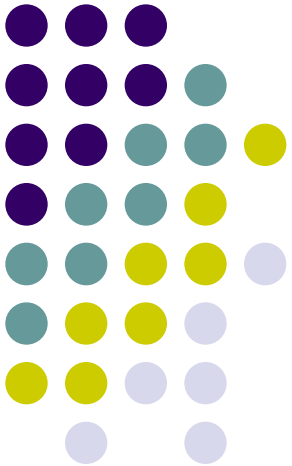
NPRG039

Advanced Aspects and New Trends in XML (and Related) Technologies

RNDr. Irena Holubová, Ph.D.

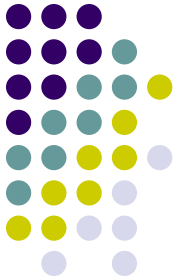
holubova@ksi.mff.cuni.cz



Lecture 7. XProc

http://www.ksi.mff.cuni.cz/~holubova/NPRG039/

What is XProc?

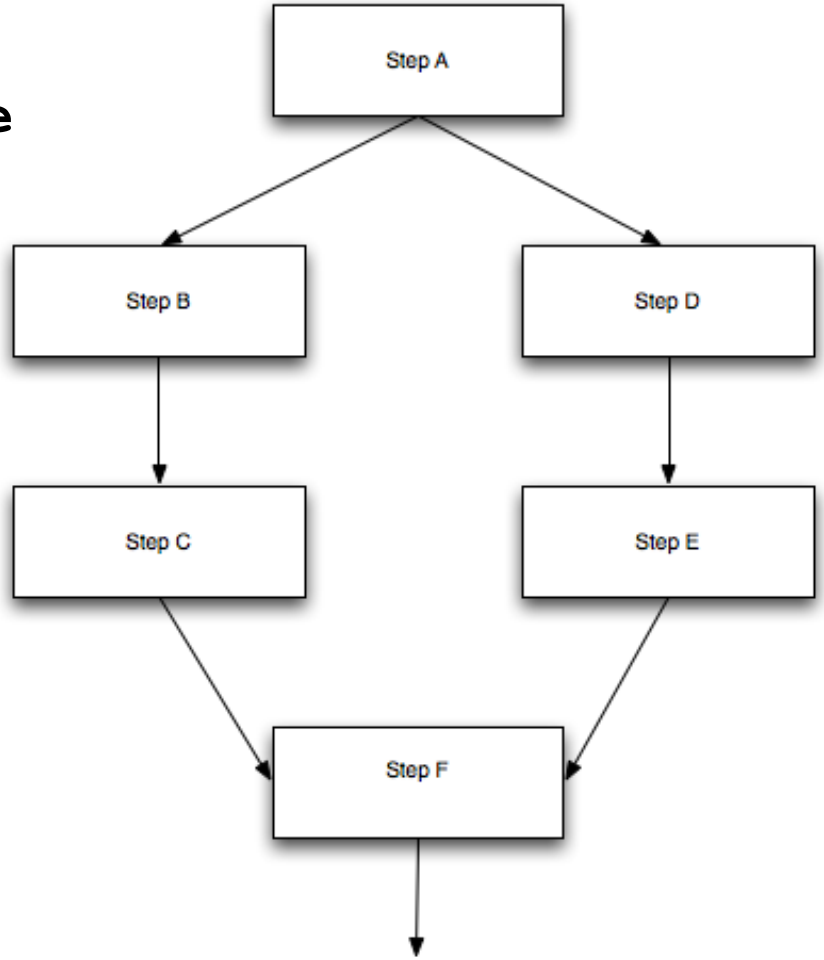


- A tool to generate XML-oriented workflows
- Language in XML (W3C Recommendation)
  - Namespaces:
    - http://www.w3.org/ns/xproc, prefix: p
    - http://www.w3.org/ns/xproc-step, prefix: c
    - http://www.w3.org/ns/xproc-error, prefix: err
- Uses several XML technologies performed in a sequence within XML pipeline
  - XML pipeline = a sequence of operations (steps) to be performed on a collection of XML input documents
    - e.g., XSL transformations, XML Schema validations, …
- XProc processor – executes the operations

