IBM Software Group

WebSphere DataPower SOA Appliances and XSLT (Part 1 of 2) - Tools

Hermann Stamm-Wilbrandt (stammw@de.ibm.com)
DataPower XML Compiler Developer, L3
6 July 2010

ON DEMAND BUSINESS™

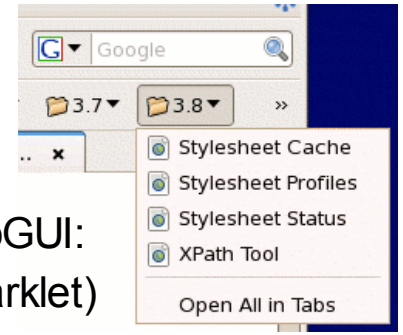
Agenda

DataPower tools

- **built-in**
- others

Appendix

Bookmarking DataPower screens for easy access



- Multibox bookmarklets – they avoid navigation through left navbar in WebGUI:
  - Stylesheet Cache (3.7.x bookmarklet / 3.8.0 bookmark / 3.8.0 bookmarklet)
    - javascript:window.mainFrame.genericStatusRequest('StylesheetCachingSummary',%20'genericStatus')
    - https://yourbox:webguiport/status/StylesheetCachingSummary
    - javascript:void(location=window.location.href.replace(/(^[^:]*:\/\/[^/]*).*$/g,'$1')+'/status/StylesheetCachingSummary')
  - Xpath Tool (3.7.x bookmark / 3.8.0 bookmark / 3.8.0 bookmarklet)
    - .../TaskTemplates/SelectXPath.xml?...
    - https://boxip:webguiport/service/SelectXPath?popup=true&newObjPopup=true&newObjPopupInput=input_XPath&step=get-started&popuplocation=
    - javascript:void(window.open(window.location.href.replace(/(^[^:]*:\/\/[^/]*).*$/g,'$1')+'/service/SelectXPath?popup=true&newObjPopup=true&newObjPopupInput=input_XPath&step=get-started&popuplocation=').focus())
  - Works for FF, IE, Chrome and Safari browsers

Source: Bookmarklets and WebGUI posting on developerWorks DataPow er Forum

Stylesheet profiling (1/2)

- Configure profiling of a stylesheet by
  - a compile option policy (for its XML Manager)
  - and a matching “Profile Rule” there.
- Sending requests against the service containing the stylesheet gives profiling information.
- This may be looked up in WebGUI under “Status→XML Processing→Stylesheet Profiles”
- Flushing the XML Manager resets the profile information.
- <dp:profile> allows to define profiling regions

manager	Output Mode	URL	Name	Location	Count	Type	Time(ms)
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	match="/"	local:///xyz.xsl:17	100	template	1401
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	name="calc-offset"	local:///xyz.xsl:26	4300	template	7938
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	name="build-token-array"	local:///xyz.xsl:54	2500	template	13260
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	name="process-delimiter"	local:///xyz.xsl:78	2500	template	13069
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	name="find-end-delimiter-index-from-root"	local:///xyz.xsl:143	400	template	5537
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	name="hex-to-text"	local:///xyz.xsl:184	23900	template	57835
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	local:///hexWrapper.ffd:4	built-in	100	template	662
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	Lazy Template for [GV40]:	local:///xyz.xsl:5	100	template	187
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	input-message	local:///xyz.xsl:5	100	global var	172
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	Lazy Template for [GV41]:	local:///xyz.xsl:6	100	template	2626
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	offsets	local:///xyz.xsl:6	100	global var	24671
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	Lazy Template for [GV42]:	local:///xyz.xsl:16	100	template	79
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	delimiter-count	local:///xyz.xsl:16	100	global var	59
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	Init Template	built-in	100	template	687

Source: Appendix D. Compile Options Policy configuration/Profiling overview (`CommandReference`)

Stylesheet profiling

```
<xsl:template name="hex-to-text">
  <xsl:param name="str"/>
  <xsl:variable name="raw"
    select="dp:binary-decode(dp:radix-convert($str,16,64))"/>
  <xsl:value-of select="$raw"/>
</xsl:template>
```

