cast (3.10.2)
[19] CastExpr ::= UnaryExpr[20] ("cast" "as" SingleType[49])?

Arithmetic Expressions (3.4)
[20] UnaryExpr ::= ("-" | "+")* ValueExpr[21]
[21] ValueExpr ::= PathExpr[25]

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Comparison Expressions (3.5)
[22] GeneralComp ::= "=" | "!=" | "<" | "<=" | ">" | ">="
[23] ValueComp ::= "eq" | "ne" | "lt" | "le" | "gt" | "ge"
[24] NodeComp ::= "is" | "<<" | ">>"

Path Expressions (3.2)
[25] PathExpr ::= ("/" RelativePathExpr[26]?)| ("//" RelativePathExpr[26])| RelativePathExpr[26]
[26] RelativePathExpr ::= StepExpr[27] (("/" | "//") StepExpr[27])*

Steps (3.2.1)
[27] StepExpr ::= FilterExpr[38] | AxisStep[28]
[28] AxisStep ::= (ReverseStep[32] | ForwardStep[29]) PredicateList[39]
[29] ForwardStep ::= (ForwardAxis[30] NodeTest[35]) | AbbrevForwardStep[31]

Axes (3.2.1.1)
[30] ForwardAxis ::= ("child" "::")| ("descendant" "::")| ("attribute" "::")| ("self" "::")| ("descendant-or-self" "::")| ("following-sibling" "::")| ("following" "::")| ("namespace" "::")

Abbreviated Syntax (3.2.4)
[31] AbbrevForwardStep ::= "@"? NodeTest[35]

Steps (3.2.1)
[32] ReverseStep ::= (ReverseAxis[33] NodeTest[35]) | AbbrevReverseStep[34]

Axes (3.2.1.1)
[33] ReverseAxis ::= ("parent" "::")| ("ancestor" "::")| ("preceding-sibling" "::")| ("preceding" "::")| ("ancestor-or-self" "::")

Abbreviated Syntax (3.2.4)
[34] AbbrevReverseStep ::= ".."

Node Tests (3.2.1.2)
[35] NodeTest ::= KindTest[54] | NameTest[36]
[36] NameTest ::= QName[78] | Wildcard[37]
[37] Wildcard ::= "*"| (NCName[79] ":" "*")| ("*" ":" NCName[79])

Filter Expressions (3.3.2)
[38] FilterExpr ::= PrimaryExpr[41] PredicateList[39]

Steps (3.2.1)
[39] PredicateList ::= Predicate[40]*

Predicates (3.2.2)
[40] Predicate ::= "[" Expr[2] "]"

Primary Expressions (3.1)
[41] PrimaryExpr ::= Literal[42] | VarRef[44] | ParenthesizedExpr[46] | ContextItemExpr[47] | FunctionCall[48]

Literals (3.1.1)
[42] Literal ::= NumericLiteral[43] | StringLiteral[74]
[43] NumericLiteral ::= IntegerLiteral[71] | DecimalLiteral[72] | DoubleLiteral[73]

Variable References (3.1.2)
[44] VarRef ::= "$" VarName[45]
[45] VarName ::= QName[78]

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Parenthesized Expressions (3.1.3)
[46] ParenthesizedExpr ::= "(" Expr[2]? ")"

Context Item Expression (3.1.4)
[47] ContextItemExpr ::= "."

Function Calls (3.1.5)
[48] FunctionCall ::= QName[78] "(" (ExprSingle[3] ("," ExprSingle[3])*)? ")"

Cast (3.10.2)
[49] SingleType ::= AtomicType[53] "?"?