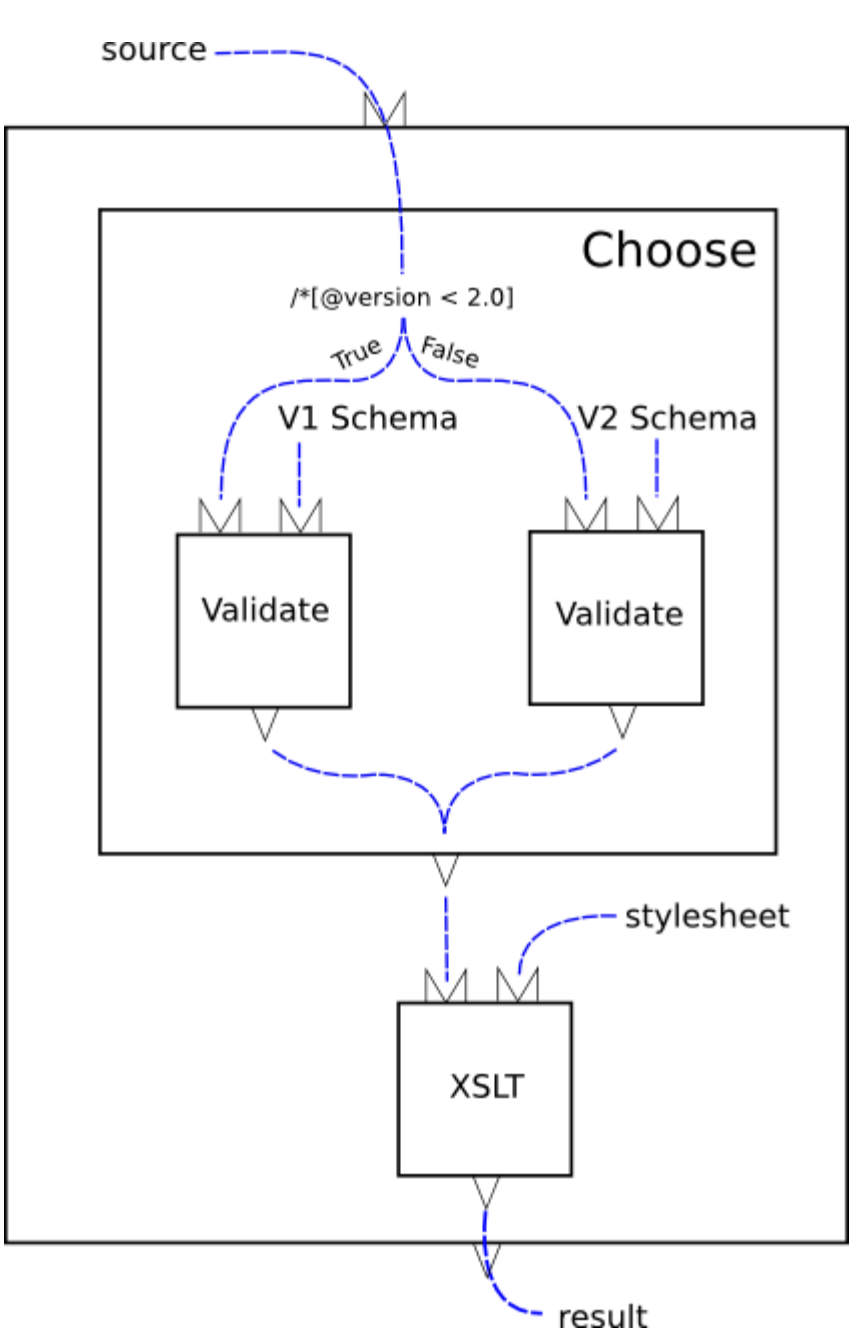
http://www.w3.org/TR/xproc/

XProc Pipeline

- XProc pipeline = XML document having a root element **`p:pipeline`** (or **`p:declare-step`**)
  - Contains one or more steps
- Steps can
  - Have options and parameters
  - Be nested
  - Declare and use variables
    - As in XSLT – immutable
  - …
- Steps are mutually connected
  - Implicitly or explicitly

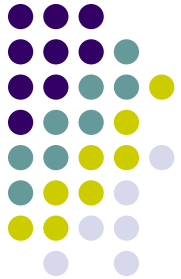


XProc Pipeline Example



```
<p:pipeline xmlns:p="http://www.w3.org/ns/xproc"
            version="1.0">
  <p:choose>
    <p:when test="/*[@version &lt; 2.0]">
      <p:validate-with-xml-schema>
        <p:input port="schema">
          <p:document href="v1schema.xsd"/>
        </p:input>
      </p:validate-with-xml-schema>
    </p:when>
    <p:otherwise>
      <p:validate-with-xml-schema>
        <p:input port="schema">
          <p:document href="v2schema.xsd"/>
        </p:input>
      </p:validate-with-xml-schema>
    </p:otherwise>
  </p:choose>
  <p:xslt>
    <p:input port="stylesheet">
      <p:document href="stylesheet.xsl"/>
    </p:input>
  </p:xslt>
</p:pipeline>
```

