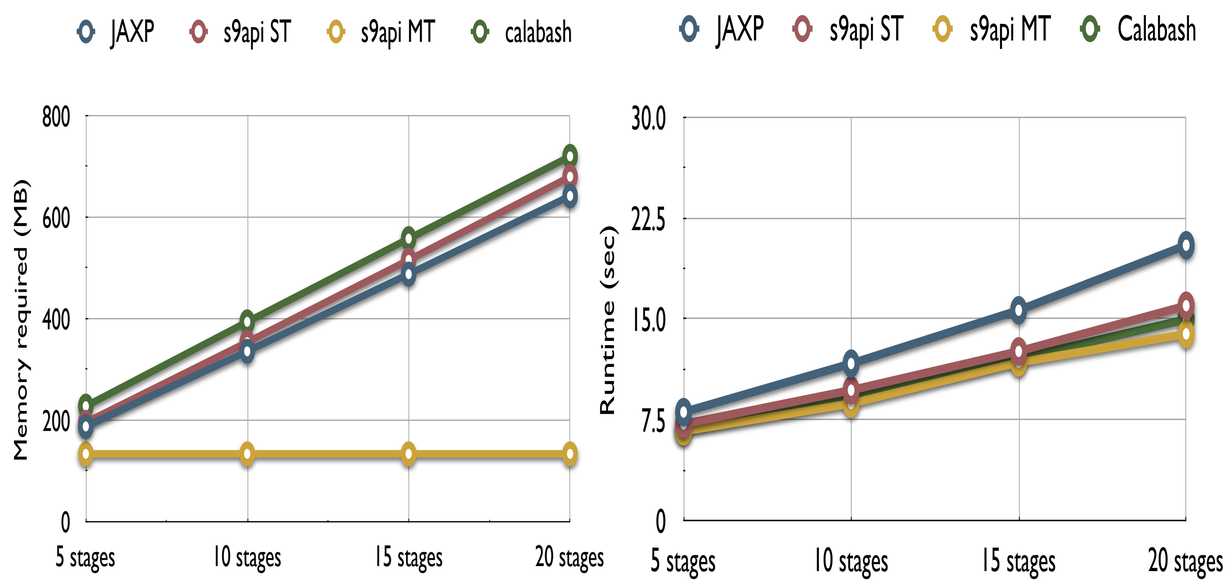
XML Pipeline Performance

Nigel Whitaker, Tristan Mitchell, DeltaXML Ltd.

DeltaXML.com
Change control for XML

Simple Benchmarks



Pipelines:
JAXP Pipeline uses features present in JDK 1.4 and later. The XMLFilter and TransformerHandler classes from a TransformerFactory allow XSLT processing pipelines to be constructed.
s9api 'Single Trigger' uses a simple way of chaining pipelines in the s9api package provided with Saxon v9.0 and subsequent releases.
s9api 'Multi Trigger' is designed for easy garbage collection of intermediate trees.
The Calabash pipeline uses XProc and uses the same filters as the other pipelines.

Details:

Saxon Version:	HE 9.2.0.6j
Calabash version:	0.9.19
Java version:	1.6.0_17 (64 bit server VM)
Hardware:	MacBook Pro, 2.53 GHz Core 2 Duo
OS:	MacOS 10.6.2

Observations:

JAXP appears slower than the other s9api based approaches using Saxon's XdmNode/TinyTree data structures: shared namepool, simpler/faster integer rather than char[]/String events.
Care is needed to ensure garbage collection of previous intermediate trees after the pipeline stage finishes and dependent stages have used the data.
These results were repeated to check the latest releases of Saxon and Calabash - we discovered that memory consumption had decreased since the first experiments using Saxon 9.1 in May 2009.

Data:

The data passed through the pipeline is the XSLT2.0 specification. This is an xhtml file. However, at 1.6MB it is fairly small, so the contents of the s inside the <body> were duplicated 10 times using a text editor to create a 16MB input file.