- Sample of improvement (replace pure XSLT implementation of “hex-to-text” by an implementation making use of extension functions -- template hex-to-text gets inlined)

manager	Output Mode	URL	Name	Location	Count	Type	Time(ms)
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	match="/"	local:///xyz.xsl:17	100	template	1401
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	name="calc-offset"	local:///xyz.xsl:26	4300	template	7938
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	name="build-token-array"	local:///xyz.xsl:54	2500	template	13260
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	name="process-delimiter"	local:///xyz.xsl:78	2500	template	13069
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	name="find-end-delimiter-index-from-root"	local:///xyz.xsl:143	400	template	5537
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	name="hex-to-text"	local:///xyz.xsl:184	23900	template	57835
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	local:///hexWrapper.ffd:4	built-in	100	template	662
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	Lazy Template for [GV40]:	local:///xyz.xsl:5	100	template	187
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	input-message	local:///xyz.xsl:5	100	global var	172
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	Lazy Template for [GV41]:	local:///xyz.xsl:6	100	template	2626
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	offsets	local:///xyz.xsl:6	100	global var	24671
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	Lazy Template for [GV42]:	local:///xyz.xsl:16	100	template	79
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	delimiter-count	local:///xyz.xsl:16	100	global var	59
profile	Stream	xslt-p-s-f-M://local:///xyz.xsl	Init Template	built-in	100	template	687

manager	Output Mode	URL	Name	Location	Count	Type	Time(ms)
profile	Stream	xslt-p-s-f-M://local:///abc.xsl	match="/"	local:///abc.xsl:17	100	template	1342
profile	Stream	xslt-p-s-f-M://local:///abc.xsl	name="calc-offset"	local:///abc.xsl:26	4300	template	7847
profile	Stream	xslt-p-s-f-M://local:///abc.xsl	name="build-token-array"	local:///abc.xsl:54	2500	template	12557
profile	Stream	xslt-p-s-f-M://local:///abc.xsl	name="process-delimiter"	local:///abc.xsl:78	2500	template	20638
profile	Stream	xslt-p-s-f-M://local:///abc.xsl	name="find-end-delimiter-index-from-root"	local:///abc.xsl:143	400	template	5038
profile	Stream	xslt-p-s-f-M://local:///abc.xsl	local:///hexWrapper.ffd:4	built-in	100	template	643
profile	Stream	xslt-p-s-f-M://local:///abc.xsl	Lazy Template for [GV40]:	local:///abc.xsl:5	100	template	185
profile	Stream	xslt-p-s-f-M://local:///abc.xsl	input-message	local:///abc.xsl:5	100	global var	169
profile	Stream	xslt-p-s-f-M://local:///abc.xsl	Lazy Template for [GV41]:	local:///abc.xsl:6	100	template	2428
profile	Stream	xslt-p-s-f-M://local:///abc.xsl	offsets	local:///abc.xsl:6	100	global var	23687
profile	Stream	xslt-p-s-f-M://local:///abc.xsl	Lazy Template for [GV42]:	local:///abc.xsl:16	100	template	73
profile	Stream	xslt-p-s-f-M://local:///abc.xsl	delimiter-count	local:///abc.xsl:16	100	global var	52
profile	Stream	xslt-p-s-f-M://local:///abc.xsl	Init Template	built-in	100	template	598

Stylesheet tracing

- Configure tracing of a stylesheet by
  - a compile option policy (for its XML Manager)
  - and a matching “Debug Rule” there.
- Sending a request against the service containing the stylesheet returns a trace HTML page.
- If needing a (less informative) trace facility not affecting MultiStep processing you may want to have a look at trace.xslt by Oliver Becker.

I found "trace.xslt" on Oliver Becker's homepage:
http://www2.informatik.hu-berlin.de/~obecker/XSLT/#trace

I tried it -- and it seems to work!
Basically "trace.xslt" transforms a stylesheet to another one which then produces the trace output by <xsl:message> statements.

So how get this working in DataPower?
As an example I created a XML FW with just a match and xform action.
For transformation I used attached complicated identity xform id.xsl (it is more complicate than needed in order to demonstrate the tracing feature).

Now enabling tracing is done by:
1) add fetch action after the match fetching "local:///id.xsl", output PIPE
2) add xform action after fetch, input PIPE, output context "trace" and "local:///trace.xslt" xform
3) apply all changes for the policy and the XML FW
4) Under "Objects-->XML Processing-->Processing Action" open the non-added xform action
5) Replace "local:///id.xsl" by "trace" in "Transform" field and apply.

