yax - eine XProc (XML Pipeline) Implementierung

Joerg Moebius

Die Dokumentaton befindet sich im Aufbau.
Release: 0.11.0 / 2008-03-03

Inhaltsverzeichnis

Was ist yax?

yax ist eine Java Implementiierung der XProc Spezifikation, einer XML Pipeline Sprache (XProc: An XML Pipeline Language W3C Working Draft 17 November 2006 [http://www.w3.org/TR/2006/WD-xproc-20061117/]), mit der XProc Skripte wie das folgende verarbeitet werden können.

```
<?xml version="1.0" encoding="UTF-8"?>
<xproc:pipeline name="pipe1"
   xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
   xmlns:yax="http://opsdesign.eu/yax/1.0">
   <xproc:step name="trans1" type="xproc:XSLT"
      yax:description="transforms 'a*' elements to 'b*' elmenents.">
      <xproc:input port="stylesheet" href="test/transformation1.xsl"/>
   </xproc:step>
   <xproc:step name="trans2" type="xproc:XSLT"
      yax:description="transforms 'b*' elements to 'c*' elmenents.">
      <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
      <xproc:parameter name="transformer" value="Saxon6" />
   </xproc:step>
   <xproc:step name="trans3" type="xproc:XSLT"
      yax:description="transforms 'c*' elements to 'd*' elmenents.">
      <xproc:input port="stylesheet" href="test/transformation3.xsl"/>
      <xproc:parameter name="transformer" value="XSLTC" />
   </xproc:step>
</xproc:pipeline>
```

Schnellstart

Voraussetzungen

Folgende Software muss verfügbar (installiert) sein:

- Java (Version 5 or above)
- Graphviz - Graph Visualization Software [http://www.graphviz.org/],

  if want to use the visualisation features (pipeline trace, port trace).
- further XSLT processors (like saxon, xalan et al),

  if you want to use other processor(s) than the one (XSLTC), which comes with java.

Installation

Laden Sie yax http://yax.sourceforge.net (hier der Menüpunkt 'download') herunter und entpacken Sie die Datei in ein beliebiges <*yax-installation-directory*>.

Konfiguration

Erster Einsatz

```
# java -classpath lib/yax-0.7.jar net.sf.yax.Yax examples/example1.xproc examples/example1.xml
```

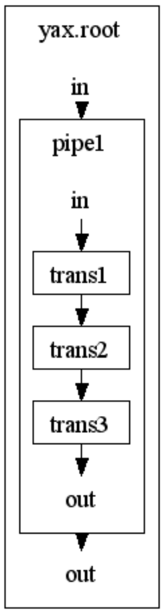
```
default transformer is set to system transformer.
automaticOutput of pipe trace is set to 'yes'.
automaticOutput of port trace is set to 'yes'.
suppress of process is set to 'no'.
step 'trans1': transformer Apache Software Foundation (Xalan XSLTC)(Version 1.0) is used.
step 'trans2': transformer Apache Software Foundation (Xalan XSLTC)(Version 1.0) is used.
step 'trans3': transformer Apache Software Foundation (Xalan XSLTC)(Version 1.0) is used.
Yax run sucessful completed.
```

- 1) <*input filename*>.output.xml
- 2) <*pipeline filename*>.config.xml

```
<yax:configuration xmlns:yax="http://www.opsdesign.eu/yax/1.0">
  <!--configuration description generated by Yax - Do not edit by hand-->
  <pipeline name="pipe1">
    ..
    <step name="trans1">
      <output port="out" sequence="no" yax:creator="implementation.xproc.standard"/>
      <input port="in" sequence="no" yax:creator="implementation.xproc.standard">
        <yax:connection port="in" yax:component="pipe1"/>
      </input>
      <input port="stylesheet" sequence="no" yax:creator="implementation.xproc.standard"/>
      <input href="examples/transformation1.xsl" port="stylesheet" yax:creator="pipeline"/>
    </step>
    ...
  </pipeline>
  <xproc:pipeline-library name="xproc.options" xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0">
    <xproc:declare-step-type type="Rename" yax:description="">
      <xproc:output port="out" select="" sequence="no" yax:bySourceRequired="no"/>
      <xproc:input port="in" select="" sequence="no" yax:bySourceRequired="no"/>
      <xproc:parameter name="name" required="yes" yax:values="{$any}"/>
      <xproc:parameter name="select" required="yes" yax:values="{$xpathExpression}"/>
    </xproc:declare-step-type>
    ...
  </xproc:pipeline-library>
  <xproc:pipeline-library name="xproc.standard" xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0">
    ...
  </xproc:pipeline-library>
  <xproc:pipeline-library name="yax.standard" xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0">
    ...
  </xproc:pipeline-library>
</yax:configuration>
```

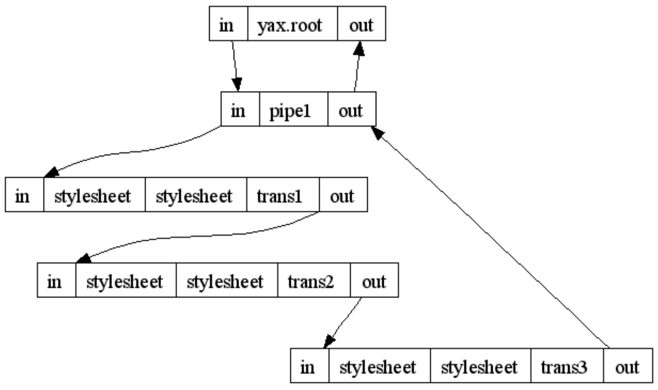
- 3) <*pipeline filename*>.pipeTrace.png

Abbildung 1. Komprimierte grafische Darstellung der Pipeline 'example1.xproc'.



- 4) *<pipeline filename>*.portTrace.png

Abbildung 2. Grafische Darstellung der Pipeline 'example1.xproc'.



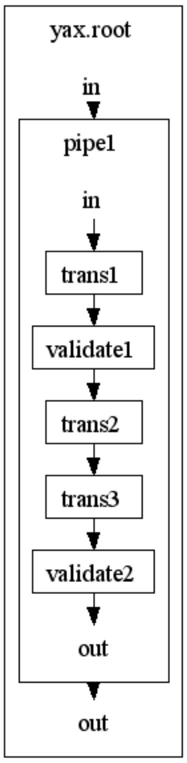
Ansatz

Der Kerngedanke von yax ...ToDo

Eingabe / Ausgabe

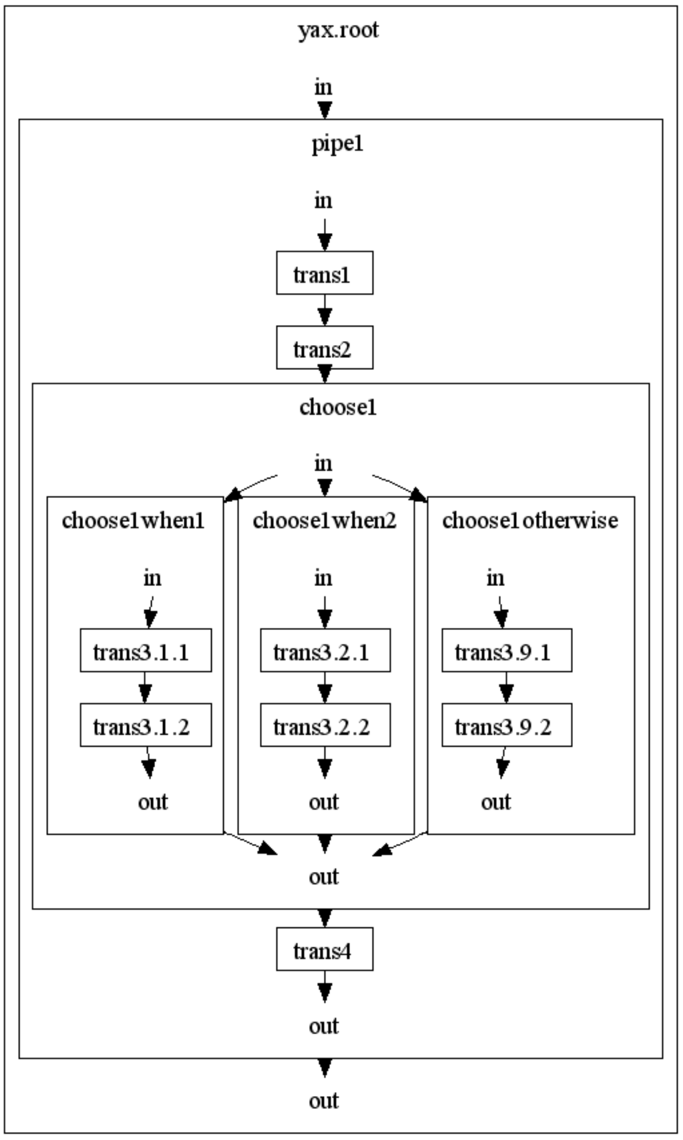
```
<xproc:pipeline name="pipe1"
   xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
   xmlns:yax="http://opsdesign.eu/yax/1.0">
   <xproc:step name="trans1" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation1.xsl"/>
   </xproc:step>
   <xproc:step name="validate1" type="xproc:Validate"/>
   <xproc:step name="trans2" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
   </xproc:step>
   <xproc:step name="trans3" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation3.xsl"/>
   </xproc:step>
   <xproc:step name="validate2" type="xproc:Validate"/>
</xproc:pipeline>
```

