The Manakin dependencies: XML/XSLT and Cocoon

1. XML

This introduction to XML is based on the w3schools "XML tutorial" (http://www.w3schools.com/xml/)

What is XML?

XML stands for EXtensible Markup Language and is a markup language much like HTML. XML was designed to carry data, not to display data and tags are not predefined, you must define your own tags. XML is a W3C recommendation.

An Example XML Document

XML documents use a self-describing and simple syntax:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = Latin-1/West European character set). The next line describes the root element of the document (like saying: "this document is a note"). The next 4 lines describe 4 child elements of the root (to, from, heading, and body). All elements can have text content and attributes (just like in HTML, `<book category="children">`).

Rules

- All XML Elements Must Have a Closing Tag
- XML Tags are Case Sensitive
- XML Elements Must be Properly Nested
- XML Documents Must Have a Root Element
- XML Attribute Values Must be Quoted

Displaying XML with XSLT

It is possible to use CSS to format and layout an XML document. However XSLT (eXtensible Stylesheet Language Transformations) is the recommended style sheet language of XML and is far more sophisticated than CSS.

The XSLT transformation can be done by the browser, when the browser reads the XML file. Different browsers may produce different results when transforming XML with XSLT. To reduce this problem the XSLT transformation can also be done on the server, as is the case for the DSpace XMLUI.

XML Namespaces

XML Namespaces provide a method to avoid element name conflicts. In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications. Name conflicts in XML can easily be avoided using a name prefix (`<h:table>`)

The xmlns Attribute

When using prefixes in XML, a so-called namespace for the prefix must be defined. The namespace is defined by the xmlns attribute in the start tag of an element. The namespace declaration has the following syntax, `xmlns:prefix="URI"`.

When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace. Namespaces can also be declared in the elements where they are used (`<h:table xmlns:h="http://www.w3.org/TR/html4/">`). Or namespaces can be declared in the XML root element (`<root xmlns:h="http://www.w3.org/TR/html4/">`).

The namespace URI is not used by the parser to look up information. The purpose is to give the namespace a unique name. However, often companies use the namespace as a pointer to a web page containing namespace information.

XML encoding

```
<?xml version="1.0" encoding="windows-1252"?>
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-16"?>
```

Best Practices:

- Always use the encoding attribute
- Use an editor that supports encoding
- Make sure you know what encoding the editor uses
- Use the same encoding in your encoding attribute

XML Schema

Purpose

The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD. An XML Schema defines the elements and attributes that can appear in a document. The schema also defines the possible child elements and their number and order. It also determines whether an element can be empty and if an the element is not empty the schema defines data types, default and fixed values for both elements and attributes.

XML Schema Example

XML example

```
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML Schema Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

The note element is a complex type because it contains other elements. The other elements (to, from, heading, body) are simple types because they do not contain other elements.

Consider the following XML node:

```
<dateborn>1970-03-27</dateborn>
```

The following line will define in XML Schema that the node `<dateborn>` can only contain valid date values.

```
<xs:element name="dateborn" type="xs:date"/>
```

Referencing an XML schema

```
<?xml version="1.0"?>
<note  xmlns="http://www.w3schools.com"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

2. XPath

This introduction to XPath is based on the w3schools “XPath tutorial“ (http://www.w3schools.com/xpath/)

What is XPath?

XPath is a syntax for defining parts of an XML document and uses path expressions to navigate in XML documents. XPath is a major element in XSLT.

XPath in XSLT

XPath is an important part of the XSLT standard. XPath knowledge is necessary to be able to manipulate XML with XSLT documents. XSLT will perform operations on an XML file to manipulate and transform it into some other form (eg. XHTML) and XPath is essential to XSLT in the sense that XPath is the technique used to select portions of the input XML file to be transformed by XSLT templates. More about XPath in XSLT in section “3. XSLT”.

Terminology

Nodes - In XPath, there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document nodes. XML documents are treated as trees of nodes. The topmost element of the tree is called the root element.

Atomic values - Atomic values are nodes with no children or parent.

Items - Items are atomic values or nodes.

Parent - Each element and attribute has one parent.

Children - Element nodes may have zero, one or more children.

Siblings - Nodes that have the same parent.

Ancestors - A node's parent, parent's parent, etc.

Descendants - A node's children, children's children, etc.