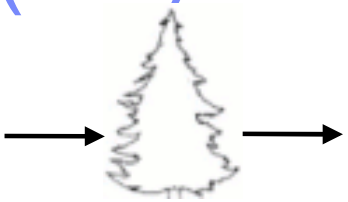
That's all, after setting log-level to "information" sending

Source: CommandReference / trace.xslt by Oliver Becker

Debug Output Trace

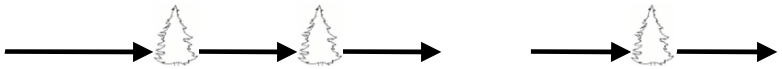
file	line	position()	info
CommonPrefix.xsl	58	1	Template entered: match="/"
CommonPrefix.xsl	59	1	<out
CommonPrefix.xsl	6	1	Template entered: name="CommonPrefix"
CommonPrefix.xsl	7	1	Variable 'a' defined as a string with value 'abcdefghijklmnopqrstuvwxyz'
CommonPrefix.xsl	8	1	Variable 'b' defined as a string with value 'abcdefghijklmnopqrstuvwxy0'
CommonPrefix.xsl	10	1	Variable 'A' defined as a number with value 26
CommonPrefix.xsl	11	1	Variable 'B' defined as a number with value 26
CommonPrefix.xsl	29	1	Template entered: name="CommonPrefixRec"
CommonPrefix.xsl	30	1	Variable 'a' defined as a string with value 'abcdefghijklmnopqrstuvwxyz'
CommonPrefix.xsl	31	1	Variable 'b' defined as a string with value 'abcdefghijklmnopqrstuvwxy0'
CommonPrefix.xsl	33	1	Variable 'mid' defined as a number with value 13.000000
CommonPrefix.xsl	34	1	Variable 'a1' defined as a string with value 'abcdefghijklm'
CommonPrefix.xsl	35	1	Variable 'b1' defined as a string with value 'abcdefghijklm'
CommonPrefix.xsl	41	1	>abcdefghijklm
CommonPrefix.xsl	29	1	Template entered: name="CommonPrefixRec"
CommonPrefix.xsl	30	1	Variable 'a' defined as a string with value 'nopqrstuvwxyz'
CommonPrefix.xsl	31	1	Variable 'b' defined as a string with value 'nopqrstuvwxy0'
CommonPrefix.xsl	33	1	Variable 'mid' defined as a number with value 6.000000
CommonPrefix.xsl	34	1	Variable 'a1' defined as a string with value 'nopqrs'
CommonPrefix.xsl	35	1	Variable 'b1' defined as a string with value 'nopqrs'
CommonPrefix.xsl	41	1	nopqrs
CommonPrefix.xsl	29	1	Template entered: name="CommonPrefixRec"
CommonPrefix.xsl	30	1	Variable 'a' defined as a string with value 'tuvwxyz'
CommonPrefix.xsl	31	1	Variable 'b' defined as a string with value 'tuvwxy0'
CommonPrefix.xsl	33	1	Variable 'mid' defined as a number with value 3.000000
CommonPrefix.xsl	34	1	Variable 'a1' defined as a string with value 'tuv'
CommonPrefix.xsl	35	1	Variable 'b1' defined as a string with value 'tuv'
CommonPrefix.xsl	41	1	tuv
CommonPrefix.xsl	29	1	Template entered: name="CommonPrefixRec"
CommonPrefix.xsl	30	1	Variable 'a' defined as a string with value 'wxyz'
CommonPrefix.xsl	31	1	Variable 'b' defined as a string with value 'wxy0'
CommonPrefix.xsl	33	1	Variable 'mid' defined as a number with value 2.000000
CommonPrefix.xsl	34	1	Variable 'a1' defined as a string with value 'wx'
CommonPrefix.xsl	35	1	Variable 'b1' defined as a string with value 'wx'
CommonPrefix.xsl	41	1	wx
CommonPrefix.xsl	29	1	Template entered: name="CommonPrefixRec"
CommonPrefix.xsl	30	1	Variable 'a' defined as a string with value 'yz'
CommonPrefix.xsl	31	1	Variable 'b' defined as a string with value 'y0'
CommonPrefix.xsl	33	1	Variable 'mid' defined as a number with value 1.000000
CommonPrefix.xsl	34	1	Variable 'a1' defined as a string with value 'y'
CommonPrefix.xsl	35	1	Variable 'b1' defined as a string with value 'y'
CommonPrefix.xsl	41	1	y
CommonPrefix.xsl	29	1	Template entered: name="CommonPrefixRec"
CommonPrefix.xsl	30	1	Variable 'a' defined as a string with value 'z'
CommonPrefix.xsl	31	1	Variable 'b' defined as a string with value '0'
CommonPrefix.xsl	33	1	Variable 'mid' defined as a number with value 0.000000
CommonPrefix.xsl	34	1	Variable 'a1' defined as a string with value ''
CommonPrefix.xsl	35	1	Variable 'b1' defined as a string with value ''
CommonPrefix.xsl	41	1	</out>

DataPower XML data processing (1/4)



- Document Object Model (DOM)
  - Reads all of the XML data and forms the corresponding tree in memory.
  - After building the tree, the application program processes the data.
  - Processing consists of one or more traversals of the data in arbitrary order.
  - As it performs these traversals, the application program generates the output.