Abbildung 3. Einfache 'geschlossene' Pipeline



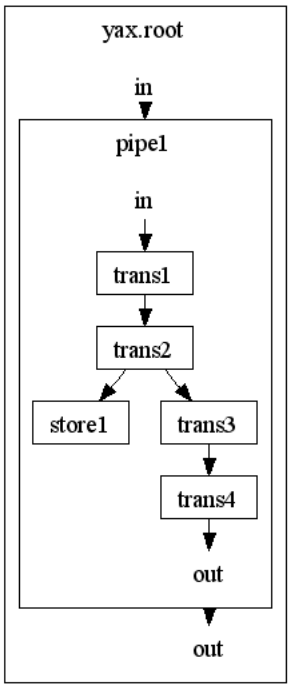
```
<xproc:pipeline name="pipe1"
   xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
   xmlns:yax="http://opsdesign.eu/yax/1.0">
   <xproc:step name="trans1" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation1.xsl"/>
   </xproc:step>
   <xproc:step name="trans2" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
   </xproc:step>
   <xproc:choose name="choose1">
      <xproc:when name="choose1when1">
         <xproc:step name="trans3.1.1" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation3.xsl"/>
         </xproc:step>
         <xproc:step name="trans3.1.2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation3.xsl"/>
         </xproc:step>
      </xproc:when>
      <xproc:when name="choose1when2">
         <xproc:step name="trans3.2.1" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation3.xsl"/>
         </xproc:step>
         <xproc:step name="trans3.2.2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation3.xsl"/>
         </xproc:step>
      </xproc:when>
      <xproc:otherwise name="choose1otherwise">
         <xproc:step name="trans3.9.1" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation3.xsl"/>
         </xproc:step>
         <xproc:step name="trans3.9.2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation3.xsl"/>
         </xproc:step>
      </xproc:otherwise>
   </xproc:choose>
   <xproc:step name="trans4" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation4.xsl"/>
   </xproc:step>
</xproc:pipeline>
```

Abbildung 4. 'Geschlossene' Pipeline mit verzweigendem Construct



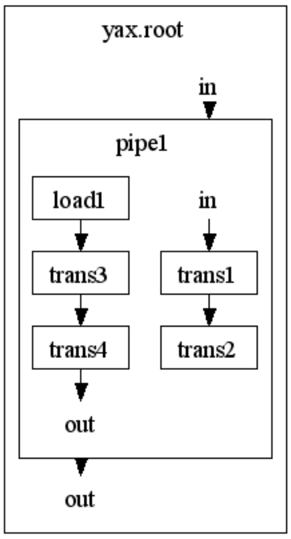
```
<xproc:pipeline name="pipe1"
   xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
   xmlns:yax="http://opsdesign.eu/yax/1.0">
   <xproc:step name="trans1" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation1.xsl"/>
   </xproc:step>
   <xproc:step name="trans2" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
   </xproc:step>
   <xproc:step name="store1" type="xproc:Store"/>
   <xproc:step name="trans3" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation4.xsl"/>
   </xproc:step>
   <xproc:step name="trans4" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation4.xsl"/>
   </xproc:step>
</xproc:pipeline>
```

Abbildung 5. 'Nach hinten geöffnete' Pipeline mit Ausgabe Komponente



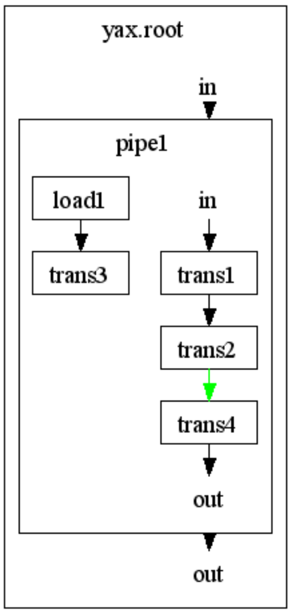
```
<xproc:pipeline name="pipe1"
   xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
   xmlns:yax="http://opsdesign.eu/yax/1.0">
   <xproc:step name="trans1" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation1.xsl"/>
   </xproc:step>
   <xproc:step name="trans2" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
   </xproc:step>
   <xproc:step name="load1" type="xproc:Load"/>
   <xproc:step name="trans3" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation4.xsl"/>
   </xproc:step>
   <xproc:step name="trans4" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation4.xsl"/>
   </xproc:step>
</xproc:pipeline>
```

Abbildung 6. 'Nach vorn geöffnete' Pipeline mit Eingabe Komponente



```
<xproc:pipeline name="pipe1"
   xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
   xmlns:yax="http://opsdesign.eu/yax/1.0">
   <xproc:step name="trans1" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation1.xsl"/>
   </xproc:step>
   <xproc:step name="trans2" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
   </xproc:step>
   <xproc:step name="load1" type="xproc:Load"/>
   <xproc:step name="trans3" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation4.xsl"/>
   </xproc:step>
   <xproc:step name="trans4" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation4.xsl"/>
      <xproc:input port="in" step="trans2" source="out"/>
   </xproc:step>
</xproc:pipeline>
```

Abbildung 7. 'Nach vorn geöffnete' Pipeline mit Exliziter Verbindung



Extension Mechanism

All component type classes have to reside in a package called 'net.sf.yax.types'. The built-in components are physically located in the yax library file *yax-<version>.jar*. They are distributed on three pipeline-libraries:

- xproc-standard
- xproc-options
- yax-standard

yax depicts all existant components grouped by libraries in the pipeline config file.

you can extend yax with step implemented by yourown by simply extend the class net.sf.yax.types.StepType and place the class file(s) of this implementation into the *custom* directory. For further details please refer to 'references/custom steps'.

Verarbeitung und Fehlerbehandlung

ToDo

offene Punkte zur Spezifikation

offene Punkte zur Spezifikation

- Associating Documents with Ports / Specified by source

  Since constructs also have ports, their ports should also be reachable by referencing via 'by source'. So

```
<p:step name="expand" type="p:xinclude">
  <p:input port="document" step="otherstep" source="result"/>
</p:step>
```

  should become something like (better with a more handsome term for 'component'):

```
<p:step name="expand" type="p:xinclude">
  <p:input port="document" component="otherComponent" source="result"/>
</p:step>
```

- Associating Documents with Output Ports

  Are there use cases this kind of association makes sense?

- context of choose/when/otherwise

  the context should also be provided by a (e.g. reference) port.

- are there important reasons to differenciate between steps and constructs?

- Should a 'declare-construct-type' exists?

  In my understanding the only difference between constructs and steps is that steps treats the result while constructs only passes the result without any treatment. Otherwise both have to follow the rule for components.

  One can define custom 'step's. So should one also be able to define constructs.

- what expression language(s) (el) should be used in XProc scripts?

  For compatibility purposes an el should be defined by the specification. Otherwise the scripts runs only on processors which understands a certain el.

- Try/Catch

  Group within Try necessary?

  multiple Catch clauses would make sense.

- namespace uri for XProc

- base URI for XInclude and Store

  A pipeline might has more than one input document simultaniously via the root input and loads. How to determine a single base URI?

- Load / Store

  As Load should provide an unambiguous content on its *result* port it must not have other inputs than the source referenced by href.

  Store provides (or better delivers) an unambigous content to the external realm via the href. If Store would provide this content simulantiously an its *result* the pipeline chain could be kept closed at a Store step point.