XProc Pipeline



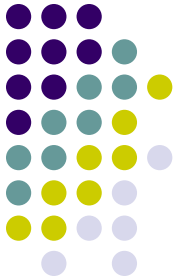
- Pipeline/step input/output: zero or more XML documents
  - Documents flow between steps along their connections
- Inputs of a step come:
  - From the web
  - From a pipeline document
  - From the inputs to the pipeline itself
  - From the outputs of other steps in the pipeline
- Outputs from a step are:
  - Consumed by other steps
  - Outputs of the pipeline as a whole
  - Discarded

XProc Step Example

```
<?xml version="1.0" encoding="UTF-8"?>
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc"
    xmlns:c="http://www.w3.org/ns/xproc-step"
    version="1.0">
  <p:input  port="source" sequence="true"/>
  <p:output port="result" sequence="true"/>
  <p:identity/>
</p:declare-step>
```

- **`p:declare-step`** – declares a step
- **`sequence="true"`** – several input documents may be used
- **`p:identity`** – returns identity
  - Usually for testing purposes

XProc Pipeline Example



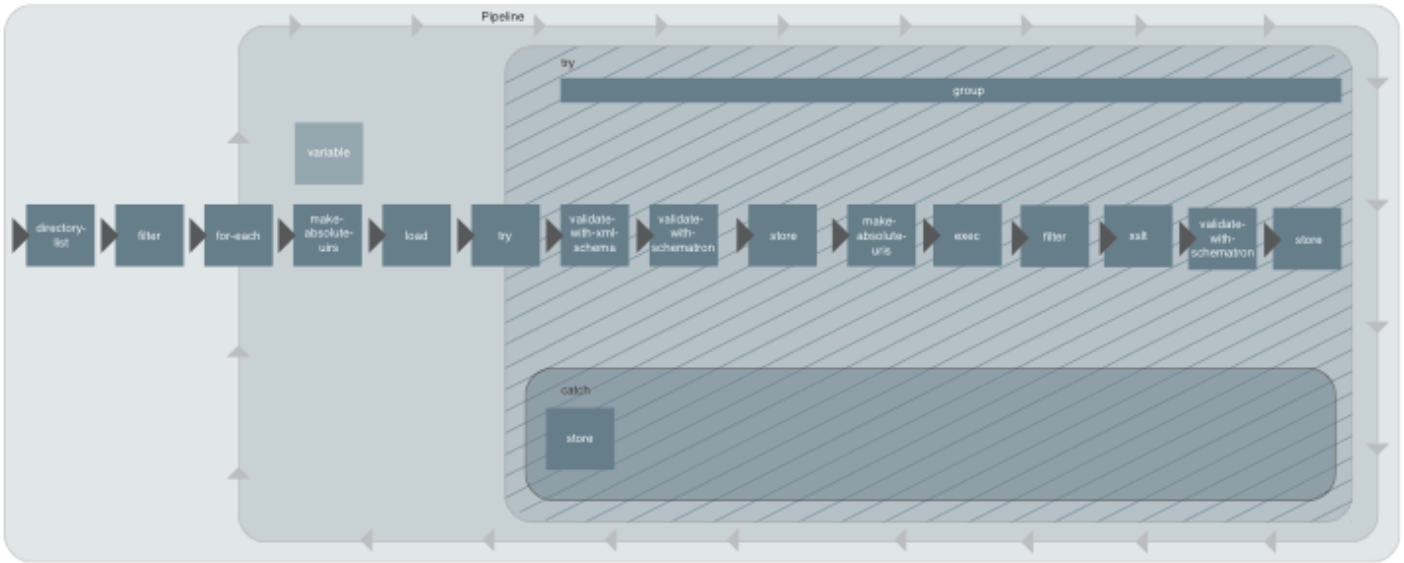
```
<?xml version="1.0" encoding="UTF-8"?>
<p:pipeline xmlns:p="http://www.w3.org/ns/xproc"
    xmlns:c="http://www.w3.org/ns/xproc-step"
    version="1.0"
    name="TrivialPipeline">
  <p:identity/>
</p:pipeline>
```