Path Expressions & Standard Functions

XPath uses *path expressions* to select nodes or node-sets in an XML document. These path expressions look very much like the expressions you see when you work with a traditional computer file system.

Selecting Nodes

XPath uses path expressions to select nodes in an XML document. The node is selected by following a path or steps. The most useful path expressions are listed below:

Expression	Description
`nodename`	Selects all child nodes of the named node
`/`	Selects from the root node
`//`	Selects nodes in the document from the current node that match the selection no matter where they are
`.`	Selects the current node
`..`	Selects the parent of the current node
`@`	Selects attributes

Example XML

```
<?xml version="1.0"?>
<bookstore>
<book>
  <title lang="eng">Harry Potter</title>
  <price>29.99</price>
</book>
<book>
  <title lang="eng">Learning XML</title>
  <price>39.95</price>
</book>
</bookstore>
```

In the table below we have listed some path expressions and the result of the expressions based on the example above:

Path Expression	Result
`bookstore`	Selects all the child nodes of the bookstore element
`/bookstore`	Selects the root element bookstore **Note:** If the path starts with a slash (/) it always represents an absolute path to an element!
`bookstore/book`	Selects all book elements that are children of bookstore
`//book`	Selects all book elements no matter where they are in the document
`bookstore//book`	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element
`//@lang`	Selects all attributes that are named lang

Predicates

Predicates are used to find a specific node or a node that contains a specific value. Predicates are always embedded in square brackets. In the table below there are some path expressions with predicates and the result of the expressions:

Path Expression	Result
`/bookstore/book[1]`	Selects the first book element that is the child of the bookstore element.
`/bookstore/book[last()]`	Selects the last book element that is the child of the bookstore element
`/bookstore/book[last()-1]`	Selects the last but one book element that is the child of the bookstore element
`/bookstore/book[position()<3]`	Selects the first two book elements that are children of the bookstore element
`//title[@lang]`	Selects all the title elements that have an attribute named lang
`//title[@lang='eng']`	Selects all the title elements that have an attribute named lang with a value of 'eng'
`/bookstore/book[price>35]`	Selects all the book elements of the bookstore element that have a price element with a value greater than 35
`/bookstore/book[price>35]/title`	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35

Operators

Below is a list of the operators that can be used in XPath expressions:

Operator	Description	Example	Return value
`|`	Computes two node-sets	`//book | //cd`	Returns a node-set with all book and cd elements
`+`	Addition	`6 + 4`	10
`=`	Equal	`price=9.80`	true if price is 9.80 false if price is 9.90
`!=`	Not equal	`price!=9.80`	true if price is 9.90 false if price is 9.80
`<`	Less than	`price<9.80`	true if price is 9.00 false if price is 9.80
`<=`	Less than or equal to	`price<=9.80`	true if price is 9.00 false if price is 9.90
`or`	or	`price=9.80 or price=9.70`	true if price is 9.80 false if price is 9.50
`and`	and	`price>9.00 and price<9.90`	true if price is 9.80 false if price is 8.50

For a exhaustive list of XPath operators please refer to “XQuery 1.0 and XPath 2.0 Functions and Operators” [1]

[1] http://www.w3.org/TR/xpath-functions/

Selecting Unknown Nodes

XPath wildcards can be used to select unknown XML elements.

Wildcard	Description
`*`	Matches any element node
`@*`	Matches any attribute node
`node()`	Matches any node of any kind

In the table below we have listed some path expressions and the result of the expressions:

Path Expression	Result
`/bookstore/*`	Selects all the child nodes of the bookstore element
`//*`	Selects all elements in the document
`//title[@*]`	Selects all title elements which have any attribute

Selecting Several Paths

By using the | operator (OR operator) in an XPath expression you can select several paths. In the table below we have listed some path expressions and the result of the expressions:

Path Expression	Result
`//book/title | //book/price`	Selects all the title AND price elements of all book elements
`//title | //price`	Selects all the title AND price elements in the document
`/bookstore/book/title | //price`	Selects all the title elements of the book element of the bookstore element AND all the price elements in the document

Standard Functions

Name	Description
`fn:not(arg)`	Returns true if the boolean value is false, and false if the boolean value is true
`fn:starts-with(string1,string2)`	Returns true if string1 starts with string2, otherwise it returns false
`fn:concat(string,string,...)`	Returns the concatenation of the strings
`fn:substring(string,start,len)`	Returns the substring from the start position to the specified length. Index of the first character is 1. If length is omitted it returns the substring from the start position to the end
`fn:string-length(string)`	Returns the length of the specified string. If there is no string argument it returns the length of the string value of the current node
`fn:number(arg)`	Returns the numeric value of the argument. The argument could be a boolean, string, or node-set