SequenceType Syntax (2.5.3)
[50] SequenceType ::= ("empty-sequence" "(" ")")| (ItemType[52]
OccurrenceIndicator[51]?)
[51] OccurrenceIndicator ::= "?" | "*" | "+"
[52] ItemType ::= KindTest[54] | ("item" "(" ")") | AtomicType[53]
[53] AtomicType ::= QName[78]
[54] KindTest ::= DocumentTest[56]| ElementTest[64]| AttributeTest[60]|
SchemaElementTest[66]| SchemaAttributeTest[62]| PITest[59]|
CommentTest[58]| TextTest[57]| AnyKindTest[55]
[55] AnyKindTest ::= "node" "(" ")"
[56] DocumentTest ::= "document-node" "(" (ElementTest[64] |
SchemaElementTest[66])? ")"
[57] TextTest ::= "text" "(" ")"
[58] CommentTest ::= "comment" "(" ")"
[59] PITest ::= "processing-instruction" "(" (NCName[79] | StringLiteral[74])?
")"
[60] AttributeTest ::= "attribute" "(" (AttribNameOrWildcard[61] ","
TypeName[70])?)? ")"
[61] AttribNameOrWildcard ::= AttributeName[68] | "*"
[62] SchemaAttributeTest ::= "schema-attribute" "(" AttributeDeclaration[63]
")"
[63] AttributeDeclaration ::= AttributeName[68]
[64] ElementTest ::= "element" "(" (ElementNameOrWildcard[65] "," TypeName[70]
"?"?)?)? ")"
[65] ElementNameOrWildcard ::= ElementName[69] | "*"
[66] SchemaElementTest ::= "schema-element" "(" ElementDeclaration[67] ")"
[67] ElementDeclaration ::= ElementName[69]
[68] AttributeName ::= QName[78]
[69] ElementName ::= QName[78]
[70] TypeName ::= QName[78]

Literals (3.1.1)
[71] IntegerLiteral ::= Digits[81]
[72] DecimalLiteral ::= ("." Digits[81]) | (Digits[81] "." [0-9]*)
[73] DoubleLiteral ::= (("." Digits[81]) | (Digits[81] ("." [0-9]*)?)) [eE]
[+-]? Digits[81]
[74] StringLiteral ::= ('"' (EscapeQuot[75] | [^"])* '"') | ("'"
(EscapeApos[76] | [^'])* "'")
[75] EscapeQuot ::= '""'
[76] EscapeApos ::= "''"

Comments (2.6)
[77] Comment ::= "(:" (CommentContents[82] | Comment[77])* ":)"

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Terminal Symbols (1.2.1)
[78] QName ::= [http://www.w3.org/TR/REC-xml-names/#NT-QName][XML-
Names-7]
[79] NCName ::= [http://www.w3.org/TR/REC-xml-names/#NT-NCName][XML-
Names-4]
[80] Char ::= [http://www.w3.org/TR/REC-xml#NT-Char][XML-
2]

Literals (3.1.1)
[81] Digits ::= [0-9]+

Comments (2.6)
[82] CommentContents ::= (Char[80]+ - (Char* ('(:' | ':)') Char*))