- Simple API for XML model (SAX)
  - Combines the reading of the data with the processing.
  - Reading the nodes the application processes them and then deletes its storage.
  - In other words, the tree is not built. Instead, the data is processed on-the-fly.
  - All schema validation in the DataPower appliance uses the SAX model.



- Streaming model
  - Hybrid technique, the application starts processing in SAX model manner.
  - If needed the application program builds trees for that part of the data as the DOM.
  - After processing of this tree completes it is immediately removed.
  - Flexibility of the DOM model with some of the storage advantages of the SAX model.

Source: OptimizingThroughStreaming-v1.pdf

Streaming (2/4)

- This stylesheet is streamable.
- The commenting out was necessary (double use of “.”).
- You can send documents containing millions of records through the box without noticeable memory increase.

```
<?xml version="1.0" encoding="UTF-8"?>
<document>id=0000
firstname=Al
lastname=Aranow
street=1 Any St.
city=Anytown
state=AL
zip=22000

id=0001
...
...
id=999999
firstname=James
lastname=Jones
street=100 Any St.
city=Anytown
state=WY
zip=22199

</document>
```

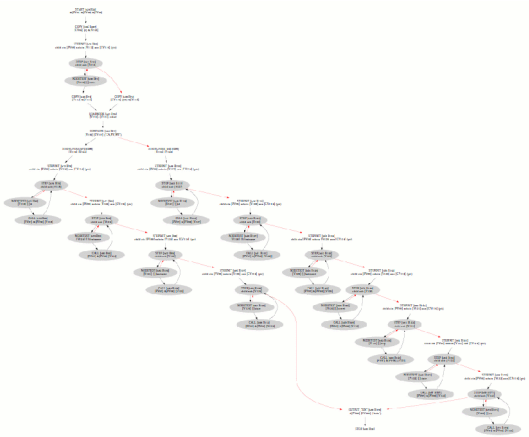
```
<document>0000
Al
Aranow
1 Any St.
Anytown
AL
22000

0001
...
...
999999
James
Jones
100 Any St.
Anytown
WY
22199

</document>
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
  <xsl:template match="table">
    <document>
      <xsl:apply-templates select="row"/>
    </document>
  </xsl:template>

  <xsl:template match="row">
    <xsl:apply-templates select="id"/>
    <xsl:apply-templates select="firstname"/>
    <xsl:apply-templates select="lastname"/>
    <xsl:apply-templates select="street"/>
    <xsl:apply-templates select="city"/>
    <xsl:apply-templates select="state"/>
    <xsl:apply-templates select="zip"/>
    <xsl:text>&#x0A;</xsl:text>
  </xsl:template>

  <xsl:template match="id|firstname|lastname|street|city|state|zip">
  <!--
    <xsl:value-of select="name(.)"/>
    <xsl:text>=</xsl:text>
  -->
    <xsl:value-of select="."/>
    <xsl:text>&#x0A;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Source: Find stylesheet and script for generating large input data in appendix

Partial Streaming (3/4)

- This template is not streamable SAX like.
- Reason is the <xsl:choose> statement.
- But the stylesheet is partially streamable.
- A single record is processed DOM like.
- After processing the memory is given back immediately.
- Again you can send documents containing millions of records through the box without noticeable memory increase