- **`p:pipeline`** – declares a pipeline
  - A special kind of step
  - Automatically equipped with:
    - Primary input port called **`source`**
    - Parameter port called **`parameter`**
    - Primary output port called **`result`**

required by most steps

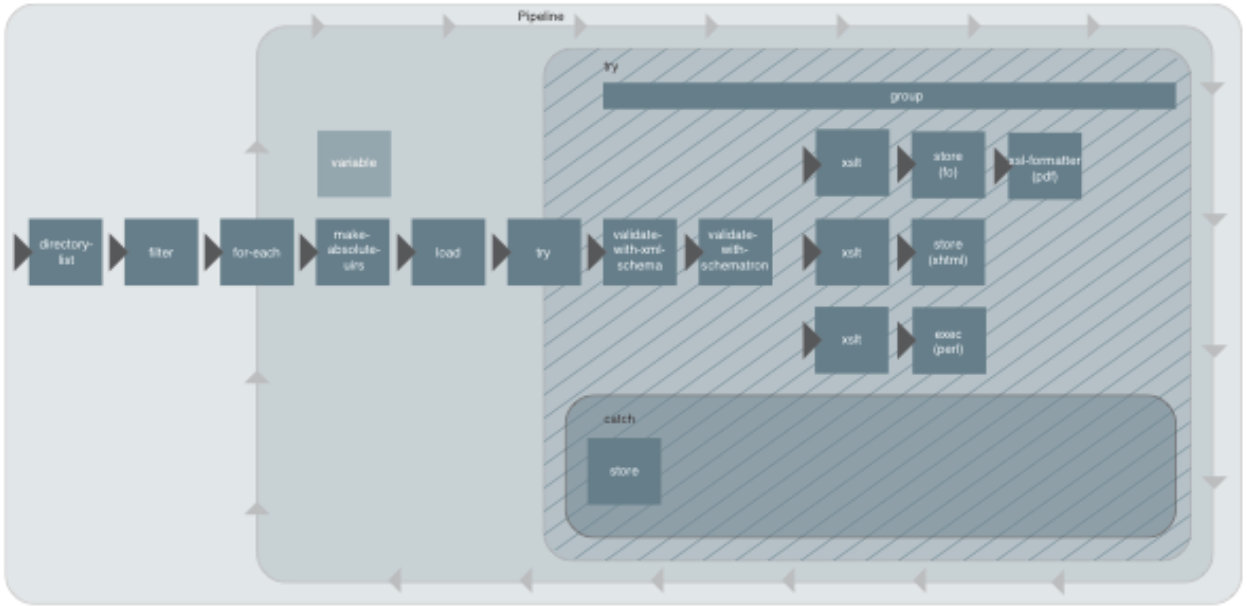
Sample Scenario 1: Data Migration

- **Situation:** Two companies have the intention to merge
  - Both have accumulated enormous amounts of XML data
  - The data have to be combined
- Can be realised using several XSLT transformations
  - Before the transformation the XML documents are checked against an XML Schema
  - Afterwards against a Schematron schema

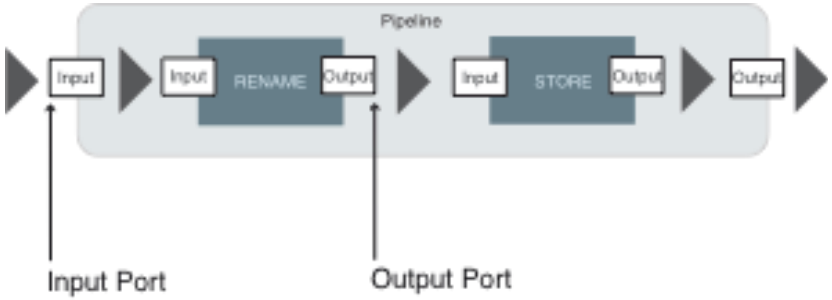


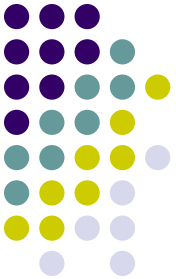
Sample Scenario 2: Cross Media Publishing

- **Situation:** A company working in the field of publishing sells its publications in several formats
  - e.g., PDF, XHTML, …
- Raw data is provided in the form of XML documents
- They are passing various quality assurance processes
  - i.e., validated against XML Schema/Schematron files
- Then they are transformed into the desired formats by XSLT transformations



Ports



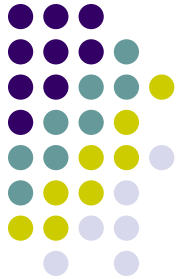


- Define the inputs (**`p:input`**) and outputs (**`p:output`**) of pipelines / steps
- Have to be named
  - Usually "source" or "result"
- In built-in steps they are usually required
- Attributes:
  - **`kind`** – **`document`** / (application-specified) **`parameter`**
  - **`select`** – XPath expression filtering data in an input XML document
- Subelements:
  - **`p:document`** – resource from a file
  - **`p:data`** – resource from a URI

see parameters

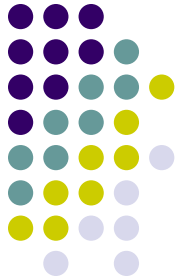
```
<p:input port="source">
   <p:document href="FilmCollection.xml"/>
</p:input>
```