For an exhaustive list of XPath standard functions please refer to “XQuery 1.0 and XPath 2.0 Functions and Operators” [2]

[2] http://www.w3.org/TR/xpath-functions/

Exercises

```
<catalog>
       <cd stock="2">
              <title>Nevermind</title>
              <artist>Nirvana</artist>
              <country>USA</country>
              <company>Geffen</company>
              <genre>Grunge</genre>
              <length>59:23</length>
              <year>1991</year>
       </cd>
       <cd stock="0">
              <title>Kind of blue</title>
              <artist>Miles Davis</artist>
              <country>USA</country>
              <company>Columbia</company>
              <genre>Jazz</genre>
              <length>45:44</length>
              <year>1959</year>
       </cd>
       <cd stock="2">
              <title>In a Silent Way</title>
              <artist>Miles Davis</artist>
              <country>USA</country>
              <company>Columbia</company>
              <genre>Jazz</genre>
              <length>38:10</length>
              <year>1969</year>
       </cd>
       <cd stock="3">
              <title>Led Zeppelin IV</title>
              <artist>Led Zeppelin</artist>
              <country>UK</country>
              <company>Atlantic</company>
              <genre>Rock</genre>
              <length>42:33</length>
              <year>1971</year>
       </cd>
       <cd stock="0">
              <title>Grace</title>
              <artist>Jeff Buckley</artist>
              <country>USA</country>
              <company>Columbia</company>
              <genre>Rock</genre>
              <length>51:44</length>
              <year>1994</year>
       </cd>
       <cd stock="1">
              <title>My Generation</title>
              <artist>The Who</artist>
              <country>UK</country>
              <company>Brunswick</company>
              <genre>Rock</genre>
              <length>36:13</length>
              <year>1965</year>
       </cd>
</catalog>
```

For the XML example above, write XPtah expressions to select the information in the following statements:

2.1. All genre nodes
2.2. All cd titles
2.3. All cd nodes that are out of stock
2.4. All cd nodes that have the genre 'Rock'
2.5. All cd titles that were made in the USA
2.6. The years 'Miles Davis' has released a cd
2.7. All cd titles by the company 'Columbia' of the genre 'Rock'
2.8. All cd titles that are in stock, that don't have the genre 'Jazz'.

3. XSLT

This introduction to XSLT is based on the w3schools “XSLT tutorial“ (http://www.w3schools.com/xsl/)

What is XSLT?

XSLT stands for XSL Transformations and transforms an XML document into another XML document. XSLT uses XPath to navigate in XML documents. XSLT is also a W3C recommendation.

XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element.

With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.

XSLT Uses XPath

XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document.

XSL Transformations

The root element that declares the document to be an XSL style sheet is `<xsl:stylesheet>` or `<xsl:transform>`. The correct way to declare an XSL style sheet according to the W3C XSLT Recommendation is: `<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`

XML Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd>
    <title>Nevermind</title>
    <artist>Nirvana</artist>
    <country>USA</country>
    <company>Geffen</company>
    <genre>Grunge</genre>
    <length>59:23</length>
    <year>1991</year>
  </cd>
  <cd>
    ...
  </cd>
</catalog>
```

The <xsl:template> Element

The `<xsl:template>` element is used to build templates.

The match attribute is used to associate a template with an XML element. The match attribute can also be used to define a template for the entire XML document. The value of the match attribute is an XPath expression (i.e. match="/" defines the whole document).

```
<xsl:template match="/">
<xsl:template match="//company">
<xsl:template match="catalog/cd">
```

The <xsl:value-of> Element

The `<xsl:value-of>` element can be used to extract the value of an XML element and add it to the output stream of the transformation.

```
<xsl:value-of select="catalog/cd/title"/>
```

The <xsl:for-each> Element

The XSL `<xsl:for-each>` element can be used to select every XML element of a specified node-set.

```
<xsl:for-each select="catalog/cd"> ... </xsl:for-each>
```

The <xsl:if> Element

To put a conditional if test against the content of the XML file, add an `<xsl:if>` element to the XSL document.