```
...
  <xsl:templ
    <xsl:choose>
      <xsl:when test="contains('CA,NY,WY',state)">
        <xsl:apply-templates select="id"/>
        <xsl:apply-templates select="firstname"/>
        <xsl:apply-templates select="lastname"/>
        <xsl:apply-templates select="state"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates select="id"/>
        <xsl:apply-templates select="firstname"/>
        <xsl:apply-templates select="lastname"/>
        <xsl:apply-templates select="street"/>
        <xsl:apply-templates select="city"/>
        <xsl:apply-templates select="state"/>
        <xsl:apply-templates select="zip"/>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:text>&#x0A;</xsl:text>
  </xsl:template>
...
```

Source: Find stylesheet and script for generating large input data in appendix

Determine why a stylesheet is not streamable (4/4)

- A stylesheet will only be processed streaming if
  - a compile option policy is configured for its XML Manager,
  - a matching “Streaming Rule” or “Attempt Streaming Rule” is configured there
  - and the stylesheet itself is streamable.
- In order to determine whether a stylesheet is streamable
  - use matching “Streaming Rule” (to enforce streaming mode)
  - send any traffic against a service containing an action using the stylesheet
  - in case the stylesheet is not streamable you will get a generic error in the log.
- For stylesheets not being streamable find the reason by
  - switch to “Status→XML Processing→Stylesheet Status” in WebGUI
  - inspect stylesheet's entry
  - lookup column “Compiler Log” for detailed errors

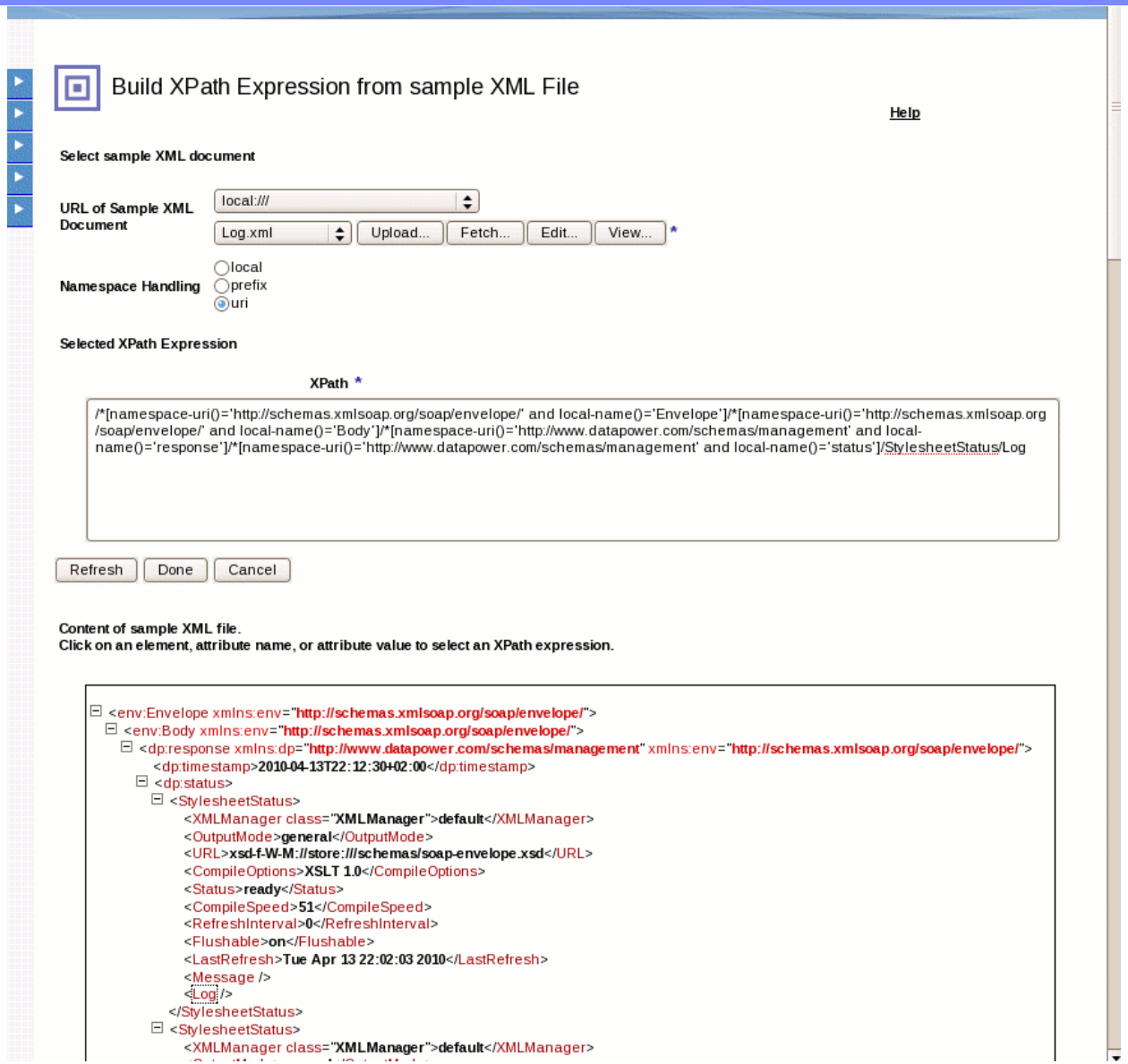
force	Stream	xslt-s-m-f-M-X://local:///number.xsl	XSLT 1.0, failed to stream input	Ready	2	0	on	Mon Apr 12 17:50:54 2010		Error at local:///number.xsl:5: xsl:number not streamable.

XPath Tool

- In WebGUI's left navbar click on "Objects->XML Processing->XPath Routing Map"
- Click on "Add" and select the "Rules" tab
- Click "Add" again, a popup opens
- Click "XPath Tool" in the popup, "Xpath Tool" popup opens.

- Or use “bookmark”

- Upload XML file
- Click somewhere (eg. Log)
- Inspect generated Xpath and take it as a start



Agenda

DataPower tools

- built-in
- **others**

Appendix

Using xpath tool to gain information on XML and XSL files

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dyn="http://exslt.org/dynamic"
  extension-element-prefixes="dyn"
>
  <xsl:output method="xml" omit-xml-declaration="yes"/>
  <xsl:template match="/">
    <xsl:copy-of select="dyn:evaluate(string((//comment())[last()]))"/>
  </xsl:template>

</xsl:stylesheet>
```