Types of Ports



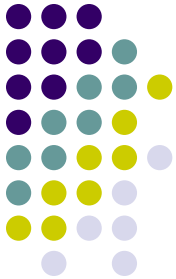
- **Primary** (**`primary="true"`**)
  - At most one input / output port
  - If there is a single input / output port, then it is primary by default
- **Default Readable Port**
  - If the input is not explicitly connected
  - Can be undefined (if not found)
  - For the first step in a sub-pipeline is its parent's primary input port
  - For each step after the first in a sub-pipeline is the primary output port of its preceding sibling

Port Binding



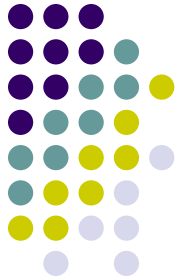
- Ports have to be bound (connected) with their corresponding sources
  - Input: source documents
  - Output: to which step / pipeline it belongs
- The input data may:
  - Originate from the output of a previous step (**`p:pipe`**)
    - Specify name of a **`step`** and its **`port`**
  - Be directly defined by **`p:inline`**
  - Originate from a document sequence
    - Input of the pipeline
  - Be bound by an external file (using URI)
  - Be explicitly empty (**`p:empty`**)
    - Read nothing

Port Binding



```
<?xml version="1.0" encoding="UTF-8"?>
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc"
                xmlns:c="http://www.w3.org/ns/xproc-step"
                version="1.0" name="ExamplePipeline">
  <p:input port="source">
    <p:inline>
      <doc>Hello World</doc>
    </p:inline>
  </p:input>
  <p:output port="result"/>
  <p:identity>
    <p:input port="source">
      <p:pipe port="source" step="ExamplePipeline"/>
    </p:input>
  </p:identity>
</p:declare-step>
```