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xhtml="http://www.w3.org/1999/xhtml" version="1.0">

  <xsl:output method="xml"/>


  <!-- identity template -->
  <xsl:template match="@*!node()">
    <xsl:copy>
      <xsl:apply-templates select="@*!node()"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="xhtml:body">
    <xsl:copy>
      <xsl:apply-templates select="@*!node()"/>
      <xhtml:p>This is a new paragraph inserted by the stage1 filter</xhtml:p>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

```
/** JAXP 5 stages */
public static void main(String[] args)
throws TransformerException, TransformerConfigurationException
{
  TransformerFactory tf= TransformerFactory.newInstance();
  if (!tf.getFeature(SAXSource.FEATURE) ||
      !tf.getFeature(SAXResult.FEATURE) ||
      !tf.getFeature(SAXTransformerFactory.FEATURE) ||
      !tf.getFeature(SAXTransformerFactory.FEATURE_XMLFILTER))
    throw new Error("SAXTransformerFactory not supported");
  SAXTransformerFactory stf= (SAXTransformerFactory) tf;

  XMLFilter stage1= stf.newXMLFilter(new StreamSource("stage1.xsl"));
  XMLFilter stage2= stf.newXMLFilter(new StreamSource("stage2.xsl"));
  Transformer stage3= stf.newTransformer(new StreamSource("stage3.xsl"));
  TransformerHandler stage4=
      stf.newTransformerHandler(new StreamSource("stage4.xsl"));
  TransformerHandler stage5=
      stf.newTransformerHandler(new StreamSource("stage5.xsl"));
  stage2.setParent(stage1);
  stage4.setResult(new SAXResult(stage5));
  stage5.setResult(new StreamResult(args[1]));
  stage3.transform(new SAXSource(stage2, new InputSource(args[0])),
                   new SAXResult(stage4));
}
```

```
/* s9api 5 stages single-trigger */
public static void main(String[] args)
  throws SaxonApiException
{
  Processor proc= new Processor(false);
  XsltCompiler comp= proc.newXsltCompiler();
  XdmNode in= proc.newDocumentBuilder().build(new StreamSource(new File(args[0])));
  Serializer out= new Serializer();
  out.setOutputFile(new File(args[1]));
  XsltTransformer stage1=
               comp.compile(new StreamSource(new File("stage1.xsl"))).load();
  XsltTransformer stage2=
               comp.compile(new StreamSource(new File("stage2.xsl"))).load();
  XsltTransformer stage3=
               comp.compile(new StreamSource(new File("stage3.xsl"))).load();
  XsltTransformer stage4=
               comp.compile(new StreamSource(new File("stage4.xsl"))).load();
  XsltTransformer stage5=
               comp.compile(new StreamSource(new File("stage5.xsl"))).load();
  stage1.setDestination(stage2);
  stage2.setDestination(stage3);
  stage3.setDestination(stage4);
  stage4.setDestination(stage5);
  stage5.setDestination(out);
  stage1.setInitialContextNode(in);
  stage1.transform();
}
```

```
/* s9api 5 stages multi-triggered */
public static void main(String[] args)
throws SaxonApiException
{
  Processor proc = new Processor(false);
  XsltCompiler comp = proc.newXsltCompiler();
  XdmNode in= proc.newDocumentBuilder().build(new StreamSource(new File(args[0])));
  Serializer out= new Serializer();
  out.setOutputFile(new File(args[1]));
  XsltTransformer stage1=
               comp.compile(new StreamSource(new File("stage1.xsl"))).load();
  XsltTransformer stage2=
               comp.compile(new StreamSource(new File("stage2.xsl"))).load();
  XsltTransformer stage3=
               comp.compile(new StreamSource(new File("stage3.xsl"))).load();
  XsltTransformer stage4=
               comp.compile(new StreamSource(new File("stage4.xsl"))).load();
  XsltTransformer stage5=
               comp.compile(new StreamSource(new File("stage5.xsl"))).load();

  XdmDestination stage1result= new XdmDestination();
  stage1.setDestination(stage1result);
  stage1.setInitialContextNode(in);
  stage1.transform();

  in= null;
  XdmDestination stage2result= new XdmDestination();
  stage2.setDestination(stage2result);
  stage2.setInitialContextNode(stage1result.getXdmNode());
  stage2.transform();

  stage1result= null; stage1= null;
  XdmDestination stage3result= new XdmDestination();
  stage3.setDestination(stage3result);
  stage3.setInitialContextNode(stage2result.getXdmNode());
  stage3.transform();

  stage2result= null; stage2= null;
  XdmDestination stage4result= new XdmDestination();
  stage4.setDestination(stage4result);
  stage4.setInitialContextNode(stage3result.getXdmNode());
  stage4.transform();

  stage3result= null; stage3= null;
  XdmDestination stage5result= new XdmDestination();
  stage5.setDestination(out);
  stage5.setInitialContextNode(stage4result.getXdmNode());
  stage5.transform();

  stage4result= null; stage4= null;
  stage5result= null; stage5= null;
}
```