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLT 2.0 grammar productions

Annex C - Instruction, function and grammar summaries
Section 2 - Vocabulary, functions and grammars XSLT 2.0 and XPath 2.0

```
[1] Pattern ::= PathPattern[2]
              |  Pattern[1] '|'  PathPattern[2]
[2] PathPattern ::= RelativePathPattern[3]
                  | '/'  RelativePathPattern[3]?
                  | '//'  RelativePathPattern[3]
                  |  IdKeyPattern[6] (('/' | '//')  RelativePathPattern[3])?
[3] RelativePathPattern ::= PatternStep[4] (('/' | '//')
                           RelativePathPattern[3])?
[4] PatternStep ::= PatternAxis[5]? NodeTest[XPath-35]   PredicateList[XPath-
                  39]
[5] PatternAxis ::= ('child' '::' | 'attribute' '::' | '@')
[6] IdKeyPattern ::= 'id' '('  IdValue[7] ')'
                   | 'key' '('  StringLiteral[XPath-74] ',' KeyValue[8] ')'
[7] IdValue ::= StringLiteral[XPath-74] | VarRef[XPath-44]
[8] KeyValue ::= Literal[XPath-42]  | VarRef[XPath-44]
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Annex D - Tool questions

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Sample questions for vendors

Annex D - Tool questions

Answers to the following questions may prove useful when trying to better understand a 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 and XQuery that may impact on the stylesheets and transformation specifications you write.

- how is the product identified?
  - what is the name of the processor in product literature?
  - what value is returned by the system properties?
    - recall The transformation environment (page 83)
  - what version of specification is supported?
    - returned by the `xsl:version` system property in XSLT
  - 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?
  - XHTML?
  - 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?

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Sample questions for vendors (cont.)

Annex D - Tool questions

- how does the processor differ from the W3C working drafts or recommendations?
  - upon which dated W3C documents describing the specifications 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?
  - 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?

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Sample questions for vendors (cont.)

Annex D - Tool questions

- how are particular 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>`?
  - 1.5 what is the URI syntax for data projection for input?
  - 1.5 which collations are supported for string comparison?
  - 1.5 is the processor schema-aware?
  - 1.5 what are the details of the collection URI syntax?
- how are 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?

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Practical Transformation Using XSLT and XPath

Sample questions for vendors (cont.)

Annex D - Tool questions

- what are the details of the implementation and invocation of the processor?
  - how are user values passed in to the transformation?
  - 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?

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Information subject to restrictive legend on first page.

XSLStyle™

Annex D - Tool questions
Section 1 - XSLStyle™

An XSLT stylesheet embedded documentation methodology

- an XSLT stylesheet is an XML document
- one can embed a documentation vocabulary in an XSLT stylesheet through standard XML namespace techniques
- a stylesheet for stylesheets renders the embedded documentation to HTML and CSS
- freely downloadable environment from the Crane Softwrights Ltd. web site as a developer resource

Can invoke as a separate stylesheet or through embedded association

- see `http://www.w3.org/TR/xml-stylesheet/`
- recall Stylesheet association (page 34)

Embedded documentation uses a Crane namespace for structure

- DocBook for content
  - see `http://www.docbook.org/` for details
- DITA for content
  - see `http://dita.xml.org/` for details
- XHTML for content

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLStyle™ (cont.)

Annex D - Tool questions
Section 1 - XSLStyle™

Enforces "stylesheet writing rules" on the writer of the stylesheet

- adds rigor to the stylesheet
- e.g. all top-level constructs must be separately documented
- e.g. all parameters of all templates and functions must be separately documented
- e.g. all parameters of all templates and functions must have declared types
- e.g. all named top-level constructs must be namespace qualified
- many other rules

Resulting HTML report is similar to an enhanced Javadoc report

- encompasses the complete import tree
- alphabetized index of all named top-level constructs
- deficiencies report
  - fully hyperlinked content
  - one could institute a development rule of not allowing the check-in of a stylesheet until the library is fully documented according to the writing rules

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLStyle™ (cont.)

Annex D - Tool questions
Section 1 - XSLStyle™

An example fragment:

```
<?xml-stylesheet type="text/xsl" href="xslstyle-docbook.xsl"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xs="http://www.CraneSoftwrights.com/ns/xslstyle"
                xmlns:i="internal-namespace"
                exclude-result-prefixes="xs i"
                version="1.0">

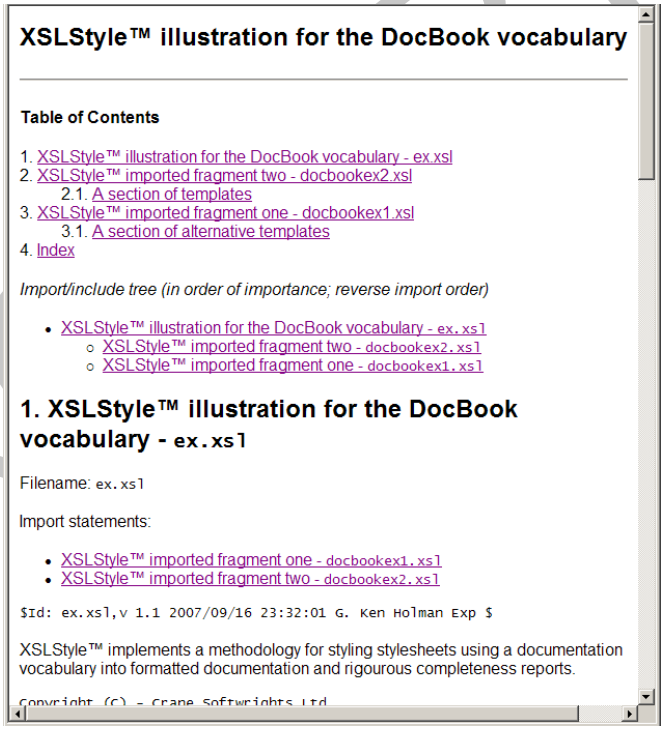
<!--an example import-->
<xsl:import href="docbookex1.xsl"/>
<!--another example import-->
<xsl:import href="docbookex2.xsl"/>