Leistungsumfang

Tabelle 1. Leistungsumfang

Leistungsmerkmal	**Beschreibung**
Allgemeine Leistungsmerkmale	
logging	Sie können log4j oder ein anderes Loggin Subsystem nutzen. Wenn kein subsystem bestimmt ist, protokolliert yax auf die system ports.
Implementierung der Constructs	

Leistungsmerkmal	**Beschreibung**
<p:pipeline>	
<p:choose>/<p:when>/<p:choose>	
<p:try>/<p:group>/<p:catch>	
Implementierung der Steps	
<p:XSLT>	
<p:XInclude>	
<p:Load>	
<p:Store>	

Status

Tabelle 2. Status der Implementierung

Leistungsmerkmal	**Status**	**Bemerkung**
Allgemeine Themen		
inputs and outputs	in Arbeit	implizite ports realisiert.
property / property files	erledigt	
preferences / preference file	erledigt	
Kernkonstrukte der Sprache		
Pipeline	erledigt	
Group	erledigt	same behaviour as Pipeline.
For-Each	geplant	
Viewport	geplant	
Choose/When/Otherwise	erledigt	first usage of explicit reference ports: if exists reference ports of when clauses can refer to the reference port of the choose construct (or obviously th another reference port)
Try/Group/Catch	erledigt	extended with differentiating catch clauses.
Import	geplant	
Pipeline-Library	erledigt	yax builds libraries based on the current implementation (see pipeline config file). CRs expected.
Kern Schritte		
Identity	erledigt	
XSLT	erledigt	CRs expected.
XInclude	erledigt	
Serialize	in Arbeit	
Parse	in Arbeit	
Load	erledigt	will be extended with access to database server and other data services.

Leistungsmerkmal	**Status**	**Bemerkung**
Store	erledigt	will be extended with access to database server and other data services.
Subelemente der Components		
input	(erledigt)in Arbeit	
output	(erledigt)in Arbeit	
Parameter	erledigt	
input-parameter	??	didn't understood.
Mikrooperationsschritte		
Rename	geplant	
Wrap	geplant	
Insert	in Arbeit	
Set-Attributes	geplant	

Referenz

Überblick

Beschreibung ToDo

Voraussetzungen

Folgende Software muss an einem frei wählbaren Ort verfügbar (installiert) sein:

Tabelle 3. Notwendige Packages

Package	**Nutzung**
Java (Version 5 or above)	Since Version 1.4 Java contains a default transformation processor (XSLTC). This processor will be used if no other processors are visible (in the classpath). yax assumes the presence of this processor. So at least Java 1.4 is necessary for using yax. As yas is developed under Java 5 it is the recommended Java version.
Ant(Version 1.7.0 or above)	yax comes with an ant task, so you can use yax also under ant. An ant installation is only needed if you want to use the yax ant task.
Log4j(Version 1.2.14 or above)	log4j is the recommended logging system. If log4j is not installed (resp. not reachable) yax uses its own ConsoleLogger. If you yax only under ant from yax perspective a log4j installation is not necessary because all log messages will be redirected to the ant logger.
Saxon [http://www.saxonica.com/](xls1: Version 6.5.3 or above, xls2: Version 8.6. or above)	yax provides the usage of different XSL processors. The saxon processors builds the preferred test environment for the yax development. (Obviously you can use any xsl processor which is reachable vie JAXP). The preferences file still includes the Saxon processors. If you want to use it follow Saxonias instruction how to obtain and install them.

Package	Nutzung
	If no xsl processor is installed yax uses the system default transformer which comes with JRE (XSLTC).
Xalan /(XSLTC) [http://xml.apache.org/xalan-j/](Version 2.7.0 or above)	yax provides the usage of different XSL processors. Xalan comes from Apache. The preferences file still includes the Xalan and the XSLTC processor. If you want to use them follow Apaches instruction how to obtain and install them. If no xsl processor is installed yax uses the system default transformer which comes with JRE (XSLTC).
Graphviz - Graph Visualization Software [http:// www.graphviz.org/]	

Installation

Laden Sie yax http://yax.sourceforge.net (hier der Menüpunkt 'download') herunter und entpacken Sie die Datei in ein beliebiges *<yax-installation-directory>*.

Feel free to place yax-*.jar files somewhere in your system and make it/them accessable by the right classpath entry. But at least keep a copy of the main jar file within the *<yax-installation-directory>/ lib* for retrieving the implemented components. The minimal installation can be:

```
<yax-installation-directory>/lib
|
+--- lib
     |
     +--- yax-<n.m>.jar (<n.m> = Version Number)
```

Konfiguration

Überblick

Eigenschaften

Tabelle 4. Eigenschaft

Eigenschaft	Beschreibung
Allgemeine Eigenschaften	
baseURI	The baseURI represents the base directory which is used by a relative path to become an absolute path.

Property Dateien

Voreinstellungen

Logging

classpath

For using yax in commandline mode 'yax-n.m.jar' (n.m = version number) must be reachable. if you want to run yax under ant the libraries 'yax-n.m.jar' and 'ant-yax.jar' must be reachable. Both of the libraries resides by default in YAX_HOME/lib but can be placed at any other location.

If you want to use any other xslt processor than the system default transformer you have to place their libraries into the classpath. In commandline mode you can either place the libraries into the system classpath or pass it via classpath option (-classpath resp. -cp) during the program start. When running yax under ant all necessary libraries must be assigned to the classpath BEFORE starting ant. Passing the transformer libraries via ant's classpath features (either within taskdef or with classpathref attribute) do not work.

XML Katalog

An XML catalog maps (usually remote) URIs to other (usually local) URIs.

yax uses Norman Walsh's resolver [http://xml.apache.org/commons/components/resolver/]. You can configure the resolver - especially the link to your catalog file - by editing the config/ CatalogManager.properties file. How to configure the the resolver is explained in an excellent manner on the resolver's site [http://xml.apache.org/commons/components/resolver/]

For your confinience the yax distribution contains the recent resolver library (lib/resolver.jar). If you want to use the xml catalog feature the config directory and the resolver.jar have to be part of your the classpath. Start yax with the '-noCatalog' option If you want to suppress the xml catalog usage.

baseURI

Ausdruckssprache

```
# java -classpath lib/yax-0.7.jar
       net.sf.yax.Yax
       -Dexample.dir=examples
       -Dtest.dir=test
       -DoutputFilename=output1.xml
       example1.xproc
```

```
<xproc:pipeline name="pipe1"
    xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
    xmlns:yax="http://opsdesign.eu/yax/1.0">
    <xproc:step name="Load1" type="xproc:Load" href="${examples.dir}/${inputFilename}"/>
    <xproc:step name="Store1" type="xproc:Store" href="${test.dir}/${outputFilename}"/>
</xproc:pipeline>
```

Startparameter

Tabelle 5. Leistungsumfang

Attribute	Beschreibung	Muss-Feld
script	Name des XProc Skriptes, entweder relativ zur baseURI oder absolut angegeben.	Ja
in	Spezifiziert ein einzelnes XML Dokument, das bearbeitet werden soll. Dieser Parameter ist für die gemeinsamen Nutzung mit dem 'out' Parameter vorgesehen. für die Bearbeitung von mehreren Eingabedateien bitte ein fileset benutzen.	Nein
out	Spezifiziert den Ausgabenamen für das bearbeitete Dokument.	Nein. Nicht verwenden, wenn 'in' nicht verwendet ist.

Attribute	Beschreibung	Muss-Feld
force	recreates output files, even if they are newer than their corresponding input files or the XProc script.	Nein
outDir	use in case of multiple input (via fileset). specifies the directory where the output files are to write to.	Nein.
outPattern	use in case of multiple input (via fileset). specifies how to build the output filename.	Nein.
noOutputfile	suppresses generation of an outputfile (overrides all other output parameters). The usage of this parameter makes sense when you only want to use inner output.	Nein.
noCatalog	suppresses the usage of an xml catalog.	Nein.
propertyfile	the properties this file contains will be passed to yax for further usage.	Nein.
baseURI	the base directory for all containing steps and containers.	Nein.
verbose	yax logs verbosely. For getting these log messages it is required that ant is also started in verbose mode.	Nein.
quiet	yax only logs errors, warnings and very important informations.	Nein.

Bedienung

Die Wege Input zu verwenden und Output zu erzeugen

Überblick

Input . There a (currently) two ways to access input data:

- use start parameters to pass input location(s)
- use the *<p:Load>* step.

Output .

Äußere Ein-/Ausgabe

All I/O effects by start parameters (or other interface input) is called outer I/O.

Nur den Startparameter für die Eingabe-Datei Lokation übergeben. . The most simple variant of an outer I/O is to use an input location parameter beside the mandatory xproc script location parameter:

```
java -classpath lib/yax-0.8.jar net.sf.yax.Yax examples/example1.xproc examples/example1.xml
```

yax completes the necessary information based on the input location. Assuming you pass the input file

```
path/to/input/resource/inputname.xml
```

derived from that input location yax generates the output location:

```
path/to/input/resource/inputname.output.xml
```

Startparameter für die Eingabe- und die Ausgabe-Datei Lokation übergeben. . The straight forward way of outer I/O is to pass an input location parameter and a corresponding output location parameter:

```
java -classpath lib/yax-0.8.jar net.sf.yax.Yax examples/example1.xproc
input/inputname.xml
output/outputname.xml
```

In that case the processing is obvious. yax takes the input document, passes it through the pipeline and writes the result to the ouput document.