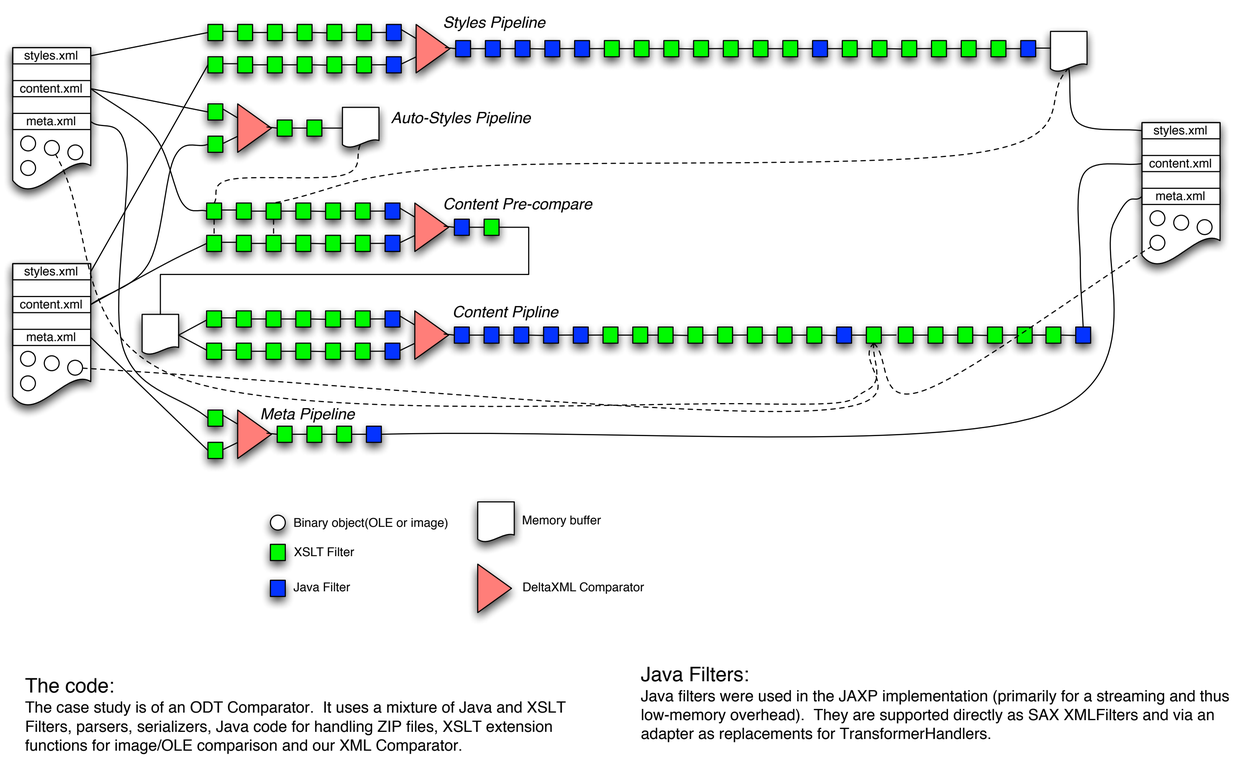
```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc"
  name="five-stages"  version="1.0"
  xmlns:dx="http://www.deltaxml.com/ns/extensions/xproc"
  xmlns:cx="http://xmlcalabash.com/ns/extensions"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:c="http://www.w3.org/ns/xproc-step">

  <p:input port="source" kind="document" sequence="false" primary="true"/>
  <p:input port="parameters" kind="parameter" primary="false"/>
  <p:output port="result" sequence="false" primary="true"/>

  <p:xslt>
    <p:input port="stylesheet">
      <p:document href="stage1.xsl"/>
    </p:input>
    <p:input port="parameters"/>
  </p:xslt>
  <p:xslt>
    <p:input port="stylesheet">
      <p:document href="stage2.xsl"/>
    </p:input>
  </p:xslt>
  <p:xslt>
    <p:input port="stylesheet">
      <p:document href="stage3.xsl"/>
    </p:input>
  </p:xslt>
  <p:xslt>
    <p:input port="stylesheet">
      <p:document href="stage4.xsl"/>
    </p:input>
  </p:xslt>
  <p:xslt>
    <p:input port="stylesheet">
      <p:document href="stage5.xsl"/>
    </p:input>
  </p:xslt>
</p:declare-step>
```