```
<xsl:if test="expression"> ... </xsl:if>
```

The <xsl:choose> Element

The <xsl:choose> element is used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests.

```
<xsl:choose>
  <xsl:when test="expression"> ... </xsl:when>
  <xsl:otherwise> ... </xsl:otherwise>
</xsl:choose>
```

The <xsl:sort> Element

To sort the output, simply add an `<xsl:sort>` element inside the `<xsl:for-each>` element in the XSL file.

The <xsl:apply-templates> Element

The `<xsl:apply-templates>` element applies a template to the current element or to the current element's child nodes.

If we add a select attribute to the `<xsl:apply-templates>` element it will process only the child element that matches the value of the attribute. We can use the select attribute to specify the order in which the child nodes are processed.

The <xsl:import> Element

Imports the contents of one style sheet into another. Note: An imported style sheet has lower precedence than the importing style sheet. The `<xsl:import>` element is often used to create layers of XSL code. The basic idea is to create a customization layer in which you put all your changes, and then rely on standard stylesheets for everything else by using the `<xsl:import>` element. The following line, the first line in a DSpace Manakin Theme, defines the dri2xhtml.xsl as the standard stylesheet and everything following this line can override any templates in the dri2xhtml.xsl.

```
<xsl:import href="../dri2xhtml.xsl"/>
```

Exercises

3.1. Develop an XSL Stylesheet that will transform the example XML into a HTML page containing a bullet point list formatted as follows:

- Nirvana, Nevermind - in stock
- Artist C, Album D - out of stock
- ...

Example XML