Das XProc Skript auf mehrere Eingabe-Dokumente anwenden. . yax can also process multiple input documents in one run. For applying this feature it is recommended to use ant because in ant you can use the full flexibility of the fileset feature. Example:

```
...
  <target
    name="pipeline1"
    description="uses different multiple input files"
    >
    <yax
      verbose="yes"
      script="test/pipeline00.xproc"
      outDir="test"
      outPattern="${inputName}.out.${inputExtension}"
      >

      <fileset
        dir="examples"
        includes="
        example*.xml
        "
        excludes="
        example2.xml
        "
      />

    </yax>
  </target>
...
```

Alltough you can you this feature in commmandline mode. You pass a colon separated input file list with the *-inList* parameter:

```
java -classpath lib/yax-0.8.jar net.sf.yax.Yax examples/example1.xproc
-inList={input/inputname1.xml;input/inputname1.xml}
outDir=test outPattern="${inputName}.out.${inputExtension}
```

In case of processing multiple input documents it is not possible (or better makes no sense) to pass an output location for each input document. Therefore yax provides to other parameter for determining the output location(s).

With *-outDir* you can choose an arbitrary output directory>. If you this parameter as the only output parameter the processed documents will be written to the output directory with its input filename.

The other parameter is the *-outPattern* parameter. With *-outPattern* you describes the pattern for building the output file name. Example:

Assuming you are using the pattern:

```
outPattern="${inputName}.out.${inputExtension}
```

and you are processing the input files example1.xml and example3.xml by using ant's fileset

```
...
<fileset
  dir="examples"
  includes="
  example*.xml
  "
  excludes="
  example2.xml
  "
/>
...
```

yax will created the two file:

```
...
 examples
 |
 +-- example1.out.xml
     example2.out.xml
```

If you combine this parameter with the *-outDir* parameter determining the 'test' directory as output directory:

```
...

outDir="test"
outPattern="${inputName}.out.${inputExtension}"

...
```

yax writes the two files to the 'test' directory:

```
...

 test

 |
 +-- example1.out.xml
     example2.out.xml
```

Innere Ein-/Ausgabe

Beside the outer I/O which is effected by the usage of start parameters you can use the steps *p:Load* and *p:Store* to read resp. write xml documents during the pipeline process.

Although they are alternative instruments outer I/O and inner I/O they work simultaniously. If you process for example a pipeline which includes inner I/O without any output start parameter:

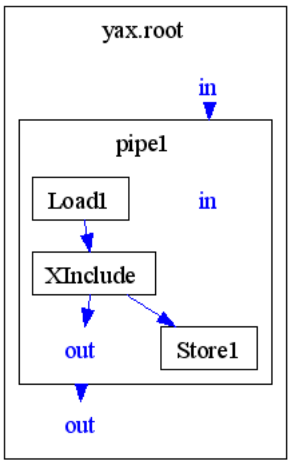
```
...
<target
  name="runExample5.1"
  description="uses inner and outer I/O"
  >
  <yax
    script="examples/example5.xproc"
    >
    ...
  </yax>
</target>
...
```

You will get two times the same output in different files:

```
...
INFO [Load:Load1] Reading input file '...\examples\xincludeArticle.xml' into Context '2'.
...
INFO [Store:Store1] Writing output file '...\test\example5.output.cmd.xml' from Context '2'.
...
INFO [root:yax.root] Writing output file '...\yaxOutput.xml' from Context '2'.
...
```

This is triggered by a)*p:Store* step and b) the automatic ouput determination of the *root* element. The following graphic shows this circumstance:

Abbildung 8. Äußere und innere Ein-/Ausgabe



The *p:Store* step creates a side branch but yax ports the content through the hole pipeline to the *root* container which is manages the outer I/O.

To avoid such undesired output behaviour (assuming in this case the outer output is undesired) you can use the -*noOutputfile* parameter:

```
...
<target
  name="runExample5.2"
  description="uses only inner I/O"
  >
  <yax
    noOutputfile="yes"
    script="examples/example5.xproc"
    >
    ...
  </yax>
</target>
...
```

In this case the content on the output port of *root* will be discarded.

Start über die Kommandozeile

```
# java -classpath lib/yax-0.7.jar net.sf.yax.Yax [options]pipeline file location [XML input data file loca
```

```
element 'yax:transformers' has ...something about transformer preference
...
step 'step name': something about transformer which will be used
...
messages from each step
...
Yax run sucessful completed.
```

```
Usage: java yax [options] pipelineURI [inputfileURI [outputfileURI]]

Options:

-h, -help              print this message (and exit)
-version               print version information (and exit)
-quiet, -q             be extra quiet
-verbose, -v           be extra verbose (quiet takes precedence)
-debug, -d             print debugging information
-baseURI=<value>       set the baseURI to value
-noOutputfile          suppress generation of an outputfile (overrides all other output parameters)
                       (make sense if output is created within the pipeline)
-D<property>=<value>   provide property for using within the xproc script as '${property}'
-propertyfile <name>   load properties from file (-D<property>s take precedence)

(for further usage alternatives see http://yax.sourceforge.net/)
```

Warnung

debug effects that some steps (especially XSLT) logs the hole content of a context. So use thit option with care. At best use only small inputs when you use the debug option.

- 1) <*input filename*>.output.xml
- 2) <*pipeline filename*>.config.xml
- 3) <*pipeline filename*>.pipeTrace.png
- 4) <*pipeline filename*>.portTrace.png

Ant Task

Überblick

As the XSLT task do not provide multiple input (and output) yax provides an own solution using ant's well known fileset feature in collaboration with the *outDir* and/or *outPattern* attributes.

Corresponding to the XSLT task yax also provides passing parameters to the pipeline processor and the used subsystems (as transformers). While these parameters are not exclusively passed for the usage in transformers to yax the different name 'property' was used. But these 'properties' will also be passed as parameters to the transformers.

If you intend to use other xslt processors than the system defualt processor please consider that all necessary libraries must be assigned to the classpath BEFORE starting ant. Passing the transformer libraries via ant's classpath features (either within taskdef or with classpathref attribute) do not work.

```
...
<target
  name="runExample1"
  description="transforms an input file with concatenated transformation steps"
  >
  <yax
    in="examples/example1.xml"
    out="test/example1.output.ant.xml"
    script="examples/example1.xproc"
    >
  </yax>
</target>
...
```

Beschreibung

Parameter

Tabelle 6. Leistungsumfang

Attribute	Beschreibung	Muss-Feld
script	Name des XProc Skriptes, entweder relativ zur baseURI oder absolut angegeben.	Ja
in	Spezifiziert ein einzelnes XML Dokument, das bearbeitet werden soll. Dieser Parameter ist für die gemeinsamen Nutzung mit dem 'out' Parameter vorgesehen. für die Bearbeitung von mehreren Eingabedateien bitte ein fileset benutzen.	Nein
out	Spezifiziert den Ausgabenamen für das bearbeitete Dokument.	Nein. Nicht verwenden, wenn 'in' nicht verwendet ist.
force	recreates output files, even if they are newer than their corresponding input files or the XProc script.	Nein
outDir	use in case of multiple input (via fileset). specifies the directory where the output files are to write to.	Nein.
outPattern	use in case of multiple input (via fileset). specifies how to build the output filename.	Nein.
noOutputfile	suppresses generation of an outputfile (overrides all other output parameters).	Nein.

Attribute	**Beschreibung**	**Muss-Feld**
	The usage of this parameter makes sense when output is created within the pipeline.	
propertyfile	the properties this file contains will be passed to yax for further usage.	Nein.
baseURI	the base directory for all containing steps and containers.	Nein.
verbose	yax logs verbosely. For getting these log messages it is required that ant is also started in verbose mode.	Nein.
quiet	yax only logs errors, warnings and very important informations.	Nein.