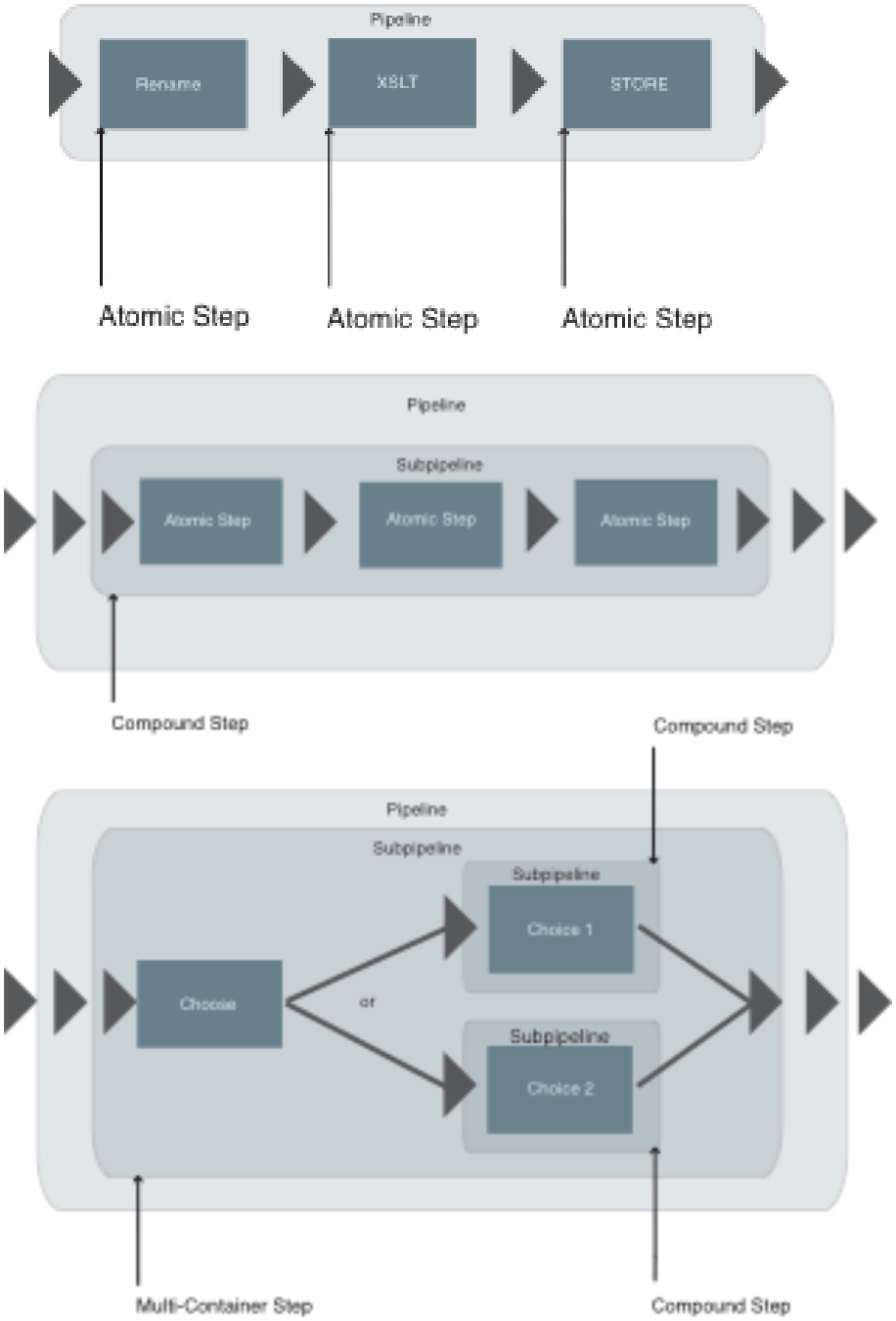
Kinds of Steps



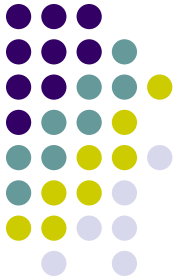
- Specification:
  - Required (built-in, standard) – have to be supported by the appropriate implementations
    - e.g., Calabash, Calumet
  - Optional – the processors do not necessarily have to use them
- User-defined steps (with own namespace):
  - Put together from the existing XProc steps
  - Developed in a high-level language
    - Preferably in Java – to be embedded into the existing processor(s)

Kinds of Steps

- Complexity:
  - Atomic – single operations
    - No substructure
  - Compound – contain sub-pipelines
    - Arbitrary nesting
  - Multi-container – two or more alternative sub-pipelines



Atomic Step Example



- Example: All elements **`Title`** are filtered out of the input document **`FilmCollection.xml`**
  - **`select`** attribute – XPath expression to filter the data

```
<?xml version="1.0" encoding="UTF-8"?>
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc"
    xmlns:c="http://www.w3.org/ns/xproc-step"
    version="1.0">
  <p:input port="source">
    <p:document href="FilmCollection.xml"/>
  </p:input>
  <p:output port="result" sequence="true"/>
  <p:filter select="/FilmCollection/Film/Title"/>
</p:declare-step>
```

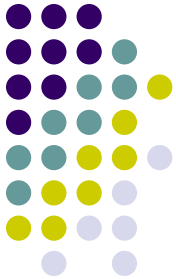
Compound Step Example

`p:for-each`

- **`p:for-each`** step is a loop implementation
- Documents which are assigned to this step are processed sequentially
- The input has to be provided:
  - By a preceding step as a sequence
  - By **`p:iteration-source`**
    - As attribute an XPath expression can be used which addresses desired contents from the input port
    - Alternatively, contents can also be loaded or created via **`p:document`**, **`p:inline`** and **`p:data`**

Compound Step Example

p:for-each



- Elements **Year** of the source document are processed loopwise
- Step **p:rename** renames the element as **Date**

```
<?xml version="1.0" encoding="UTF-8"?>
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc"
    xmlns:c="http://www.w3.org/ns/xproc-step"
    version="1.0">
  <p:input port="source">
    <p:document href="FilmCollection.xml"/>
  </p:input>
  <p:output port="result" sequence="true"/>
  <p:for-each>
    <p:iteration-source select="//Year"/>
    <p:output port="result"/>
    <p:rename match="/Year" new-name="Date"/>
  </p:for-each>
</p:declare-step>
```

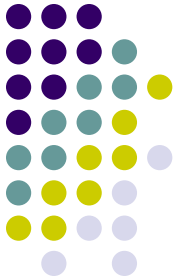
Compound Step Example

`p:viewport`

- Parts of a document can be selected by an XSLT match (attribute **`match`**) expression to process them in its sub-pipeline
  - Alternatively, the desired content can also be provided by subelement **`p:viewport-source`**
    - By its subelements **`p:pipe`**, **`p:document`**, **`p:inline`** or **`p:data`**
- Steps:
  1. Each matching node in the source document is wrapped in a document
  2. The documents are provided, one at a time, to the viewport's sub-pipeline
     - On a port named “current”
- Result: a copy of the original document where the selected sub-trees have been replaced by the results of applying the sub-pipeline to them