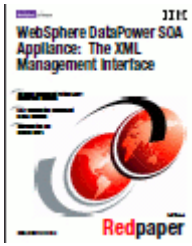
- $ xpath "string(//*[name()='xsl:copy-of']/@select)" xpath.xsl
  dyn:evaluate(string((//comment())[last()]))
  $
  → get insight in stylesheet

- $ xpath "count(//*[name()='xsl:for-each']//*[name()='xsl:for-each']//*[name()='xsl:for-each']//*[name()='xsl:for-each']//*[name()='xsl:for-each']//*[name()='xsl:for-
  …
  each']//*[name()='xsl:for-each']//*[name()='xsl:for-each']//*[name()='xsl:for-each']//*[name()='xsl:for-each']//*[name()='xsl:for-each']//*[name()='xsl:for-each'])" HugeXslFor-each.xsl
  4
  $
  → uncovering (structural) performance issues in stylesheets

- xpath "/*/*/*[name()='dp:response']/*[name()='dp:status']/StylesheetStatus/Log[text()]" …
  → returns all warnings/errors retrieved by get-status SOMA call for compiled stylesheets

Source: xpath tool posting on developerWorks DataPow er Forum

Scripting DataPower



- DataPower Scripting is preferably done over the XML Management Interface
- This is a really good RedPaper on this topic:
  WebSphere DataPower SOA Appliance: The XML Management Interface
  Rolf Wittich
  http://www.redbooks.ibm.com/abstracts/redp4446.html
- Internal CLI scripting is possible, too:
  - store a CLI script eg. as “local:///script.cfg”
  - open a CLI session to the box
  - do “exec local:///script.cfg”
- External CLI scripting can be done eg. with “netcat” command (nc, Linux®/Cygwin)
  - have Telnet service enabled (on intranet interface)
  - store sequence of entries of a CLI session into eg. cli.txt
  - execute “nc yourbox telnetport < cli.txt”
- Executing CLI commands from within a stylesheet is possible (posting link below)
  - While DataPower currently has no raw TCP support on the front side it HAS support for raw TCP on the back side, the protocol "tcp:" (→ Telnet service)

```
$ cat cli.txt
stammw
geheim
default
show mem
show cpu
exit
$
```

Source: Executing CLI commands from within a stylesheet posting on developerWorks DataPower Forum

Processing UTF-8 URL-encoded URIs with WebSphere DataPower

Output of below stylesheet in Multi-step Probe:

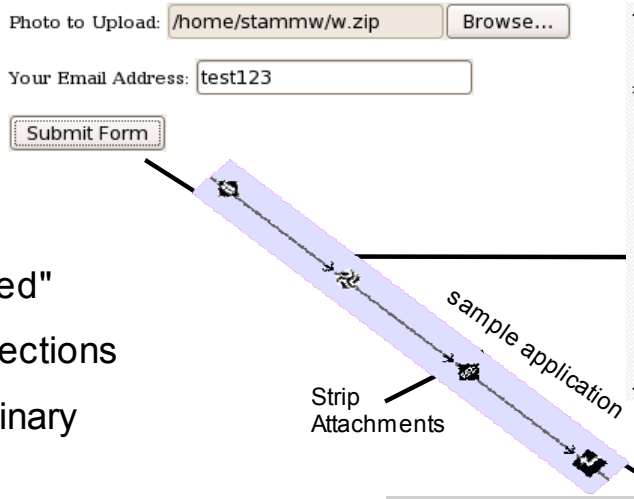
```
<request xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:dp="http://www.datapower.com/schemas/management">
<url>/utf8uri?danish=%C3%86-%C3%98-%C3%85&french=%C5%92-%C3%A6&
german=%C3%84-%C3%96-%C3%9C-%C3%9F&spanish=%CA%A7-%EA%9D
%86-%C3%91</url>
<base-url>/utf8uri</base-url>
<args src="url">
<arg name="danish">Æ-Ø-Å</arg>
<arg name="french">Œ-æ</arg>
<arg name="german">Ä-Ö-Ü-ß</arg>
<arg name="spanish">ʧ-ʩ-Ñ</arg></args></request>
```