Eingebettete Elemente als Parameter

Beispiele

see section 'Examples'

SOAP Schnittstelle

Die Pipeline-Konfigurationsdatei

```
<yax:configuration xmlns:yax="http://www.opsdesign.eu/yax/1.0">
<!--configuration description generated by Yax - Do not edit by hand-->
<pipeline name="pipe1">
..
<step name="trans1">
<output port="out" sequence="no" yax:creator="implementation.xproc.standard"/>
<input port="in" sequence="no" yax:creator="implementation.xproc.standard">
<yax:connection port="in" yax:component="pipe1"/>
</input>
<input port="stylesheet" sequence="no" yax:creator="implementation.xproc.standard"/>
<input href="examples/transformation1.xsl" port="stylesheet" yax:creator="pipeline"/>
</step>
...
</pipeline>
<xproc:pipeline-library name="xproc.options" xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0">
<xproc:declare-step-type type="Rename" yax:description="">
<xproc:output port="out" select="" sequence="no" yax:bySourceRequired="no"/>
<xproc:input port="in" select="" sequence="no" yax:bySourceRequired="no"/>
<xproc:parameter name="name" required="yes" yax:values="{$any}"/>
<xproc:parameter name="select" required="yes" yax:values="{$xpathExpression}"/>
</xproc:declare-step-type>
...
</xproc:pipeline-library>
<xproc:pipeline-library name="xproc.standard" xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0">
...
</xproc:pipeline-library>
<xproc:pipeline-library name="yax.standard" xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0">
...
</xproc:pipeline-library>
</yax:configuration>
```

Grafische Visualisierung einer Pipeline

Abbildung 9. Komprimierte grafische Darstellung der Pipeline 'example1.xproc'.

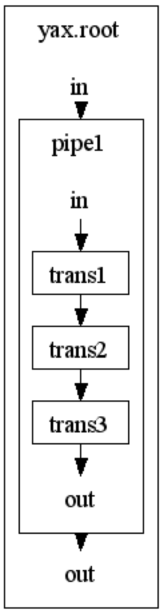


Abbildung 10. Pipeline mit impliziter und expliziten Verbindungen

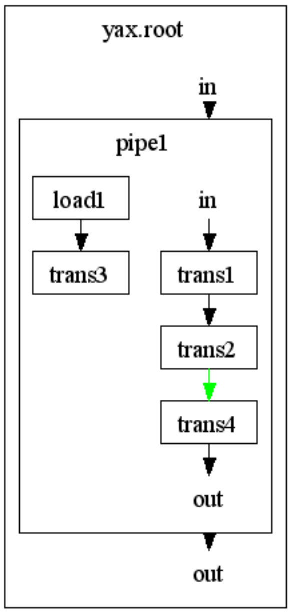
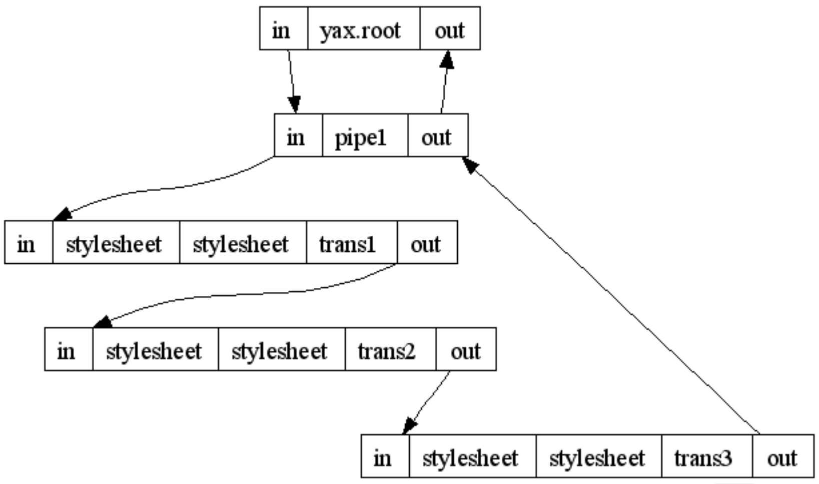


Abbildung 11. Komprimierte Grafische Darstellung einer Pipeline.



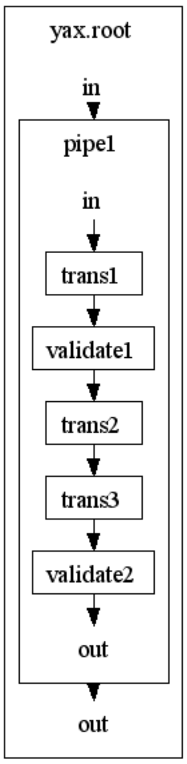
Entwerfen und Aufsetzen einer Pipeline

Eine Pipeline verarbeiten

Das Konstrukt <*p:pipeline*> einsetzen

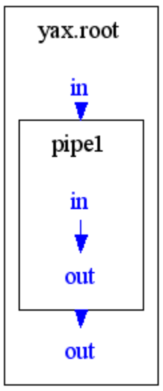
```
<xproc:pipeline name="pipe1"
   xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
   xmlns:yax="http://opsdesign.eu/yax/1.0">
   <xproc:step name="trans1" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation1.xsl"/>
   </xproc:step>
   <xproc:step name="validate1" type="xproc:Validate"/>
   <xproc:step name="trans2" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
   </xproc:step>
   <xproc:step name="trans3" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation3.xsl"/>
   </xproc:step>
   <xproc:step name="validate2" type="xproc:Validate"/>
</xproc:pipeline>
```

Abbildung 12. Pipeline



```
<xproc:pipeline name="pipe1"
    xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
    xmlns:yax="http://opsdesign.eu/yax/1.0">
</xproc:pipeline>
```

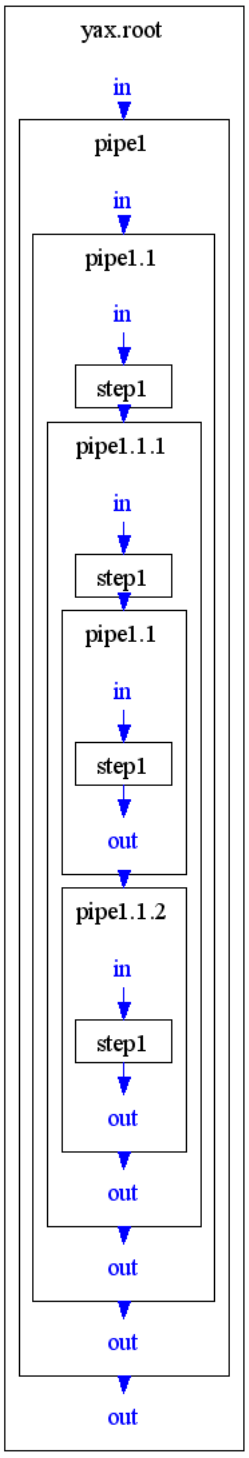
Abbildung 13. Choose/When/Otherwise Konstrukt mit expiziten Ports am *choose* Element.



```
...
Construct 'pipe1' is empty and will be bridged.
...
```

```
<xproc:pipeline name="pipe1"
    xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
    xmlns:yax="http://opsdesign.eu/yax/1.0">
    <xproc:pipeline name="pipe1.1">
        <xproc:step name="step1" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="examples/transformation1.xsl"/>
        </xproc:step>
        <xproc:pipeline name="pipe1.1.1">
            <xproc:step name="step1" type="xproc:XSLT">
                <xproc:input port="stylesheet" href="examples/transformation2.xsl"/>
            </xproc:step>
            <xproc:pipeline name="pipe1.1.1.1">
                <xproc:step name="step1" type="xproc:XSLT">
                    <xproc:input port="stylesheet" href="examples/transformation3.xsl"/>
                </xproc:step>
            </xproc:pipeline>
        </xproc:pipeline>
    </xproc:pipeline>
</xproc:pipeline>
```

Abbildung 14. Pipeline mit verschachtelten Komponenten



Den Step *<p:XSLT>* einsetzen

```
<xproc:step name="trans1" type="xproc:XSLT">
   <xproc:input port="stylesheet" href="test/transformation1.xsl"/>
</xproc:step>
```

```
<xproc:step name="trans1" type="xproc:XSLT">
  <xproc:input port="stylesheet" href="test/transformation1.xsl"/>
               <xproc:parameter name="transformer" value="Saxon8"/>
</xproc:step>
```

Tabelle 7. Exceptions des Steps <*XSLT*>

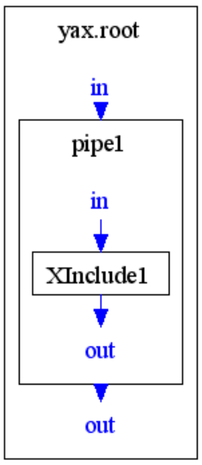
Exception	**Bemerkung**
aufgetreten während der Abarbeitung	
EmptyResultException	
FileNotFoundException	If href is used to reach file and this file can not be found.
TransformerConfiguration	
Transformer	
DOM	
aufgetreten während der Bestimmung des Transformers	
TransformerConfiguration	
Transformer	
DOM	

Den gewünschten Transformer festlegen

Den Step <*p:XInclude*> einsetzen

```
...
<xproc:step name="XInclude1" type="xproc:XInclude"/>
...
```