Compound Step Example

p:viewport

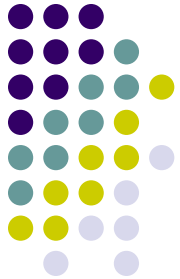


- All elements **Title** are renamed as **test** (a standard behaviour of **p:wrap**)

```
<?xml version="1.0" encoding="UTF-8"?>
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc"
                xmlns:c="http://www.w3.org/ns/xproc-step"
                version="1.0">
  <p:input port="source">
    <p:document href="FilmCollection.xml"/>
  </p:input>
  <p:output port="result"/>
  <p:viewport match="/FilmCollection/Film/Title">
    <p:wrap match="/" wrapper="test"/>
  </p:viewport>
</p:declare-step>
```

Compound Step Example

`p:group`



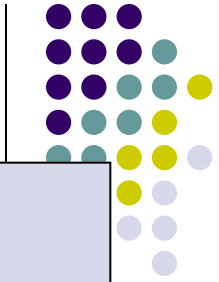
- Wrapper for accumulation of several steps
  - Encapsulates the behavior of its sub-pipeline

Compound Step Example

p:group

```
<?xml version="1.0" encoding="UTF-8"?>
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc"
                xmlns:c="http://www.w3.org/ns/xproc-step"
                version="1.0">
  <p:input port="source">
    <p:empty/>
  </p:input>
  <p:output port="result"/>
  <p:group>
    <p:identity>
      <p:input port="source">
        <p:inline>
          <doc>Example</doc>
        </p:inline>
      </p:input>
    </p:identity>
    <p:identity/>
  </p:group>
</p:declare-step>
```

Multi-container Step Example



```
<?xml version="1.0" encoding="UTF-8"?>
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc"
    xmlns:c="http://www.w3.org/ns/xproc-step"
    version="1.0">
  <p:input port="source">
    <p:document href="FilmCollection.xml"/>
  </p:input>
  <p:output port="result">
    <p:empty/>
  </p:output>
  <p:choose>
    <p:when test="count(//Film) &gt; 1">
      <p:store href="morethanonefile.xml"/>
    </p:when>
    <p:when test="count(//Film) = 1">
      <p:store href="onefile.xml"/>
    </p:when>
    <p:otherwise>
      <p:store href="standard.xml"/>
    </p:otherwise>
  </p:choose>
</p:declare-step>
```

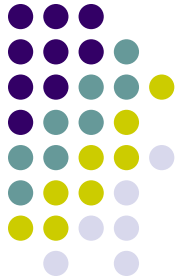
output into an XML file

Variables

- Can only be generated in compound steps
- Attributes:
  - **`name`**
  - **`select`** = XPath expression expressing the value
    - Can also be loaded with **`p:empty`**, **`p:pipe`**, **`p:document`**, **`p:inline`** or **`p:data`**

```
<?xml version="1.0" encoding="UTF-8"?>
<p:declare-step
    xmlns:p="http://www.w3.org/ns/xproc"
    xmlns:c="http://www.w3.org/ns/xproc-step"
    version="1.0">
  <p:input port="source">
    <p:inline>
      <block>
        <fileName>test.xml</fileName>
        <text>sample text.</text>
      </block>
    </p:inline>
  </p:input>
  <p:output port="result">
    <p:empty/>
  </p:output>
  <p:group>
    <p:variable name="Inline"
                select="block/fileName"/>
    <p:identity/>
    <p:store>
      <p:with-option name="href"
                     select="$Inline"/>
    </p:store>
  </p:group>
</p:declare-step>
```

Options



- Can be declared (using **`p:option`**) on atomic or compound steps
  - Influence the behaviour of the step
- Value of an option can be specified by the caller invoking the step
  - As its attributes
- Attributes:
  - **`name`**
  - **`required="true"`**
  - **`select`** – default value (XPath expression)
- An option can also be set within a step (**`p:with-option`**)

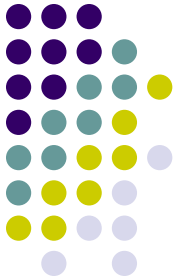
Options

declaration of step **p:delete** (in a library)

```
<p:declare-step type="p:delete">
  <p:input port="source"/>
  <p:output port="result"/>
  <p:option name="match" required="true"/>
</p:declare-step>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<p:declare-step
    xmlns:p="http://www.w3.org/ns/xproc"
    xmlns:c="http://www.w3.org/ns/xproc-step"
    version="1.0">
  <p:input port="source">
    <p:document href="FilmCollection.xml"/>
  </p:input>
  <p:output port="result"/>
  <p:delete match="/FilmCollection/Film/Cast"/>
</p:declare-step>
```