- DataPower action "convert-http"
  - creates XML document for HTTP form data
  - converts non-XML query parameters to XML
  - assumption is ISO-8859-1 encoding
- Workaround for UTF-8, see technote referenced at bottom
- Main aspects of workaround
  - Accessing the URL
    - … select="dp:variable('var://service/URI')" /> …
  - Tokenizing the query
    - <xsl:for-each select="str:tokenize(substring-after($url,'?'),'&')">
  - Decoding the UTF-8 URL-encoded data
    - … select="str:decode-uri(substring-after(.,'='))" …
- DataPower provides lots of Extension functions/elements, you may want to have a look into the Extension Functions Catalog …

Source: Processing UTF-8 URL-encoded URIs w ith WebSphere DataPow er technote

Enhancing “near” SOAP With Attachment data

- SWA
  (Soap With Attachments)
  - MIME boundaries
  - content type "multipart/related"
  - Content-ID for each/most sections
  - arbitrary content including binary
  - first section SOAP
- HTTP form data (HTTP POST)
  - content type "multipart/form-data"
  - Content-Disposition for sections
- swaform tool
  - Non-XML request type policy
  - replaces "Content-Type" multipart/form-data by multipart/related
  - replaces 'Content-Disposition ... name="xyz" ...' by 'Content-Id: <xyz>'
  - prepends root section with "dummy" SOAP message




```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dp="http://www.datapower.com/extensions"
  extension-element-prefixes="dp"
>
  <xsl:output method="xml"/>

  <xsl:template match="/">
    <attachments>
      <photo><dp:url-open target="cid:photo?Encode=hexbin" /></photo>

      <xsl:variable name="email">
        <dp:url-open target="cid:email_address?Encode=base64" />
      </xsl:variable>
      <xsl:variable name="raw" select="dp:binary-decode($email)" />
      <email_address><xsl:value-of select="$raw"/></email_address>

      <Submit><dp:url-open target="cid:Submit?Encode=base64" /></Submit>
    </attachments>
  </xsl:template>
</xsl:stylesheet>
```

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <attachments>
  - <photo>
    - <hexbin>
        504b0304140000000080009824d3b651495e1ad010000a9030000080015007772617 02e747874555409C
      </hexbin>
    </photo>
    <email_address>test123</email_address>
  - <Submit>
      <base64>U3VibWl0IEZvcm0=</base64>
    </Submit>
  </attachments>
```

Source: swaform -- using HTTP forms as generator for "Soap With Attachments" posting on developerWorks DataPower Forum

Processing embedded stylesheets

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="#style1"?>
<!DOCTYPE catalog [
<!ATTLIST xsl:stylesheet id ID #REQUIRED>
]>
<catalog>
<xsl:stylesheet id="style1" version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
```

- Not many customers are using "Transform (Using Processing Instruction)" at all.
- DataPower does not support processing of "embedded stylesheets" out of the box.

- Use an "Extract Using XPath" action to extract the stylesheet into a context "ctx".
- The XPATH to use is
- //*[attribute::*[local-name()='id']=substring-before(substring-after(/processing-instruction()[local-name()='xml-stylesheet'],'href="#'),'")]
- Be careful with the single and double quotes.
- It extracts the stylesheet by the id referenced by href in xml-stylesheet processing instruction into context "ctx".
- Next use a xform action reading from INPUT with processing control file "ctx".
- It is not possible to create this in WebGUI directly.
- After having selected any xform action (eg. store://identity.xsl) you have to
  - edit the action in CLI and execute "transform ctx" to make the xform action take the stylesheet from context "ctx"
  - or via WebGUI using Objects menu, select Objects→XML Processing→Processing Action, the Transform is a text field in this page and you input "ctx" there.