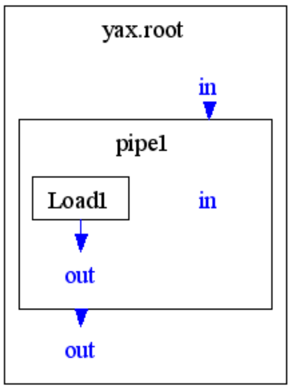
Abbildung 15. <*p:XInclude*>



Den Step <*p:Load*> einsetzen

```
...
<xproc:step name="Load1" type="xproc:Load" href="examples/example1.xml"/>
...
```

Abbildung 16. <*p:Load*>



```
# java -classpath lib/yax-0.7.jar net.sf.yax.Yax [options]pipeline file location
```

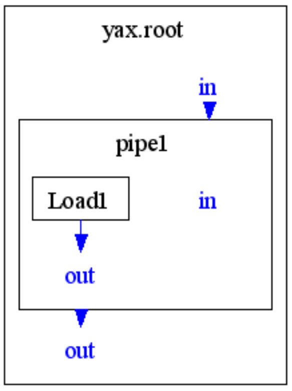
```
# java -classpath lib/yax-0.7.jar
       net.sf.yax.Yax
       -Dexample.dir=examples
       -Dtest.dir=test
       -DoutputFilename=output1.xml
       example1.xproc
```

```
<xproc:pipeline name="pipe1"
    xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
    xmlns:yax="http://opsdesign.eu/yax/1.0">
    <xproc:step name="Load1" type="xproc:Load" href="${examples.dir}/${inputFilename}"/>
    <xproc:step name="Store1" type="xproc:Store" href="${test.dir}/${outputFilename}"/>
</xproc:pipeline>
```

Den Step <*p:Store*> einsetzen

```
...
<xproc:step name="Store1" type="xproc:Store" href="examples/output.xml"/>
...
```

Abbildung 17. <*p:Store*>



```
# java -classpath lib/yax-0.7.jar net.sf.yax.Yax [options]pipeline file location
```

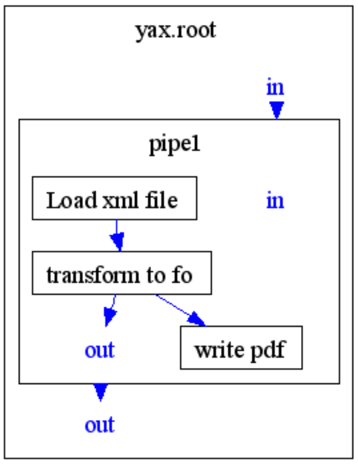
```
# java -classpath lib/yax-0.7.jar
       net.sf.yax.Yax
       -Dexample.dir=examples
       -Dtest.dir=test
       -DoutputFilename=output1.xml
       example1.xproc
```

```
<xproc:pipeline name="pipe1"
    xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
    xmlns:yax="http://opsdesign.eu/yax/1.0">
    <xproc:step name="Load1" type="xproc:Load" href="${examples.dir}/${inputFilename}"/>
    <xproc:step name="Store1" type="xproc:Store" href="${test.dir}/${outputFilename}"/>
</xproc:pipeline>
```

Den Step <*p:Fop*> zum Erzeugen von PDF-Dateien einsetzen

```
...
<xproc:step name="write pdf" type="xproc:Fop"/>
...
```

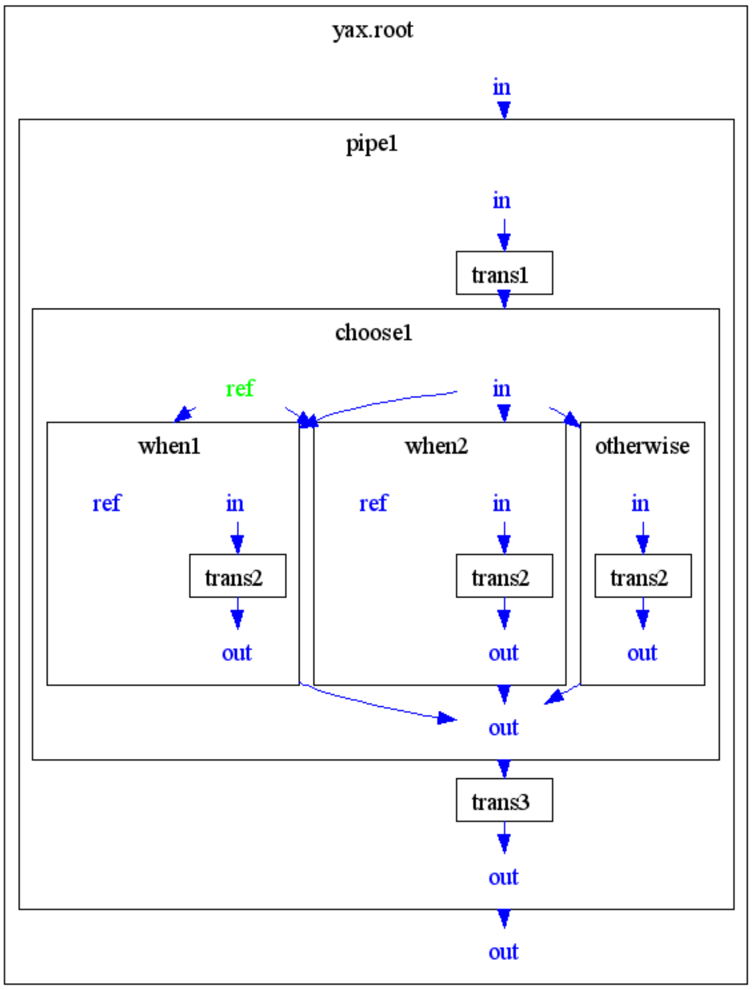
Abbildung 18. <*p:XInclude*>



Das Konstrukt *<p:choose>*/*<p:when>*/*<p:otherwise>* einsetzen

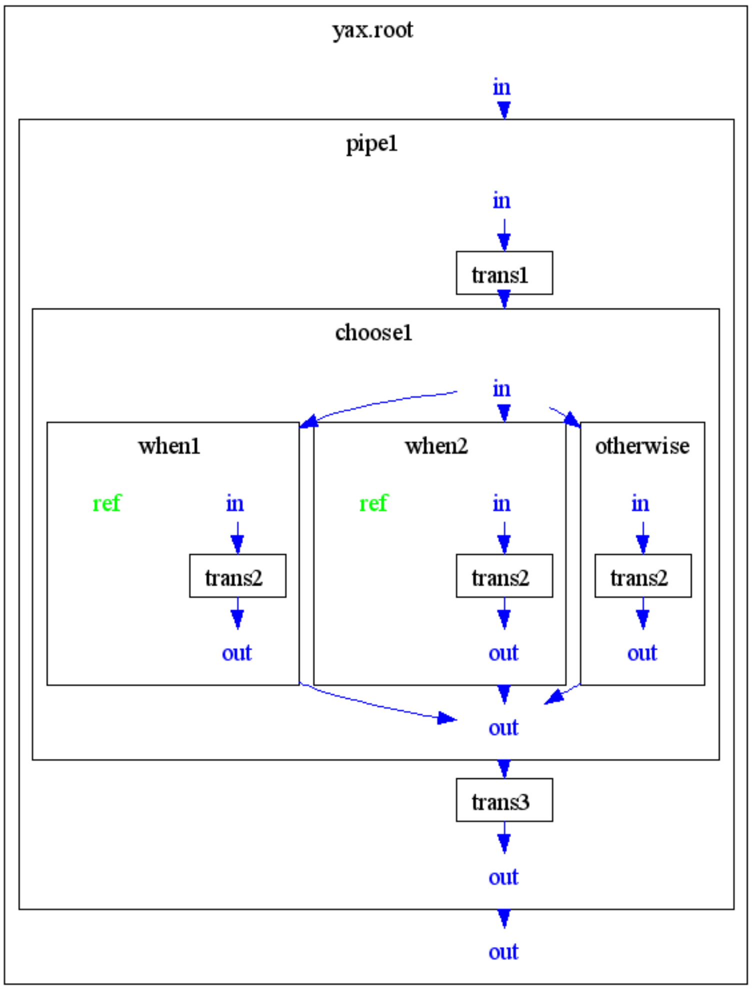
```
<xproc:pipeline name="pipe1"
   xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
   xmlns:yax="http://opsdesign.eu/yax/1.0">
   <xproc:step name="trans1" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation1.xsl"/>
   </xproc:step>
   <xproc:choose name="choose1" type="xproc:XSLT">
      <xproc:input port="ref" href="test/chooseInput0.xml"/>
      <xproc:when name="when1" test="/test1">
         <xproc:step name="trans2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
         </xproc:step>
      </xproc:when>
      <xproc:when name="when2" test="/test2">
         <xproc:step name="trans2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
         </xproc:step>
      </xproc:when>
      <xproc:otherwise name="otherwise">
         <xproc:step name="trans2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
         </xproc:step>
      </xproc:otherwise>
   </xproc:choose>
   <xproc:step name="trans3" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation3.xsl"/>
   </xproc:step>
</xproc:pipeline>
```

Abbildung 19. Choose/When/Otherwise Konstrukt mit expiziten Ports am *choose* Element.