<xs:doc info="$Id: ex.xsl,v 1.1 2007/09/16 23:32:01 G. Ken Holman Exp
$"
  filename="ex.xsl" global-ns="xs" internal-ns="i" vocabulary="DocBook">
  <xs:title>XSLStyle&#x2122; illustration for the DocBook
vocabulary</xs:title>
  <para>
    XSLStyle&#x2122; implements a methodology for styling stylesheets
    using a documentation vocabulary into formatted documentation and
    rigourous completeness reports.
  </para>
  <programlisting>
Copyright (C) - Crane Softwrights Ltd.
...
<xs:template>
  <para>The formatting of a single entry in the import tree.</para>
  <xs:param name="href">
    <para>The URI used to access the module for this entry.</para>
  </xs:param>
</xs:template>
<xsl:template name="i:format-tree-entry">
  <xsl:param name="href"/>
  <listitem>
...
```


Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

XSLStyle™ (cont.)

Annex D - Tool questions
Section 1 - XSLStyle™



XSLStyle™ illustration for the DocBook vocabulary

Table of Contents

1. XSLStyle™ illustration for the DocBook vocabulary - ex.xsl
2. XSLStyle™ imported fragment two - docbookex2.xsl
   2.1. A section of templates
3. XSLStyle™ imported fragment one - docbookex1.xsl
   3.1. A section of alternative templates
4. Index

Import/include tree (in order of importance; reverse import order)

- XSLStyle™ illustration for the DocBook vocabulary - ex.xsl
  - XSLStyle™ imported fragment two - docbookex2.xsl
  - XSLStyle™ imported fragment one - docbookex1.xsl

1. XSLStyle™ illustration for the DocBook vocabulary - ex.xsl

Filename: ex.xsl

Import statements:

- XSLStyle™ imported fragment one - docbookex1.xsl
- XSLStyle™ imported fragment two - docbookex2.xsl

```
$Id: ex.xsl,v 1.1 2007/09/16 23:32:01 G. Ken Holman Exp $
```

XSLStyle™ implements a methodology for styling stylesheets using a documentation vocabulary into formatted documentation and rigourous completeness reports.

Copyright (C) - Crane Softwrights Ltd.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Where to go from here?

Conclusion - Practical Transformation Using XSLT and XPath

The work on XSL, XQuery, XSLT and XPath continues:

- all are full W3C Recommendations undergoing designs for new features
- 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 W3C with comments about the XSLT/XPath/XQuery specifications:
    - `http://www.w3.org/XML/2005/04/qt-bugzilla`
    - `mailto:public-qt-comments@w3.org`

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

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)" Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
  - 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 chapters and two useful annexes
- site-wide and world-wide staff licenses (one-time fee) are available

See `http://www.CraneSoftwrights.com/links/trn-20110211.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 `info@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

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Practical Transformation Using XSLT and XPath

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

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

Copyright © Crane Softwrights Ltd.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Information subject to restrictive legend on first page.

Practical Transformation Using XSLT and XPath

Table of contents
Indexed by slide number

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Practical Transformation Using XSLT and XPath

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Information subject to restrictive legend on first page.

Copyright © Crane Softwrights Ltd.

Practical Transformation Using XSLT and XPath

Index

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5

Practical Transformation Using XSLT and XPath

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5
Information subject to restrictive legend on first page.

Fourteenth Edition - 2011-02-11 - ISBN 978-1-894049-24-5