```
<catalog>
       <cd stock="2">
              <title>Nevermind</title>
              <artist>Nirvana</artist>
              <country>USA</country>
              <company>Geffen</company>
              <genre>Grunge</genre>
              <length>59:23</length>
              <year>1991</year>
       </cd>
       <cd stock="0">
              <title>Kind of blue</title>
              <artist>Miles Davis</artist>
              <country>USA</country>
              <company>Columbia</company>
              <genre>Jazz</genre>
              <length>45:44</length>
              <year>1959</year>
       </cd>
       <cd stock="2">
              <title>In a Silent Way</title>
              <artist>Miles Davis</artist>
              <country>USA</country>
              <company>Columbia</company>
              <genre>Jazz</genre>
              <length>38:10</length>
              <year>1969</year>
       </cd>
       <cd stock="3">
              <title>Led Zeppelin IV</title>
              <artist>Led Zeppelin</artist>
              <country>UK</country>
              <company>Atlantic</company>
              <genre>Rock</genre>
              <length>42:33</length>
              <year>1971</year>
       </cd>
       <cd stock="0">
              <title>Grace</title>
              <artist>Jeff Buckley</artist>
              <country>USA</country>
              <company>Columbia</company>
              <genre>Rock</genre>
              <length>51:44</length>
              <year>1994</year>
       </cd>
       <cd stock="1">
              <title>My Generation</title>
              <artist>The Who</artist>
              <country>UK</country>
              <company>Brunswick</company>
              <genre>Rock</genre>
              <length>36:13</length>
              <year>1965</year>
       </cd>
</catalog>
```

4. Cocoon Basics

The Cocoon 2 Architecture is based on component pipelines, the elements in the pipeline are SAX events created by parsing XML documents. There are two major types of XML APIs for parsing XML:

Tree-based APIs: These map an XML document into an internal tree structure, then allow an application to navigate that tree. The Document Object Model (DOM) working group at the W3C maintains a recommended tree-based API for XML and HTML documents, and there are many such APIs from other sources as well.

Event-based APIs: An event-based API, on the other hand, reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree. The application implements handlers to deal with the different events, much like handling events in a graphical user interface. SAX is the best known example of such an API.

Tree-based APIs are useful for a wide range of applications, but they normally put a great strain on system resources, especially if the XML documents are large. In Cocoon th SAX API is used.

Pipelines & Components

There are 3 major Cocoon components: Generators, Transformers & Serializers. At the very minimum a pipeline consists of a generator and a serializer. Other components are: readers, selectors, matchers, actions, and pipes.

Generator

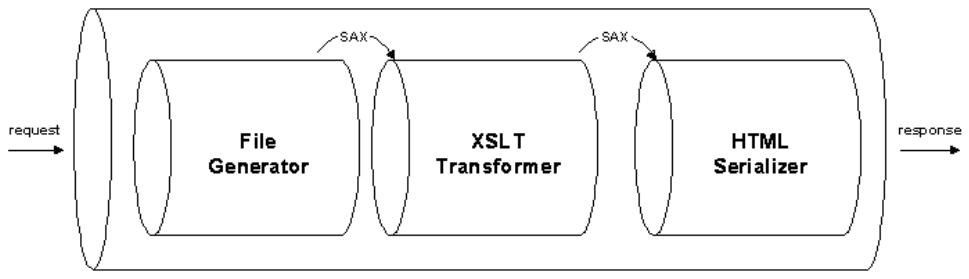
The Generator is the starting point for the pipeline. It is responsible for delivering SAX events down the pipeline. The simplest Generator is the FileGenerator: it takes a local XML document, parses it, and sends the SAX events down the pipeline. The Generator is constructed to be independent of the concept "file". If you are able to generate SAX events from another source, you can use that without having to go via a temporary file.

Transformer

A Transformer can be compared to an XSL: it gets an XML document (or SAX events), and generates another XML document (or SAX events). The simplest Transformer is the XalanTransformer: it applies an XSL to the SAX events it receives.

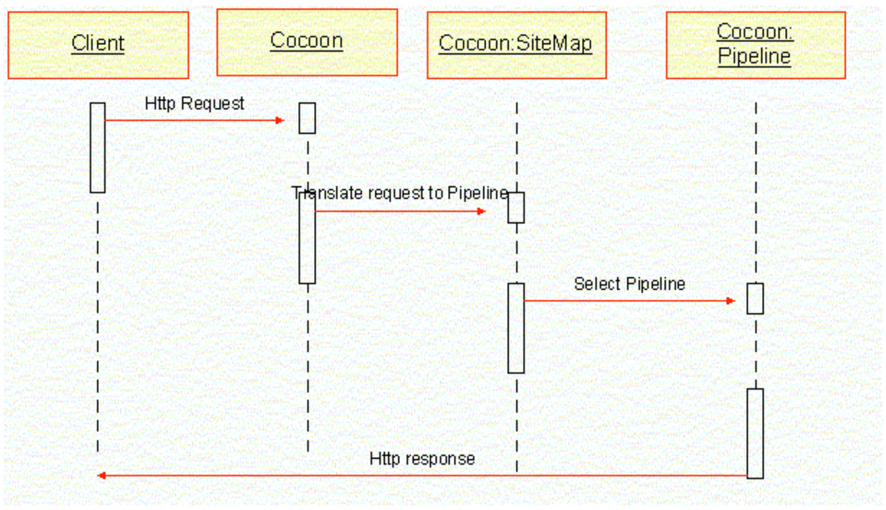
Serializer

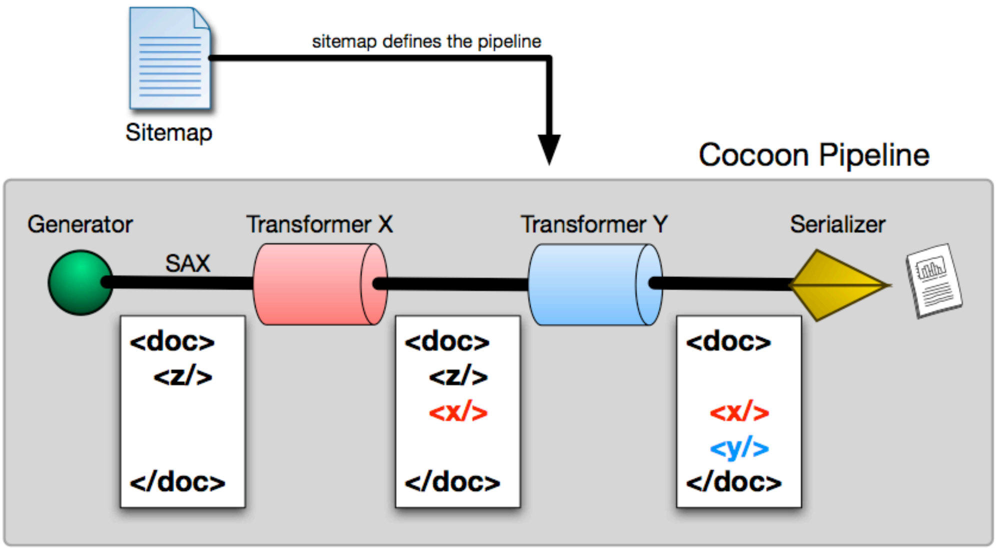
A Serializer is responsible for transforming SAX events to a presentation format. For actors looking at the back of the pipeline, it looks like a static file is delivered. So a browser can receive HTML, and will not be able to tell the difference with a static file on the filesystem of the server. Cocoon has built-in serializers for generating HTML, XML, PDF, VRML, WAP, and of course you can create your own serializers. The simplest Serializer is the XMLSerializer: it receives the SAX events from up the pipeline, and returns a "human-readable" XML file.



Sitemaps

Sitemaps are configuration documents in Cocoon that define available pipelines and components for a web application. The sitemap allows the incoming request URI to be matched with a particular pipeline that processes the request and creates the desired output.





Sitemap Structure

