xproc: a pipeline publishing language

I am not allergic to XML, quite the opposite. When I first started writing I did in Docbook XML and the associated XSLT stylesheets and have written my own stylesheets and schemas for personal projects.

Given that bit of background, XProc fills an interesting niche, a pipeline-based composition language that will allow processing of a document producing one or more output documents.

There are two versions of XProc, the original 1.0 and the newer 3.0. Many newer developments happened in 3.0 but, as of this writing, there are no processors that support the new version. I hope this will change soon.

Using version 1.0 for the examples, an XProc pipeline looks like this:

```
<p:declare-step
  xmlns:p="http://www.w3.org/ns/xproc"
  xmlns:c="http://www.w3.org/ns/xproc-step"
  version="1.0"> <!-- 1 -->
   <p:input port="source"> <p:input port="source"><!-- 2 -->    <doc>Hello
      </p:inline>
   </p:input>
   <p:output port="result"/> <!-- 3 -->
   <p:identi<doc><!-- 3 -->
</p:declare-step></p:declare-step><!-- 4 -->
</p:declare-step>
```

1. Declare the root element of the pipeline (`p:declare-step`), the name spaces for `xproc` and `xproc-step` and the version
2. The input for the pipeline, in this case it's an inline XML element
3. The output for the pipeline
4. Makes a direct copy of the input document and sends it to the ouput

This is the simplest pipeline possible. We'll continue to improve from here.

Using XProc 1.0 we seek to answer the following questions:

- Can we generate multiple outputs from the same source?
- Can we generate multiple outputs from a single pipeline or do we need multiple pipelines?
- Can we use third-party software installed on our local machines as part of the pipelines we create?
- How do we document pipelines?
- Can we use Markdown as a source document?

Note that this research dive will not cover all of XProc 1.0. We don't need the more esoterica validation (Schema and Schematron) or some of the programming structures available to the language.

Getting Started

The following example pipeline uses a single xslt step to convert an XML file into another XML file for further processing.

```
<?xml version="1.0" encoding="UTF-8"?>
<p:declare-step
    xmlns:p="http://ww<p:declare-step
    xmlns:p="http://www.w3.org/ns/xproc"
    xmlns:c="http://www.w3.org/ns/xproc-step"
    name="publishPipeline"
    version="1.0">ource"> <!-- 1 -->
            <p:document
              href="film-collection.xml"/>
        </p:input>
        <p:input port="stylesheet"> <p:document
              href="film-collection.xml"/><!-- 2 -->ist.xslt" />
        </p:input>
        <p:input port="parameters"> <!-- 3 -->
            <p:empty />
        </p:input>
</p:input><!-- 3 -->
            <p:empty />
        </p:input>
    </p:xslt>
    <p:store href="authors.xml"
```

```
      method="xml"
      indent="true"/>
</p:declare-step>
```

The xslt step takes three inputs:

1. The name of of the XML document we want to process
2. The name of the XSLT stylesheet (either version 1.0 or 2.0) that will transform the XSLT document
3. One or more parameters. This input can be specified as empty using the `<p:empty>` element

The `<p:store>` instruction takes mandatory attributes like `href` to indicate the name and location for the resulting file and `method` to indicate the type of file it is (XML in this case)

There's an optional `indent` attribute to indicate if the file should preserve indentation or not.

to summarize, the code in the example will take the XML file, process it with the given XSLT stylesheet and output the result to the given file.

Multiple Tasks in the same pipeline

Using the same concept as before we can create multiple sub-steps to accomplish different steps into a single pipeline.

These can use the same files or, as in the example, use different stylesheets and result files.

```
<?xml version="1.0" encoding="UTF-8"?>
<p:declare-step
    xmlns:p="http://ww<p:declare-step
    xmlns:p="http://www.w3.org/ns/xproc"
    xmlns:c="http://www.w3.org/ns/xproc-step"
    name="publishPipeline"
    version="1.0"><!-- 1 -->source">
            <p:document href="film-collection.xml"/>
```

```
        </p:input>
        <p:input port="stylesheet">
            <p:document href="authors_list.xslt" />
        </p:input>
        <p:input port="parameters">
            <p:empty />
        <p:document href="film-collection.xml"/>ef="authors.xml" method="x

    <p:xslt> <!-- 2 -->
        <p:in<p:xslt><!-- 2 -->
        <p:input port="source">
            <p:document href="film-collection.xml"/>
        </p:input>
        <p:input port="stylesheet">
            <p:document href="film-collection.xslt" />
        </p:input>
        <p:input port="parameters">
            <p:empty/>
        </p:input>
    </p:xslt>
    <p:store href="films.html" method="xhtml" indent="true"/>
</p:declare-step>
```

1. The first step is the same as the example in the earlier section
2. The second step in the pipeline is also the same as the example in the last section with different stylesheets and result values

Yes, you can do different transformations as part of the same pipeline.

The examples so far use different inputs but there's nothing stopping us from using the same input file in different transformation scenarios

Using third party tools to generate PDF

This is my favorite trick and it's a two part process:

1. Generate HTML from Markdown

2. Generate PDF from the HTML created in step 1

XProc 1.0 makes this harder than it needs to be as we'll have to execute external commands to accomplish the tasks.

XProc 3.0 has pre-defined custom steps that automate the process. The following pipeline example uses

```
<?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"
    name="publishPipeline"
    version="1.0">

<p:markdown-to-html>
  <p:input
    port="source">
</p:markdown-to-html>

<p:css-formatter>

<p:css-formatter>
```

```
<p:declare-step type="p:markdown-to-html">
  <p:input port="source" primary="true" content-types="text"/>
  <p:output port="result" primary="true" content-types="html"/>
  <p:option name="parameters" as="map(xs:QName, item()*)?"/>
</p:declare-step>

<p:declare-step type="p:css-formatter">
  <p:input port="source" content-types="xml html"/>
  <p:input port="stylesheet" content-types="text" sequence="true"/>
  <p:output port="result" content-types="any"/>
  <p:option name="parameters" as="map(xs:QName,item()*)?"/>
  <p:option name="content-type" as="xs:string?"/>
</p:declare-step>
```

Links, toools and references

- Tools
    - Oxygen XML editor
- XProc 1.0
    - XProc Recommendation
    - Implementations
        - Calabash maintained by Norman Walsh
        - Calumet, EMC's XProc implementation
        - MorganaXProc developed by <xml-project />
        - QuiXProc, Innovimax's (GPL) version in Java implementing Streaming and Parallel processing
        - Tubular (LGPL) maintained by Herve Quiroz
        - xprocxq, XQuery old implementation on top of eXist
        - xproc.xq, XQuery implementation on top of MarkLogic
- XProc 3.0
    - XProc 3.0 Last Call Draft
    - XProoc 3.0 Standard Step Library Last Call Draft
    - XProc 3.0: Paged Media Steps
    - XProc 3.0: text steps
    - Implementations. **Note that, as of this writing, there are currently no XProc 3.0 implementations available. The ones listed here are under development**
        - Calabash maintained by Norman Walsh
        - MorganaXProc developed by <xml-project />