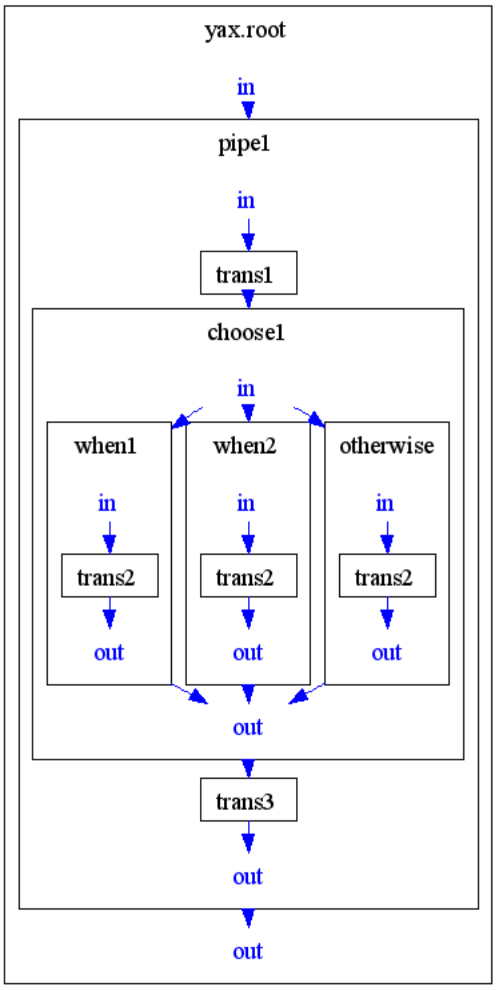
```
<xproc:pipeline name="pipe1"
   xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
   xmlns:yax="http://opsdesign.eu/yax/1.0">
   <xproc:step name="trans1" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation1.xsl"/>
   </xproc:step>
   <xproc:choose name="choose1" type="xproc:XSLT">
      <xproc:when name="when1" test="/test1">
         <xproc:input port="ref" href="test/chooseInput1.xml"/>
         <xproc:step name="trans2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
         </xproc:step>
      </xproc:when>
      <xproc:when name="when2" test="/test2">
         <xproc:input port="ref" href="test/chooseInput2.xml"/>
         <xproc:step name="trans2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
         </xproc:step>
      </xproc:when>
      <xproc:otherwise name="otherwise">
         <xproc:step name="trans2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
         </xproc:step>
      </xproc:otherwise>
   </xproc:choose>
   <xproc:step name="trans3" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation3.xsl"/>
   </xproc:step>
</xproc:pipeline>
```

Abbildung 20. Choose/When/Otherwise Konstrukt mit expiziten Ports an allen *when* Elementen.



```
<xproc:pipeline name="pipe1"
   xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
   xmlns:yax="http://opsdesign.eu/yax/1.0">
   <xproc:step name="trans1" type="xproc:XSLT">
   </xproc:step>
   <xproc:choose name="choose1" type="xproc:XSLT">
      <xproc:input port="ref" href="test/chooseInput0.xml"/>
      <xproc:when name="when1" test="/test1">
         <xproc:step name="trans2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
         </xproc:step>
      </xproc:when>
      <xproc:when name="when2" test="/test2">
         <xproc:step name="trans2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
         </xproc:step>
      </xproc:when>
      <xproc:otherwise name="otherwise">
         <xproc:step name="trans2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
         </xproc:step>
      </xproc:otherwise>
   </xproc:choose>
   <xproc:step name="trans3" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation3.xsl"/>
   </xproc:step>
</xproc:pipeline>
```

Abbildung 21. Choose/When/Otherwise Konstrukt mit expiziten Ports an allen Elementen.

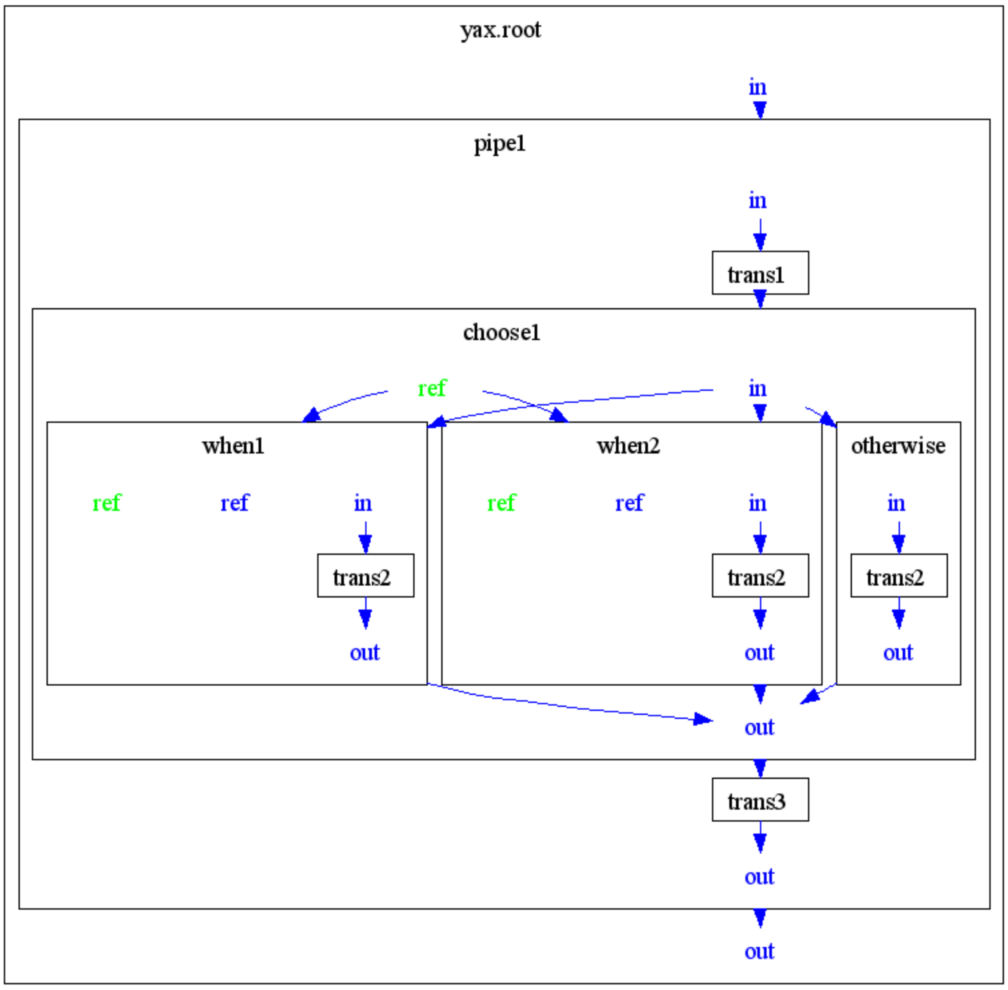


```
construct 'when1': reference port not found.
```

```
<xproc:pipeline name="pipe1"
   xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
   xmlns:yax="http://opsdesign.eu/yax/1.0">
   <xproc:step name="trans1" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation1.xsl"/>
   </xproc:step>
   <xproc:choose name="choose1" type="xproc:XSLT">
```

```
      <xproc:input port="ref" href="test/chooseInput0.xml"/>
      <xproc:when name="when1" test="/test1">
         <xproc:input port="ref" href="test/chooseInput1.xml"/>
         <xproc:step name="trans2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
         </xproc:step>
      </xproc:when>
      <xproc:when name="when2" test="/test2">
         <xproc:input port="ref" href="test/chooseInput2.xml"/>
         <xproc:step name="trans2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
         </xproc:step>
      </xproc:when>
      <xproc:otherwise name="otherwise">
         <xproc:step name="trans2" type="xproc:XSLT">
            <xproc:input port="stylesheet" href="test/transformation2.xsl"/>
         </xproc:step>
      </xproc:otherwise>
   </xproc:choose>
   <xproc:step name="trans3" type="xproc:XSLT">
      <xproc:input port="stylesheet" href="test/transformation3.xsl"/>
   </xproc:step>
</xproc:pipeline>
```