Case Study

ODT Comparator Pipeline



The code:

The case study is of an ODT Comparator. It uses a mixture of Java and XSLT Filters, parsers, serializers, Java code for handling ZIP files, XSLT extension functions for image/OLE comparison and our XML Comparator.

History:

The first implementation used JAXP pipelining as its basis. For the 700 page documents in question, we were asked why it needed 1GB of Java heap space!

The simple benchmark experiments last year confirmed our hunch that intermediate trees were not being garbage collected. The s9api experiments were successful and we started on an initial s9api based pipeline which halved memory requirements. Optimization is ongoing and through some of the techniques described here we are close to 360MB. However further optimizations are being planned - our, perhaps optimistic, target is 100MB of heap memory.

The Calabash figures are presented because we were more generally curious about performance and while we probably will not change our Java/s9api based implementation at this time, we will use XProc/Calabash for other projects.

Details:

Saxon Version:	B 9.1.0.7j
Calabash version:	0.9.15
Java version:	1.6.0_17 (64 bit server VM)
Hardware:	MacBook Pro, 2.53 GHz Core 2 Duo
OS:	MacOS 10.6.2

Results:

Pipeline	runtime (min:sec)	minimum memory (MB)
JAXP	5:14	933
s9api	4:05	457
Calabash	4:42	1631
s9api + optimizations	3:12	360

Optimization: Java filter streaming/combiming

While the overall s9api performance is better than JAXP, using XdmNodes between Java filters precludes possible event streaming. The method used to process a single filter can be extended to process a list of filters quite easily using the setParent() technique.

```
private XdmNode runJavaFilters(XMLFilterImpl[] filters, XdmNode input)
  throws SaxonApiException, XPathException
{ ... }
```

By adding a new method and applying it where adjacent filters are identified speed-ups can be achieved as indicated by the progress output. Here is the non-optimized output (from the 5 Java/blue filters in the content pipeline above):

```
finished r[0] (Id: com...filters.dx2.wbw.OrphanedWordOutfilter)
              (elapsed: 2897 ms, cpu: 2507 ms)
finished r[1] (Id: com...filters.dx2.wbw.OrphanedWordOutfilter)
              (elapsed: 2825 ms, cpu: 2468 ms)
finished r[2] (Id: com...filters.dx2.wbw.WordSpaceFixup)
              (elapsed: 3063 ms, cpu: 2634 ms)
finished r[3] (Id: com...pdf.filters.ChangeMetricsFilter)
              (elapsed: 2724 ms, cpu: 2400 ms)
finished r[4] (Id: com...filters.dx2.wbw.WordOutfilter)
              (elapsed: 2585 ms, cpu: 2257 ms)
```

The above 5 filters take 12,226 ms of CPU time; after the optimiztion time time is more than halved:

```
finished r[0]-r[4] (elapsed: 5347 ms, cpu: 4724 ms)
```