```
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
  <map:components>
    ...
  </map:components>
  <map:views>
    ...
  </map:views>
  <map:pipelines>
    <map:pipeline>
      <map:match>
        ...
      </map:match>
      ...
    </map:pipeline>
    ...
  </map:pipelines>
</map:sitemap>
```

Example Sitemap

This is a simplified example of a Cocoon sitemap (not functional)

```
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">

  <map:components>
    <map:generators>
      <map:generator name="docbook" src="com.atmire.cocoon.generation.DocbookGenerator"/>
    </map:generators>
    <map:transformers>
      <map:transformer logger="sitemap.transformer.xslt" name="xslt"
                       src="org.apache.cocoon.transformation.TraxTransformer"/>
    </map:transformers>
    <map:serlializers>
      <map:serializer name="xhtml" logger="sitemap.serializer.xhtml"
                      src="org.apache.cocoon.serialization.XMLSerializer"
                      mime-type="text/html; charset=utf-8">
        <doctype-public>-//W3C//DTD XHTML 1.0 Strict//EN</doctype-public>
        <doctype-system>http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd</doctype-
system>
        <encoding>UTF-8</encoding>
        <indent>yes</indent>
      </map:serializer>
    </map:serlializers>
  </map:components>

  <map:views/>

  <map:pipelines>
    <map:pipeline>
      <map:match pattern="*">
        <map:generate type="docbook">
          <map:parameter name="path" value="{1}"/>
        </map:generate>
        <map:transform type="xslt" src="docbook2xhtml.xsl"/>
        <map:serialize type="xhtml"/>
      </map:match>
    </map:pipeline>
  </map:pipelines>

</map:sitemap>
```

5. Solutions to exercises

Exercise 2.1. All genre nodes

```
//genre
```

Note: Using the full path instead of `//` is also allowed, in this case `/catalog/cd/genre`

Exercise 2.2. All cd titles

```
//cd/title/text()
```

Exercise 2.3. All cd nodes that are out of stock

```
//cd[@stock = 0]
```

Exercise 2.4. All cd nodes that have the genre 'Rock'

```
//cd[genre/text() = 'Rock']
```

Exercise 2.5. All cd titles that were made in the USA

```
//cd[country/text() = 'USA']/title/text()
```

Exercise 2.6. The years 'Miles Davis' has released a cd

```
//cd[artist/text() = 'Miles Davis']/year/text()
```

Exercise 2.7. All cd titles by the company 'Columbia' of the genre 'Rock'

```
//cd[company/text() = 'Columbia'][genre/text() = 'Rock']/title/text()
```

Exercise 2.8. All cd titles that are in stock, that don't have the genre 'Jazz'.

```
//cd[@stock > 0][not(genre/text() = 'Jazz')]/title/text()
```

Exercise 3.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
	<xsl:output method="html"/>
	<xsl:template match="/">
		<html>
			<body>
				<ul>
					<xsl:apply-templates/>
				</ul>
			</body>
		</html>
	</xsl:template>
	<xsl:template match="//cd">
		<li>
			<xsl:value-of select="artist/text()"/>
			<xsl:text>, </xsl:text>
			<xsl:value-of select="title/text()"/>
			<xsl:text> - </xsl:text>
			<xsl:choose>
				<xsl:when test="@stock > 0">
					<xsl:text>in stock</xsl:text>
				</xsl:when>
				<xsl:otherwise>
					<xsl:text>out of stock</xsl:text>
				</xsl:otherwise>
			</xsl:choose>
		</li>
	</xsl:template>
</xsl:stylesheet>
```