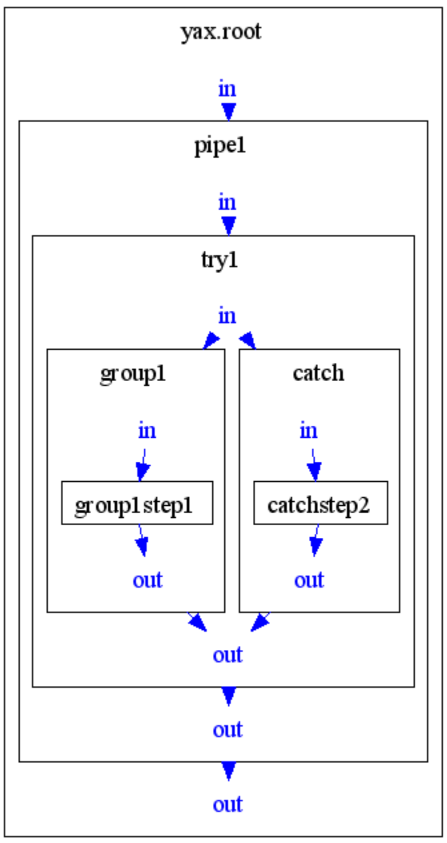
Abbildung 22. Choose/When/Otherwise Konstrukt mit expiziten Ports an allen Elementen.



Das Konstrukt *<p:try>/<p:catch>* einsetzen

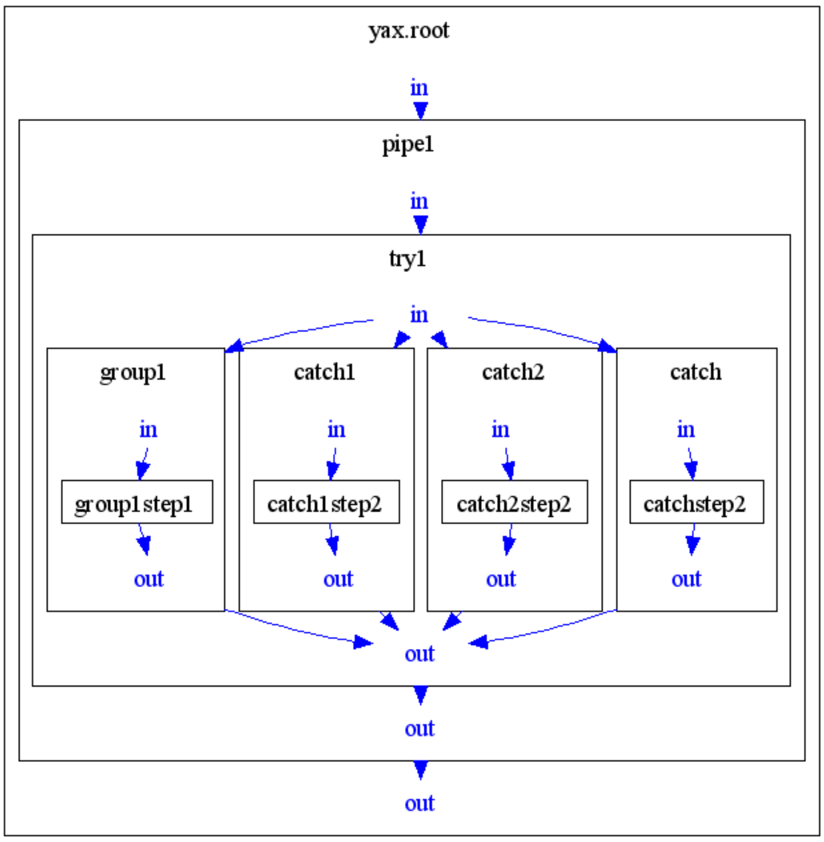
```
<xproc:pipeline name="pipe1"
    xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
    xmlns:yax="http://opsdesign.eu/yax/1.0">
    <xproc:try name="try1">
        <xproc:group name="group1">
            <xproc:step name="group1step1" type="xproc:XSLT">
                <xproc:input port="stylesheet" href="examples/transformation2.xsl"/>
            </xproc:step>
        </xproc:group>
        <xproc:catch name="catch">
            <xproc:step name="catchstep2" type="xproc:XSLT">
                <xproc:input port="stylesheet" href="examples/transformation1.xsl"/>
            </xproc:step>
        </xproc:catch>
    </xproc:try>
</xproc:pipeline>
```

Abbildung 23. Das Try/Catch construct mit einer einizen catch Klausel



```
<xproc:pipeline name="pipe1"
    xmlns:xproc="http://www.w3.org/TR/2006/xproc/1.0"
    xmlns:yax="http://opsdesign.eu/yax/1.0">
    <xproc:try name="try1">
        <xproc:group name="group1">
            <xproc:step name="group1step1" type="xproc:XSLT">
                <xproc:input port="stylesheet" href="examples/transformation2.xsl"/>
            </xproc:step>
        </xproc:group>
        <xproc:catch name="catch1" exception="EmptyResult">
            <xproc:step name="catch1step2" type="xproc:XSLT">
                <xproc:input port="stylesheet" href="examples/transformation1.xsl"/>
            </xproc:step>
        </xproc:catch>
        <xproc:catch name="catch2" exception="FileNotFound">
            <xproc:step name="catch2step2" type="xproc:XSLT">
                <xproc:input port="stylesheet" href="examples/transformation1.xsl"/>
            </xproc:step>
        </xproc:catch>
        <xproc:catch name="catch">
            <xproc:step name="catchstep2" type="xproc:XSLT">
                <xproc:input port="stylesheet" href="examples/transformation1.xsl"/>
            </xproc:step>
        </xproc:catch>
    </xproc:try>
</xproc:pipeline>
```

Abbildung 24. Das Try/Catch construct mit einer einizen catch Klausel



Roadmap

Beschreibung ToDo

Beispiele

Überblick

ToDo

Beispiel 1.1 - verkettete Transformationen

ToDo

Beispiel 1.2 - verkettete Transformationen (mit der Benutzung eines xml Katalogs)

ToDo

Beispiel 2.1. - *<p:XInclude>* in Aktion

This is an example with real data coming from the docbook.sml project [http://docbooksml/sourceforge.net/]. It collects all sections distributed over several files into the output file. Start yax with:

```
# java -classpath lib/yax-0.7.jar
       net.sf.yax.Yax
       -Dparam1=passedFromProgramStart
       example7.xproc
       examples/xincludeArticle.xml
       test/output1.xml
```

Beispiel 2.1. - *<p:XInclude>* in Aktion

Beispiel 3.1 - *<p:try>/<p:group>/<p:catch>* in Aktion

ToDo

Beispiel 3.2 - *<p:try>/<p:group>/<p:catch>* in Aktion

ToDo

Beispiel 4.1 - *<p:Load>/<p:XInclude>/<p:Store>* in Aktion

ToDo

Beispiel 4.2 - *<p:Load>/<p:XInclude>/<p:Store>* in Aktion

ToDo

Beispiel 5.1 - Using Inner and Outer I/O together

ToDo

Beispiel 5.2 - Using only Inner I/O

ToDo

Beispiel 6 - Übergabe von (substituierten) Transformationsparameters an *<p:XSLT>*

This examples shows how to

- pass a parameter from the program start through the xproc script to the xsl script (param1)
- pass a parameter from xproc script to the xsl script (param2)
- use parameters default value because nothing passed to (param3)

Start yax with:

```
# java -classpath lib/yax-0.7.jar
       net.sf.yax.Yax
       -Dparam1=passedFromProgramStart
       example6.xproc
```

```
        examples/example1.xml
        test/output1.xml
```

The output file test/output1.xml will consist of:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
    <param1>passedFromProgramStart</param1>
    <param2>${param2}</param2>
    <param3>notPassedToScript</param3>
</result>
```

Consider that param1 comes from program start, param2 comes (staticly) from xproc script and param3 is the default value from the xsl script.

Beispiel 7 - processing multiple input documents

ToDo

Beispiel 8 - *<p:PDF>*/*<p:FOP>* in Aktion

ToDo

Tutorial

Eine einfache Pipeline

ToDo

Eine Pipeline mit zusätzlicher Ausgabe

ToDo

Eine Pipeline mit zusätzlicher Datenquelle

ToDo

Eine Pipeline mit Choose/When/Otherwise und Try/Catch

ToDo

Links

Beschreibung ToDo

Rechtlicher Rahmen

Powered by

[http://sourceforge.net]

[http://docbook.sourceforge.net/]

[http://xmlgraphics.apache.org/fop//]

[http://www.saxonica.com/index.html]