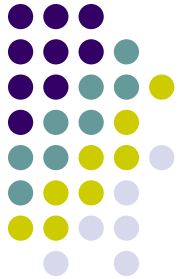
Options



```
<?xml version="1.0" encoding="UTF-8"?>
<p:declare-step
    xmlns:p="http://www.w3.org/ns/xproc"
    xmlns:c="http://www.w3.org/ns/xproc-step"
    version="1.0">
  <p:input port="source">
    <p:document href="FilmCollection.xml"/>
  </p:input>
  <p:output port="result" sequence="true"/>
  <p:filter>
     <p:with-option name="select"
                    select="'/FilmCollection/Film/Title'">
  </p:with-option>
  </p:filter>
</p:declare-step>
```

Informs XProc processor not to evaluate the XPath expression, but to hand it over to the step

Parameters



- The main difference: not declared beforehand
  - Names are unknown to the author of pipeline
- The most common use: to pass parameter values to XSLT stylesheets
- Make XProc a bit complicated…
- What happens:
  1. The parameters are grouped into element **`c:param-set`**
  2. The element is provided to the step on the respective input port (of type **`parameter`**)

param-decl.xpl

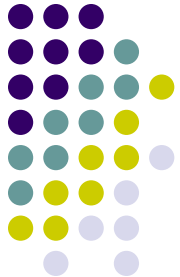
```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc"
                name="main"
                version='1.0'>
  <p:input port="source"/>
  <p:input port="parameters" kind="parameter"/>
  <p:output port="result">
    <p:pipe step="style" port="result"/>
  </p:output>

  <p:xslt name="style">
    <p:input port="stylesheet">
      <p:document href="docbook.xsl"/>
    </p:input>
    <p:input port="parameters">
      <p:pipe step="main" port="parameters"/>
    </p:input>
  </p:xslt>
</p:declare-step>
```

```
<c:param-set>
  <c:param name="body.font.family"
           value="sans-serif"/>
</c:param-set>
```

```
calabash -isource=dbdoc.xml
-pbody.font.family=sans-serif param-decl.xpl
```

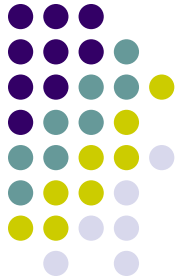
Errors



- A try/catch is specified by the element **`p:try`**
  - A multi-container step that isolates a pipeline
- Prevents dynamic errors that arise within it from being exposed to the rest of the pipeline
- Two parts:
  - **`p:group`** – represents the initial sub-pipeline
  - **`p:catch`** – recovery (or “catch”) sub-pipeline
- Behaviour:
  1. The initial sub-pipeline is executed
  2. If no errors occur, the outputs of that pipeline are the outputs of the **`p:try`** step
  3. If any errors occur:
     1. **`p:try`** abandons the first sub-pipeline
     2. It discards any output that it might have generated
     3. it executes the recovery sub-pipeline

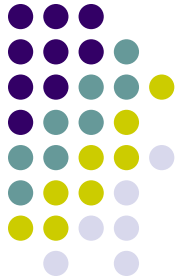
```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc"
                name="main" version='1.0'>
  <p:try>
    <p:group>
      <p:http-request>
        <p:input port="source">
          <p:inline>
            <c:request method="post"
                       href="http://example.com/form-action">
              <c:body content-type="application/x-www-form-
urlencoded">name=W3C&amp;spec=XProc</c:body>
            </c:request>
          </p:inline>
        </p:input>
      </p:http-request>
    </p:group>
    <p:catch>
      <p:identity>
        <p:input port="source">
          <p:inline>
            <c:error>HTTP Request Failed</c:error>
          </p:inline>
        </p:input>
      </p:identity>
    </p:catch>
  </p:try>
</p:declare-step>
```

And There Are Many Other Constructs…



- p:add-attribute, p:add-xml-base, p:compare, p:count, p:delete, p:directory-list, p:error, p:escape-markup, p:filter, p:http-request, p:insert, p:label-elements, p:load, p:make-absolute-uris, p:namespace-rename, p:pack, p:rename, p:replace, p:set-attributes, p:sink, p:split-sequence, p:store, p:string-replace, p:unescape-markup, p:unwrap, p:wrap, p:wrap-sequence, p:xinclude, p:xslt, …

References



- XProc
  - http://www.w3.org/TR/xproc/
- XProc Tutorial
  - http://www.data2type.de/en/xml-xslt-xslfo/xproc/
- XProc Processors
  - http://xproc.org/implementations/
- Norman Walsh: XML Pipelines – A Guide to XProc
  - http://xprocbook.com/book/book-1.html
  - Unfinished