Optimization: Filter conversion

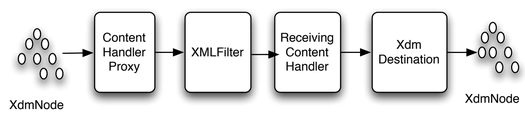
As well as streaming filters together another route to performance improvement is to consider the how filters are coded. By implementing the ProxyReceiver and related methods we avoid the cost of event conversion and associated NamePool access. We have implemented this conversion for one filter 'WordSpaceFixup', as the progress log below indicates the runtime is reduced by over half:

```
finished r[2] (Id: com...filters.dx2.wbw.WordSpaceFixup)
              (elapsed: 992 ms, cpu: 809 ms)
```

Java Filters:

Java filters were used in the JAXP implementation (primarily for a streaming and thus low-memory overhead). They are supported directly as SAX XMLFilters and via an adapter as replacements for TransformerHandlers.

The initial processing model for the s9api pipelines uses XdmNode trees as the intermediate data structures between pipeline stages. Code was developed run a SAX XMLFilter as shown below:




```
private XdmNode runJavaFilter(XMLFilterImpl f, XdmNode input)
  throws SaxonApiException, XPathException
{
  Sender s= new Sender(pc);
  ContentHandlerProxy chp= new ContentHandlerProxy();
  XMLPushFilter pf= new XMLPushFilterAdapter(f);
  chp.setUnderlyingContentHandler(pf);
  chp.setLexicalHandler(pf);
  chp.setPipelineConfiguration(pc);
  ReceivingContentHandler rch= new ReceivingContentHandler();
  rch.setPipelineConfiguration(pc);
  pc.setLocationProvider(null);
  // rch.setDocumentLocator((Locator) pf); // ToDo: investigate/test Locator
  XdmDestination result= new XdmDestination();
  rch.setReceiver(result.getReceiver(config)); // throws SaxonApiException
  pf.setResult(new SAXResult(rch));
  s.send(input.asSource(), chp); // throws XPathException
  input= null;  rch= null;  chp= null;  s= null;
  return result.getXdmNode();
}
```

In order to create an equivalent pipeline component for Calabash the s9api code above was easily adapted to create an extension step with the declaration:

```
<p:declare-step type="dx:java-filter">
  <p:input port="source" primary="true" kind="document" sequence="false"/>
  <p:input port="parameters" kind="parameter"/>
  <p:output port="result" primary="true" sequence="false"/>
  <p:option name="classname" required="true" cx:type="xs:string"/>
</p:declare-step>
```

Future directions

Using temporary trees and modes it may be possible to merge filters together in a single transformation. This may save time, but we no longer have explicit control of the garbage collection of the intermediate trees. Another issue is that when existing filters already use modes, the design of an automatic merging process becomes slightly harder.

The `saxon:next-in-chain` attribute can be used to create pipelines. One slight dis- advantage is that it hardwires the pipeline structure into the filters and has slight reusability implications (a filter may be reused in different pipelines and may be followed by different filters).

For very simple filters we are doing a lot of serialization and tree-building. Most of the time is spent running the 'identity template' in many filters. Perhaps we can instrument and quantify how much time is spent running this filter as opposed to other filters. If it is substantial, we could consider using a persistent in memory data structure (perhaps a form of DOM with XPath support) and then have Java methods for modifying and updating it. We prefer the XSLT processing model and DOM trees can consume a lot of memory; perhaps XQuery Update could be used as an alternative approach to replace one or more filter stages.

JVM Support

The java.lang.management interface provides access to JVM telemetry information, including information about the various Java heap space 'generations'. Information is also available about total and thread CPU times. We are making use of these APIs in our code and this will then allow more informed experimentation with different garbage collectors and their command-line options.

The VisualVM introduced in 1.6.0_7 and earlier command-line tools such as jstat and jmap are also useful when investigating pipelines and garbage collector behaviour.

```
ThreadMXBean mx= ManagementFactory.getThreadMXBean();
MemoryMXBean mb= ManagementFactory.getMemoryMXBean();
long start= System.nanoTime();
long startCPU= mx.getCurrentThreadCpuTime();

System.out.println("Memory:  " + mb.getHeapMemoryUsage().getUsed() +
                   "/" + mb.getHeapMemoryUsage().getMax());
```