Source: 2.7 Embedding Stylesheets (XSLT 1.0 spec) / cdcatalog.xml

Agenda

DataPower tools

- built-in
- others

Appendix

Appendix (for copy&paste)

dbgen.pl

```
#!/usr/bin/perl

$size = shift;

if ($size eq "")
{
    die "usage:  dbgen.pl [size]\n";
}

@firstnames = ("Al", "Bob", "Charles", "David",
"Egon", "Farbood",
               "George", "Hank", "Inki", "James");
@lastnames = ("Aranow", "Barker", "Corsetti",
"Dershowitz", "Engleman",
              "Franklin", "Grice", "Haverford",
"Ilvedson", "Jones");
@states = ("AL", "AK", "AZ", "AR", "CA", "CO", "CT",
"DE", "FL", "GA",
           "HI", "ID", "IL", "IN", "IA", "KS", "KY",
"LA", "ME", "MD",
           "MA", "MI", "MN", "MS", "MO", "MT", "NE",
"NV", "NH", "NJ",
           "NM", "NY", "NC", "ND", "OH", "OK", "OR",
"PA", "RI", "SC",
           "SD", "TN", "TX", "UT", "VT", "VA", "WA",
"WV", "WI", "WY");

print "<?xml version=\"1.0\"?>\n";
print "\n";
print "<table>\n";

for ($i=0; $i<$size; $i++)
{
    $first = $firstnames [$i % 10];
    $last = $lastnames [($i / 10) % 10];
    $state = $states [($i / 100) % 50];
    $zip = 22000 + $i / 5000;

    printf "  <row>\n";
    printf "    <id>%04d</id>\n", $i;
    printf "    <firstname>$first</firstname>\n", $i;
    printf "    <lastname>$last</lastname>\n", $i;
    printf "    <street>%d Any St.</street>\n", ($i %
100) + 1;
    printf "    <city>Anytown</city>\n";
    printf "    <state>$state</state>\n";
    printf "    <zip>%d</zip>\n", $zip;
    printf "  </row>\n";
}

print "</table>\n";
```

dbsql.xsl

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

<xsl:template match="table">
  <document>
    <xsl:apply-templates select="row"/>
  </document>
</xsl:template>

<xsl:template match="row">
  <xsl:apply-templates select="id"/>
  <xsl:apply-templates select="firstname"/>
  <xsl:apply-templates select="lastname"/>
  <xsl:apply-templates select="street"/>
  <xsl:apply-templates select="city"/>
  <xsl:apply-templates select="state"/>
  <xsl:apply-templates select="zip"/>
  <xsl:text>&#x0A;</xsl:text>
</xsl:template>

<xsl:template match="id|firstname|lastname|
street|city|state|zip">
  <xsl:value-of select="."/>
  <xsl:text>&#x0A;</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

dbsql1.pl

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
  <xsl:template match="table">
    <document>
      <xsl:apply-templates select="row"/>
    </document>
  </xsl:template>

  <xsl:template match="row">
    <xsl:choose>
      <xsl:when test="contains('CA,NY,WY',state)">
        <xsl:apply-templates select="id"/>
        <xsl:apply-templates select="firstname"/>
        <xsl:apply-templates select="lastname"/>
        <xsl:apply-templates select="state"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates select="id"/>
        <xsl:apply-templates select="firstname"/>
        <xsl:apply-templates select="lastname"/>
        <xsl:apply-templates select="street"/>
        <xsl:apply-templates select="city"/>
        <xsl:apply-templates select="state"/>
        <xsl:apply-templates select="zip"/>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:text>&#x0A;</xsl:text>
  </xsl:template>

  <xsl:template match="id|firstname|lastname|street|
city|state|zip">
    <xsl:value-of select="."/>
    <xsl:text>&#x0A;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Summary

- Built-in DataPower tools (Stylesheet profiling and tracing, XPath tool)
- DataPower XML Data Processing (streaming, partial streaming)
- Other DataPower tools (xpath tool, scripting, processing UTF-8 URL-encoded URIs, Enhancing “near” SOAP With Attachment data, processing embedded stylesheets)

- Next webcast (2/2) will be on “Tips and Tricks”

Additional WebSphere Product Resources

- Learn about upcoming WebSphere Support Technical Exchange webcasts, and access previously recorded presentations at:
  http://www.ibm.com/software/websphere/support/supp_tech.html
- Discover the latest trends in WebSphere Technology and implementation, participate in technically-focused briefings, webcasts and podcasts at:
  http://www.ibm.com/developerworks/websphere/community/
- Join the Global WebSphere User Group Community:
  http://www.websphere.org
- Access key product show-me demos and tutorials by visiting IBM® Education Assistant:
  http://www.ibm.com/software/info/education/assistant
- View a webcast replay with step-by-step instructions for using the Service Request (SR) tool for submitting problems electronically:
  http://www.ibm.com/software/websphere/support/d2w.html
- Sign up to receive weekly technical My Notifications emails:
  http://www.ibm.com/software/support/einfo.html

We Want to Hear From You!

Tell us about what you want to learn

Suggestions for future topics
Improvements and comments about our webcasts
We want to hear everything you have to say!

Please send your suggestions and comments to:
wsehelp@us.ibm.com

Questions and Answers